Net-Centric Implementation Framework

Part 1: Overview

Part 2: ASD(NII) Checklist Guidance

Part 3: Migration Guidance

Part 4: Node Guidance

Part 5: Developer Guidance

Part 6: Contracting Guidance for Acquisition

V 2.1.0

12 October 2007

Net-Centric Enterprise Solutions for Interoperability (NESI) is a collaborative activity of the USN Program Executive Office for Command, Control, Communications, Computers and Intelligence (PEO C4I); the USAF Electronic Systems Center (ESC); and the Defense Information Systems Agency (DISA).

Approved for public release; distribution is unlimited.

NESI-X Subdocument generated using **View, NESI Part 5: Developer Guidance**
Generated: **Wed Oct 31 18:19:08 PDT 2007**
NESI-X Version: **v1.0.0 build 1819 - 2007/08/23 15:26**

Table of Contents

Perspectives 6

NESI Overview 7

NESI Part 5: Developer Guidance 10

Technical Guidance and Tactics 11

Public Interface Design 12

Publish and Insulate Public Interfaces 17

Automate the Software Build Process 18

Standard Interface Documentation 19

Implement a Component-Based Architecture 22

Presentation Tier 23

Human-Computer Interaction 25

Designing User Interfaces for Internationalization 27

Human Factor Considerations for Web-Based User Interfaces 28

Designing User Interfaces for Accessibility 32

Browser-Based Clients 33

XML Rendering 34

Active Server Pages (ASP) 35

Active Server Pages for .NET (ASP.NET) 36

Java Server Pages (JSP) 37

Style Sheets 38

Web Portals 39

Thick Clients 41

Middle Tier 42

Messaging 44

Message-Oriented Middleware (MOM) 45

Message-Based Applications 47

Messaging with MSMQ 53

Web Services 54

Web Services with .NET 58

SOAP 59
WS-I Compliance 64
WSDL 65
Insulation and Structure 66
Error Handling 67
Universal Description, Discovery, and Integration (UDDI) 71
Java EE Environment 73
.NET Framework 76
CORBA 78
Software Communication Architecture 81
Data Distribution Service (DDS) 82
Decoupling Using DDS and Publish-Subscribe 88
DDS Quality of Service 90
DDS Data-Centric Publish-Subscribe (DCPS) 92
DDS Domains - Global Data Spaces 94
Reading/Writing Objects within a DDS Domain 96
Messaging within a DDS Domain 98
DDS Data Local Reconstruction Layer (DLRL) 100
Data Tier 101
Decouple from Applications 102
Database Implementations 103
Database Development 104
RDBMS Internals 105
Overarching Concepts 107
Data 108
XML 110
XML Syntax 111
XML Semantics 112
XML Instance Documents 113
XML Schema Documents 114
Using XML Substitution Groups 115
XML Schema Files 117

Defining XML Schemas 118
Using XML Namespaces 119
Versioning XML Schemas 120
Defining XML Types 121
XML Processing 122
XPath 123
XSLT 124
XML Validation 126
Parsing XML 127
Metadata Registry 128
Data Modeling 131
ASD(NII) Checklist 133
Metadata 152
Application Security 154
General Application Security 156
Public Key Infrastructure (PKI) and PK Enable Applications 158
Key Management 160
Encryption Services 161
Certificate Processing 162
Security Assertion Markup Language (SAML) 164
Desktop Computing 165
API Security 166
Java Security 167
Application Resource Security 168
Network Computing 169
Enterprise Computing 170
JNDI Security 172
Data Tier 173
RDBMS Security 174
LDAP Security 175
XML Web Service Security 176
Programming Languages 179

C++ 180
C++ Operator Overloading 181
C++ Namespaces and Modules 182
C++ Header Files 183
VHDL 184
VHDL Coding and Design 185
VHDL Synchronous Design 186
VHDL Synthesizable Design 187
VHDL Testbench 188
Guidance and Best Practice Details 189
Glossary 610
References 655

Perspectives

P1117: NESI Overview

Net-Centric Enterprise Solutions for Interoperability (***NESI***) provides, for all phases of the acquisition of net-centric solutions, actionable guidance that meets DoD Network-Centric Warfare goals. The guidance in NESI is derived from the higher level, more abstract concepts provided in various directives, policies and mandates such as the Net-Centric Operations and Warfare Reference Model (NCOW RM) [R1176] and the ASD(NII) Net-Centric Checklist [R1177] . As currently structured, NESI implementation covers architecture, design and implementation; compliance checklists; and a collaboration environment that includes a repository.

More specifically, NESI is a body of architectural and engineering knowledge that guides the design, implementation, maintenance, evolution, and use of the Information Technology (IT) portion of net-centric solutions for military application. NESI provides specific technical recommendations that a DoD organization can use as references. Stated another way, NESI serves as a reference set of compliant instantiations of these directives.

NESI is derived from a studied examination of enterprise-level needs and, more importantly, from the collective practical experience of recent and on-going program-level implementations. It is based on today's technologies and probable near-term technology developments. It describes the practical experience of system developers within the context of a minimal top-down technical framework. Most, if not all, of the guidance in NESI is in line with commercial best practices in the area of enterprise computing.

NESI applies to all phases of the acquisition process as defined in DoD Directive 5000.1 [R1164] and DoD Instruction 5000.2 [R1165] and to both new and legacy programs. NESI provides explicit counsel for building in net-centricity from the ground up and for migrating legacy systems to greater degrees of net-centricity.

NESI subsumes a number of references and directives; in particular, the Air Force C2 Enterprise Technical Reference Architecture (C2ERA) and the Navy Reusable Applications Integration and Development Standards (RAPIDS). Initial authority for NESI is per the Memorandum of Agreement between Commander, Space and Naval Warfare Systems Command (SPAWAR); Navy Program Executive Officer, C4I & Space (now PEO C4I); and the United States Air Force Electronic Systems Center (ESC), dated 22 December 2003, Subject: Cooperation Agreement for Net-Centric Solutions for Interoperability (NESI). The Defense Information Systems Agency (DISA) formally joined the NESI effort in 2006.

Content Structure

Perspective	NESI ***Perspectives*** describe a topic and encompass related, more specific Perspectives or encapsulate a set of Guidance and Best Practice details, Examples, References, and Glossary entries that pertain to the topic.
Guidance	NESI ***Guidance*** is in the form of atomic, succinct, absolute and definitive Statements related to one or more Perspectives. Each Guidance Statement is linked to Guidance Details which amplifying Rationale, relationships with other Guidance or Best Practices, and Evaluation Criteria with one or more Tests, Procedures and Examples which facilitate validation of using the Guidance through observation, measurement or other means. Guidance Statements are intended to be binding in nature, especially if used as part of a Statement of Work (SOW) or performance specification.

Best Practices	NESI ***Best Practices*** are advisory in nature to assist program or project managers and personnel. Best Practice Details can have all the same parts as NESI Guidance. The use of NESI Best Practices are at the discretion of the program or project manager.
Examples	NESI ***Examples*** illustrate key aspects of Perspectives, Guidance, or Best Practices.
Glossary	NESI ***Glossary*** entries provide terms, acronyms, and definitions used in The context of NESI Perspectives, Guidance and Best Practices.
References	NESI ***References*** identify directives, instructions, books, Web sites, and other sources of information useful for planning or execution.

Releasability Statement

This document has been cleared for public release by competent authority in accordance with DoD Directive 5230.9 and is granted Distribution Statement A: Approved for public release; distribution is unlimited. Obtain electronic copies of this document at http://nesipublic.spawar.navy.mil.

Vendor Neutrality

The NESI documentation sometimes refers to specific vendors and their products in the context of examples and lists. However, NESI is vendor-neutral. Mentioning a vendor or product is not intended as an endorsement, nor is a lack of mention intended as a lack of endorsement. Code examples typically use open-source products since NESI is built on the open-source philosophy. NESI accepts inputs from multiple sources so the examples tend to reflect whatever tools the contributor was using or knew best. However, the products described are not necessarily the best choice for every circumstance. Users are encouraged to analyze specific project requirements and choose tools accordingly. There is no need to obtain, or ask contractors to obtain, the open-source tools that appear as examples in this guide. Similarly, any lists of products or vendors are intended only as references or starting points, and not as a list of recommended or mandated options.

Disclaimer

Every effort has been made to make NESI documentation as complete and accurate as possible. Even with frequent updates, this documentation may not always immediately reflect the latest technology or guidance. Also, references and links to external material are as accurate as possible; however, they are subject to change or may have additional access requirements such as Public Key Infrastructure (PKI) certificates, Common Access Card (CAC) for user identification, and user account registration.

Contributions and Comments

NESI is an open-source project that involves the entire development community. Anyone is welcome to contribute comments, corrections, or relevant knowledge to the guides via the Change Request tab on the NESI Public site, http://nesipublic.spawar.navy.mil, or via the following email address: nesi@spawar.navy.mil.

Collaboration Site

The Navy has established a collaboration site to support NESI community interaction. It is located at https://nesi.spawar.navy.mil (user registration required). Use this site for collaborative software development across distributed teams.

References

- R1164: DoD Directive 5000.1, *The Defense Acquisition System*, 12 May 2003 (certified current as of 24 November 2003); http://www.dtic.mil/whs/directives/corres/pdf/500001p.pdf.
- R1165: DoD Instruction 5000.2, *Operation of the Defense Acquisition System*, 12 May 2003; http://www.dtic.mil/whs/directives/corres/pdf/500002p.pdf.
- R1166: DoD Directive 8100.1, *Global Information Grid (GIG) Overarching Policy*, 19 September 2002 (certified current as of 21 November 2003); http://www.dtic.mil/whs/directives/corres/pdf/810001p.pdf.
- R1167: DoD Directive 4630.05, *Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 05 May 2004 (certified current as of 23 April 2007); http://www.dtic.mil/whs/directives/corres/pdf/463005p.pdf.
- R1168: DoD Instruction 4630.8, *Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 30 June 2004; http://www.dtic.mil/whs/directives/corres/pdf/463008p.pdf.
- R1170: *DoD Global Information Grid (GIG) Architecture*, Version 2.0, August 2003.
- R1171: *DoD Architecture Framework (DoDAF)*, Version 1.5, 23 April 2007; https://dars1.army.mil/IER/index.jsp
- R1172: *DoD Net-Centric Data Strategy*, DoD Chief Information Officer, 9 May 2003, http://www.defenselink.mil/cio-nii/docs/Net-Centric-Data-Strategy-2003-05-092.pdf.
- R1173: CJCSI 3170.01F, *Joint Capabilities Integration and Development System*, 01 May 2007; http://www.dtic.mil/cjcs_directives/cdata/unlimit/3170_01new.pdf.
- R1174: CJCSM 3170.01C, *Operation of the Joint Capabilities Integration and Development System*, 01 May 2007; http://www.dtic.mil/cjcs_directives/cdata/unlimit/m317001.pdf.
- R1175: CJCSI 6212.01D, *Interoperability and Supportability of Information Technology and National Security Systems*, 8 March 2006; http://www.dtic.mil/cjcs_directives/cdata/unlimit/6212_01.pdf.
- R1176: *Net-Centric Operations and Warfare Reference Model (NCOW RM)*, v1.1, 17 November 2005.
- R1177: *Net-Centric Checklist*, V2.1.3, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 12 May 2004; http://www.defenselink.mil/cio-nii/docs/NetCentric_Checklist_v2-1-3_.pdf.
- R1178: *A Modular Open Systems Approach (MOSA) to Acquisition*, Version 2.0, September 2004; http://www.acq.osd.mil/osjtf/mosapart.html.
- R1179: *DoD IT Standards Registry* (*DISR*); http://disronline.disa.mil.
- R1180: *Net-Centric Attributes List*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 2 February 2007; http://www.defenselink.mil/cio-nii/docs/NetCentricAttributesOfficial.pdf .
- R1181: *Global Information Grid (GIG) Key Interface Profiles (KIPs) Framework* (DRAFT), Version 0.95, 7 October 2005.

P1118: NESI Part 5: Developer Guidance

NESI Part 5: Developer Guidance provides chief engineers and software developers with detailed implementation guidance for applications, services, and data. This effort leverages current best practices from the software development community to enable the **Department of Defense** (**DoD**) to create net-centric, extensible, scalable enterprise solutions. The goal is to modernize and improve the development of net-centric applications and services as critical warfighter capabilities.
Software developers can choose to use published applications via interfaces and services or build applications and services that interface with the infrastructure. Any application that must interoperate in the DoD Net-Centric Enterprise should be built and maintained in accordance with the standards, policies, and processes within this guide.

NESI Part 5 provides developers with detailed software development guidance, best coding practices, lessons learned, and code samples. It serves as a reference, not a document to be read cover to cover. The guidance in NESI Part 5 is designed to do the following:

- Permit independent paces of development and change on each side of the enterprise, reducing risk and impacts of changes to application developers
- Implement connection strategies that extend the life and reach of legacy applications while legacy application developers restructure their systems

Program managers and chief engineers will find the overview and guidance sections helpful while doing the following:

- Directing their programs and activities to build systems (use this information in combination with ***NESI Part 2: ASD(NII) Checklist*** Guidance and ***NESI Part 4: Node Guidance***)
- Reviewing Statements of Work (Developers may also use the information for this purpose)
- Reviewing deliverables for compliance
- Migrating legacy systems to the net-centric environment(use this information in combination with ***NESI Part 3: Migration Guidance***)

P1072: Technical Guidance and Tactics

This Complex Perspective contains guidance in the following areas.

High-Level guidance for developing Net-Centric software:

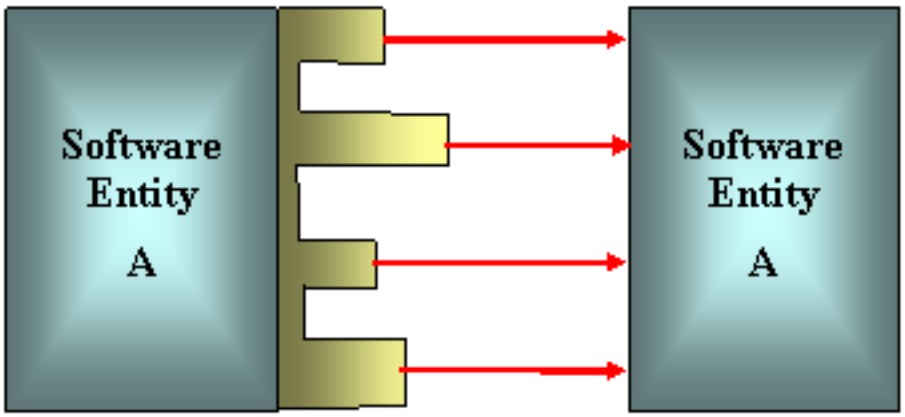
- Publish and Insulate Public Interfaces
- Implement a Component-Based Architecture
- Automate the Software Build Process

Interface Design:

- Public Interface Design
- Standard Interface Documentation

P1060: Public Interface Design

A public interface is the logical point at which independent software entities interact. The entities may interact with each other within a single computer, across a network, or across a variety of other topologies. It is important that public **interfaces** be stable and designed to support future changes, enhancements, and **deprecation** in order for the interaction to continue.



Guidance

- G1213: Provide an architecture design document.
- G1215: Provide a coding standards document.
- G1216: Provide a software release plan document.
- G1214: Provide a document with a plan for **deprecating** obsolete **interfaces**.
- G1021: Create fully insulated classes.
- G1022: Insulate public **interfaces** from compile-time dependencies.
- G1208: Add new functionality rather than redefining existing interfaces in a manner that brings incompatibility.
- G1004: Make public **interfaces** backward-compatible within the constraints of a published **deprecation** policy.
- G1018: Assign identifiers to all public interfaces.
- G1019: Deprecate public interfaces in accordance with a published deprecation policy.

Best Practices

- BP1240: Present complete and coherent sets of concepts to the user.
- BP1241: Design statically typed **interfaces**.
- BP1242: Minimize an **interface's** dependencies on other **interfaces**.
- BP1243: Express **interfaces** in terms of application-level types.
- BP1244: Use assertions only to aid development and **integration**.

Examples

Java Interface

Interface Classes

Create interface classes as shown in the following sample:

```
public interface weather {
   public String getLocation();
   public String getWind();
   public String getVisibility();
   public String getTemperature();
   public String getPressure();
} // End weather interface
```

Implementation of the interface

There are different ways to implement the interface. This approach uses a plug-in strategy.

Interface Implementation

```
public class airPortWeather implements weather {
   airPortWeather() { }

   public String getLocation() {
     // business logic goes here . . .
     return strLocation;
   } // End getLocation
   public String getWind() {
     // business logic goes here . . .
     return strWind;
   } // End getWind
   public String getVisibility() {
     // business logic goes here . . .
     return strVisibility;
   } // End getVisibility
   public String getTemperature() {
     // business logic goes here . . .
     return strTemperature;
   } // End getTemperature
   public String getPressure() {
     // business logic goes here . . .
     return strPressure;
   } // End getPressure
} // End airPortWeather
```

Interface implementation plug-in

```
public class weatherReport {
   private weather myWx = null;
   weatherReport() {
   } // End constructor
   public void addWeatherProvider(weather lclWxProvider) {
     this.myWx = lclWxProvider;
   } // End addWeatherProvider
   public String getLocation() {
     return (this.myWx.getLocation());
   } // End getLocation
   public String getWind() {
     return (this.myWx.getWind());
   } // End getWind
   public String getVisibility() {
     return (this.myWx.getVisibility());
   } // End getVisibility
```

```
   public String getTemperature() {
     return (this.myWx.getTemperature());
   } // End getTemperature
   public String getPressure() {
     return (this.myWx.getPressure());
   } // End getPressure
} // End weatherReport class
```

These examples use protocol classes/interface classes and an implementation class through composition to decouple the interface implementation. There are other ways to implement the **interfaces** to get effective insulation. The specifics are application-dependent and are up to the individual application developers.

C++ Interface

Protocol classes

Use protocol classes to define public **interfaces**.

The characteristics of a protocol class follow:

- It neither contains nor inherits from classes that contain member data, non-virtual functions, or private (or protected) members of any kind.
- It has a non-inline virtual destructor defined with an empty implementation.
- All member functions other than the destructor, including inherited functions, are declared pure virtual and left undefined.

Example

```
// Abstract base class or protocol class specifies an interface
// for derived classes
// no data members
// no constructors
// a virtual destructor
// set of pure virtual functions
#ifndef _weather_h_
   #define _weather_h_class
   weather {
     public: weather() { };
     virtual ~weather() { };
     virtual const char* getLocation() const = 0;
     virtual const char* getWind() const = 0;
     virtual const char* getVisibility() const = 0;
     virtual const char* getTemperature() const = 0;
     virtual const char* getPressure() const = 0;
   }; // End weather
#endif
```

Implementation of the interface

Interface implementation

There are different ways to implement the interface.

airPortWeather.h

```
#ifndef _airPortWeather_h_
   #define _airPortWeather_h_class
   airPortWeather : public weather {
     public: airPortWeather () { } ;
     ~airPortWeather() { } ;
```

```
      const char* getLocation() const ;
      const char* getWind() const ;
      const char* getVisibility() const ;
      const char* getTemperature() const ;
      const char* getPressure() const ;
   };//end airPortWeather
#endif
```

airPortWeather.cpp

```
#include "stdafx.h"
#include
#include
#ifndef _weather_h_
   #include "weather.h"
#endif
#ifndef _airPortWeather_h_
   #include "airPortWeather.h"
#endif
const char* airPortWeather::getLocation() const {
   // business logic goes here . . .
   return strLocation;
} // End getLocation
const char* airPortWeather::getWind() const {
   //business logic goes here . . .
   return strWind;
} // End getWind
const char* airPortWeather::getVisibility() const {
   // business logic goes here . . .
   return strVisibility;
} // End getVisibility
const char* airPortWeather::getTemperature() const {
   // business logic goes here . . .
   return strTemperature;
} // End getTemperature
const char* airPortWeather::getPressure() const {
   // business logic goes here . . .
   return strPressure;
} // End getPressure
```

Plug-in

weatherReport.h

```
#ifndef _weatherReport_h_
   #define _weatherReport_h_class weather;
   class weatherReport{
     private: weather *myWx_;public: weatherReport ( ) { } ;
     virtual ~weatherReport();
     void addWeatherProvider(weather *lclWxProvider) ;
     const char* getLocation() const ;
     const char* getWind() const ;
     const char* getVisibility() const ;
     const char* getTemperature() const ;
     const char* getPressure() const ;
   }; //end weatherReport
#endifweatherReport.cpp
#ifndef _weather_h_
   #include "weather.h"
#endif
#ifndef _airPortWeather_h_
   #include "airPortWeather.h"
#endif
#ifndef _weatherReport_h_
   #include "weatherReport.h"
#endif
weatherReport::~weatherReport( ) { } ; // End destructor
void weatherReport::addWeatherProvider ( weather *lclWxProvider ) {
   myWx_ = lclWxProvider;
}; // End addWeatherProvider
```

```
const char* weatherReport::getLocation() const {
   return (myWx_->getLocation());
}; // End getLocation
const char* weatherReport::getWind() const {
   return (myWx_->getWind());
}; // End getWind
const char* weatherReport::getVisibility() const {
   return (myWx_->getVisibility());
}; // End getVisibility
const char* weatherReport::getTemperature() const {
   return (myWx_->getTemperature());
}; // End getTemperature
const char* weatherReport::getPressure() const {
   return (myWx_->getPressure());
}; // End getPressure
```

Costs and Benefits

The benefits of using protocol classes include the following:

- Insulating applications from the external **client**
- Insulating changes that are internal to the **interface**
- Insulating changes to the public **interface** from changes to the implementation of the **interface**

Insulation has costs, but these tend to be outweighed by the gains in **interoperability** and reusability. Some of the costs include the following:

- Going through the implementation pointer
- Addition of one level of indirection per access
- Addition of the size of the implementation pointer per object to memory requirements

References

- R1071: Various methodologies, such as **refactoring**, support the determination of **interfaces**. **Refactoring** generally applies to the entire software implementation, but is especially helpful in properly flushing out **interfaces**.
- R1072: There are other approaches defined through the pattern community: http://hillside.net/patterns/.
- R1073: See chapter 3 of Java Design: Building Better Apps &

P1062: Publish and Insulate Public Interfaces

This Perspective lists high-level guidance for implementing public interfaces.

Guidance

- G1001: Define public **interfaces** in a formal standard.
- G1002: Separate public **interfaces** from implementation.
- G1003: Separate the contents of application libraries that are to be shared from libraries that are to be used internally.
- G1004: Make public **interfaces** backward-compatible within the constraints of a published **deprecation** policy.
- G1005: Separate infrastructure capabilities from **mission** functions.
- G1007: Ensure that applications use open, standardized, **vendor**-neutral **API**(s).
- G1008: Isolate platform-specific **interfaces** and **vendor** dependencies.
- G1010: Use **open-standard** logging frameworks.
- G1022: Insulate public **interfaces** from compile-time dependencies.
- G1073: Isolate vendor extensions to enterprise-services standard interfaces.
- G1018: Assign identifiers to all public interfaces.
- G1019: Deprecate public interfaces in accordance with a published deprecation policy.

P1007: Automate the Software Build Process

A software build process interfaces with source control, compiles code, creates executables, runs unit tests, packages and deploys, and generates documentation. An automated software build process is a necessary part of every software development project and ensures the software will be built in the same manner each time.

Guidance

- G1190: Use a build tool.
- G1218: Use a build tool that supports operation in an automated mode.
- G1219: Use a build tool that checks out files from configuration control.
- G1220: Use a build tool that **compiles** source code and dependencies that have been modified.
- G1221: Use a build tool that creates libraries or archives after all required compilations are completed.
- G1222: Use a build tool that creates executables.
- G1223: Use a build tool that is capable of running unit tests.
- G1224: Use a build tool that cleans out intermediate files that can be regenerated.
- G1225: Use a build tool that is independent of the **Integrated Development Environment**.

Best Practices

- BP1075: All application developers should use the Apache **Ant** build tool to build, package, and deploy **Java EE** applications.

References

- R1021: ANT - http://ant.apache.org/
- R1022: Java EE - http://java.sun.com/javaee

P1069: Standard Interface Documentation

This section provides guidance for documenting source code. The references provide links on documenting code for the Java and the Microsoft .NET environments. For all other languages, configuration files, and XML files, please follow the associated language-specified format for documentation.

Guidance

- G1027: Internally document all source code developed with DoD funding.

Examples

Java

Javadoc commands

The Javadoc tool parses special tags when they are embedded within a Javadoc comment. These doc tags enable a programmer to autogenerate a complete, well-formatted API from the source code. The tags start with an ampersand (@) and are case-sensitive; an "a" is different from an "A."

A tag must start at the beginning of a line, after any leading spaces and an optional asterisk, or it will be treated as normal text. By convention, group tags with the same name together. For example, put all **@see** tags together.

Sample Java code with Javadoc

This is a sample Enterprise Java Bean with Javadoc tags for the API that implements a method to set a string to "Hello." Use this example to generate documents from the command line and from Ant.

```
package com.testejb;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
/**
 * This session bean demonstrates a simple session bean
 */
public class TestSessionBean implements SessionBean {
   private String test = "hello from the test ejb";
   public TestSessionBean( ){ }
   public void setSessionContext(SessionContext sc){ }
   public void ejbActivate( ){ }
   public void ejbPassivate( ){ }
   public void ejbRemove( ){ }
   public void ejbCreate( ){ }
   /**
    * This method returns the test string
    * @return the value of test
    */
   public String getTest( ) {
      return test;
   } // End getTest
   /**
    * This method sets the test string
    * @param String t
    */
   public void setTest(String t) {
      test = t;
   } // End setTest
} // End TestSessionBean
package com.testejb;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
```

```
/**
 * This session bean demonstrates a simple session bean
 */
public class TestSessionBean implements SessionBean {
   private String test = "hello from the test ejb";
   public TestSessionBean( ){ }
   public void setSessionContext(SessionContext sc){ }
   public void ejbActivate( ){ }
   public void ejbPassivate( ){ }
   public void ejbRemove( ){ }
   public void ejbCreate( ){ }
   /**
    * This method returns the test string
    * @return the value of test
    */
   public String getTest( ) {
     return test;
   } // end getTest
   /**
    * This method sets the test string
    * @param String t
    */
   public void setTest(String t) {
     test = t;
   } // End setTest
} // End TestSessionBean
```

Microsoft

Sample .NET C# with documentation tags

This sample .NET application shows the necessary comment structure to generate the interface documentation.

```
using System;
namespace HelloWorldNamespace {
   ///
   /// Hello World Example C# application
   ///
   class HelloWorldClass {
     ///
     /// The main entry point for the application.
     ///
     [STAThread]
     static void Main(string[] args) {
       // Loop through some indices and display the value
       // from GetHelloText(...)
       for ( int expressionCounter = -1; expressionCounter < 4; expressionCounter ++ ) {
         Console.Out.WriteLine (expressionCounter.ToString("#0") + ": " +
GetHelloText(expressionCounter) );
       } // End for
       Console.In.Read(); // Pause the console
     } // End main
     ///
     /// Gets a "hello" string given an index
     ///
     ///
     /// Index of the "hello" string to retrieve
     ///
     ///
     /// A "hello"string if the index is valid, otherwise
     /// an error
     ///
     static stringGetHelloText(int index) {
       string[] helloExpressions = new string[] {
         "Hello World", "Hello All", "Howdy"
       };
       if (index < 0 || index >=helloExpressions.Length) {
         return "Error";
       } // End if
```

```
        else {
          returnhelloExpressions [index];
        } // End else
      } // End get Hello
    } // EndHelloWorldClass
} // End HelloWorldNamespace
```

References

- R1074: For information about tags that might appear in future releases of Java, see http://java.sun.com/products/jdk/javadoc/proposed-tags.html.
- R1075: For Microsoft Visual Studio 6 and earlier, use the open-source tools described in C/C++ Javadoc guidelines.
- R1076: For Microsoft Visual Studio .NET 2002 and above, use the XML documentation tools that come with the .NET IDE Integrated Development Environment.

P1034: Implement a Component-Based Architecture

The Federation of Government Information Processing Councils/Industry Advisory Council (FGIPC/IAC) defined **component-based architecture** (**CBA**) as follows in a March 2003 paper titled Succeeding with "Component-Based Architecture in e-Government":

"An architecture process that enables the design of enterprise solutions using pre-manufactured components. The focus of the architecture may be a specific project or the entire enterprise. This architecture provides a plan of what needs to be built and an overview of what has been built already." [Succeeding with Component-Based Architecture]

CBA represents a shift from the traditional, custom-development-oriented,"design, code, and test" approach that has been used throughout the DoD in the past to a more business-oriented "architect, acquire, and assemble" approach.

The custom-development approach has been successful in building many systems. However, the integration, evolution, reuse and cost of these systems have presented a problem. Consequently, these custom-developed systems have been labeled as archaic **stovepipes** that can not plug-and-play with other systems.

CBA promises benefits such as shorter time to market, lower risk, and modular and adaptive systems.

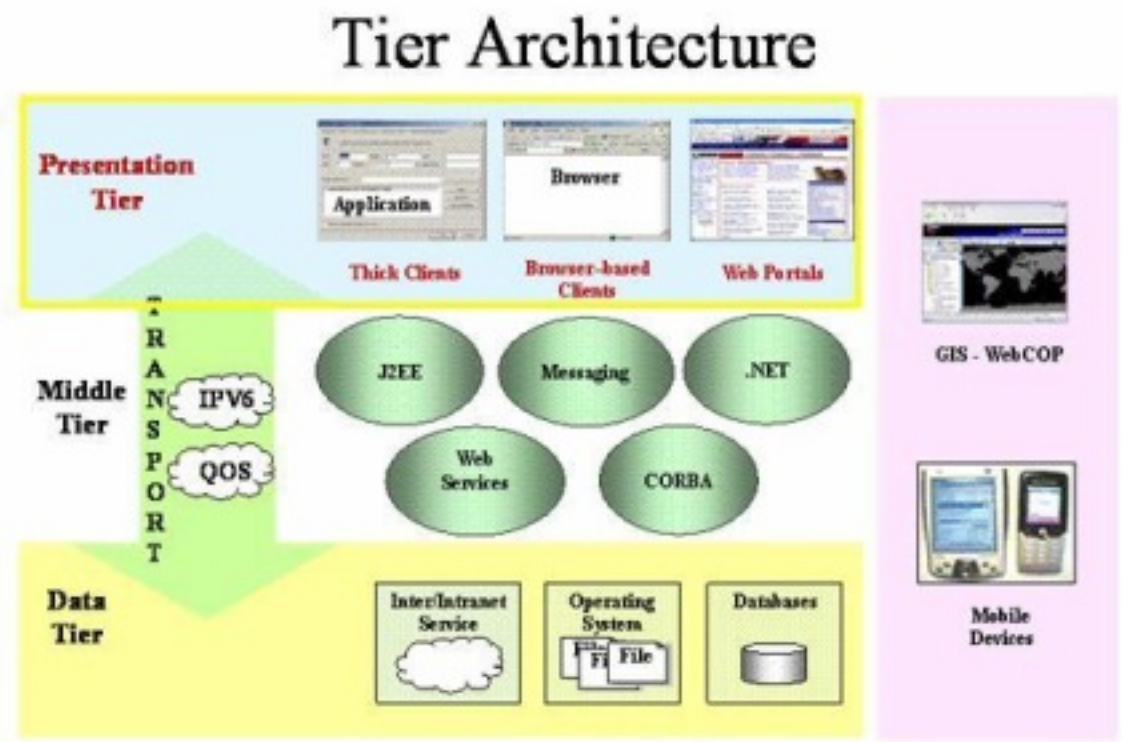
The core of CBA is components. The NESI definition of the term **component** is that it is one of the parts that make up a system; a component may be hardware or software and may be subdivided into other components. The following guidance statements capture the essence of components.

Guidance

- G1011: Make components independently deployable.
- G1012: Use a set of services to expose Component functionality.
- G1217: Develop and use externally configurable components.

P1058: Presentation Tier

The presentation tier represents all the components used to generate an interactive display that enables users to communicate with applications. The components of a presentation tier are not necessarily in the same physical location. The presentation tier communicates with the middle tier to make requests and retrieve data from the data tier. The presentation tier then shows the **end user** the data retrieved from the middle tier. Components located in the middle tier that build **Web pages** for display are considered part of the presentation tier.



Detailed Perspectives

Human-Computer Interaction

- Human Factor Considerations for Web-Based User Interfaces
- Designing User Interfaces for Accessibility
- Desinging User Interfaces for Internationalization

Browser-Based Clients

- XML Rendering
- Active Server Pages (ASP)
- Active Server Pages for .NET (ASP.NET)
- Java Server Pages (JSP)

- Style Sheets
- Web Portals

Thick Clients

Guidance

- G1032: Validate all input fields.

P1032: Human-Computer Interaction

Human-Computer Interaction (HCI) is the study, planning, and design of the interaction between humans and computers. HCI is a subset of Human Systems Integration (HSI). Human Systems Integration is a requirement for **Department of Defense** (**DoD**) acquisition as spelled out on Section 3.7 and Enclosure 7 of DoD Instruction 5000.2. In particular, this instruction requires that Program Managers shall take steps to include human factors engineering during system engineering over the lifecycle of the program to provide effective human-machine interfaces, "Where practical and cost effective, system designs shall minimize or eliminate system characteristics that require excessive cognitive, physical or sensory shills; entail extensive training or workload-intensive tasks; result in mission-critical errors; or produce safety or health hazards."

Interoperability includes both the technical exchange of information and the end-to-end operational effectiveness of that exchanged information as required for mission accomplishment. Whenever a user is required to interact with a computer user interface to accomplish a mission, and that interaction fails due to poor design (i.e., information is misunderstood or interaction results in a high cognitive load) then the risk of not accomplishing the mission is increased.

This perspective provides guidance and best practices that benefit human computer interaction to increase total system performance, reduce maintenance costs through better design, and accommodate the cognitive characteristics of the user. This perspective provides guidance for human factors common to all applications including data entry, data display, and user control appearance and behavior. The following detailed perspectives provide additional human factor guidance on more specific topics.

Detailed Perspectives

- Human Factor Considerations for Web-Based User Interfaces
- Designing User Interfaces for Accessibility
- Designing User Interfaces For Internationalization

Guidance

- G1760: Solicit feedback from users on user interface usability problems.
- G1761: Provide units of measurements when displaying data.
- G1762: Indicate all simulated data as simulated.
- G1763: Indicate the security classification for all classified data.
- G1032: Validate all input fields.
- G1268: Label all data entry fields.
- G1270: Include scroll bars for text entry areas if the data buffer is greater than the viewable area.
- G1285: Use **relative font sizes**.
- G1286: Provide text labels for all buttons.
- G1287: Provide feedback when a transaction will require the user to wait.

Best Practices

- BP1280: In tabular data displays, right justify integer data.

- BP1281: In tabular data displays, justify numeric data with decimals by using the decimal point.
- BP1767: Follow a standard process for human systems integration engineering such as the one defined by the International Organization for Standardization in ISO 13407:1999 on human-centered design processes for interactive systems.
- BP1272: Disable dependent child controls when the parentcontrol is inactive.
- BP1273: Gray out the push button label if a button is unavailable.
- BP1290: Use a tool tip to display help information about a control when the purpose of the control is not self-evident.
- BP1291: Use obvious navigation controls for moving between pages in search results that span multiple pages.
- BP1298: Provide basic search functionality as the default with a link or button that provides more advanced search features.
- BP1054: Use standard controls that provide input choices for the user. These controls might include radio buttons, check boxes, list boxes, and drop-downs.

References

- R1139: Department of Defense Instruction 5000.2, "Operation of the Defense Acquisition System," Section 3.7 and Enclosure 7
- R1140: Human Engineering MIL-STD 1472F, section 5.14 "User Computer Interface"
- R1141: "Guide for developing usable and useful web sites" [http://usability.gov]
- R1142: "Microsoft Windows User Experience: Official Guidelines for User Interface Developers and Designers," Redmond, WA: Microsoft Press, 1999
- R1143: "Apple Human Interface Guidelines," Apple Computer, Inc., 2004 [http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html]
- R1145: "The GNOME Usability Project. GNOME Human Interface Guidelines (1.0)," [http://developer.gnome.org/projects/gup/hig/1.0/]
- R1146: "Java Look and Feel Design Guidelines: Advanced Topics," Sun Microsystems, Inc., 2001
- R1147: "Java Look and Feel Design Guidelines. Second Edition," Sun Microsystems, Inc., 2001
- R1148: "Designing Interfaces: Patterns for Effective Interaction Design," Tidwell, J., O'Reilly Media, Inc., 2006

P1112: Designing User Interfaces for Internationalization

Internationalization is the process of generalizing software so that it is interoperable with multiple languages (i.e., locales) and cultural conventions without the need for re-design or re-compilation. If an application designed for a U.S. audience will be used in combined or coalition warfare operations, it needs to provide a user interface that matches users' expectations, interacts with users in their native language, and displays data in a manner that is consistent with users' cultural conventions. The purpose of this perspective is to provide a starting reference for developers needing to support internationalization and provides best practices and resources.

Best Practices

- BP1764: Make all localizable user interface elements such as text and graphics externally configurable.
- BP1765: Declare the encoding type for all user interface content.
- BP1766: Develop user interfaces to accommodate variable syntactic structure for messages.

References

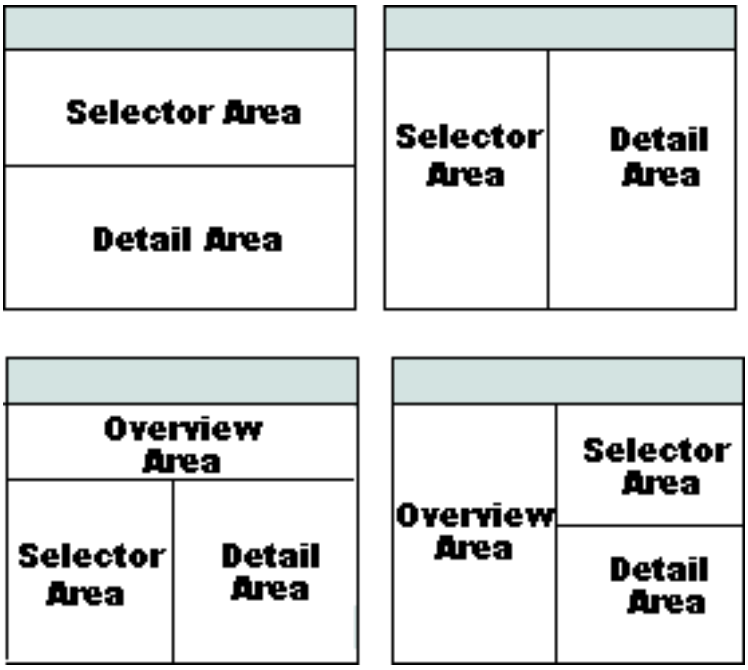
- R1159: "Internationalization Best Practices: Specifying Language in XHTML & HTML Content," W3C, [http://www.w3.org/TR/i18n-html-tech-lang/]
- R1160: "Internationalization Quick Tips for the Web," W3C [http://www.w3.org/International/quicktips/]
- R1161: "Developing and Localizing International Software," Madell, T., Parsons, C. & Abegg, J., Englewood Cliffs, NJ: Prentice Hall, 1994
- R1162: "Programming for the World: A Guide to Internationalization," O'Donnell, S.M., Englewood Cliffs, NJ: Prentice Hall, 1994
- R1163: "Software Internationalization and Localization: An Introduction," Uren, E., Howard, R. & Perinotti,T., New York: Van Nostrand Reinhold, 1993

P1108: Human Factor Considerations for Web-Based User Interfaces

Web based user interfaces include **Web sites**, **Web applications**, and **Web portals**. This perspective provides guidance and best practices relating to human factors consideration that are specific to Web-based user interfaces. Additional information concerning general user interface guidance is available in the Human Computer Interaction perspective.

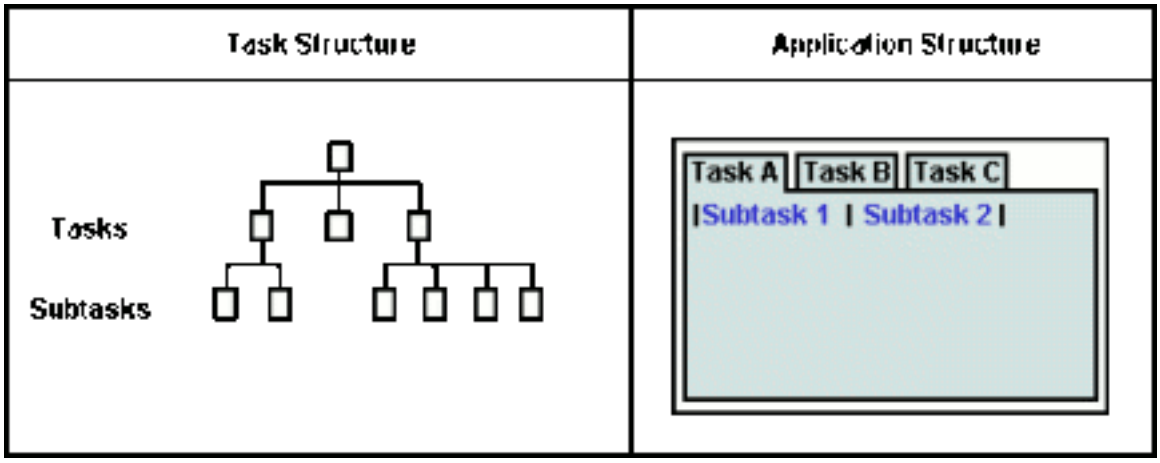
Web sites tend to be content-centric and are generally developed using **HTML** for marking up content for Web pages. Sometimes other technologies such as **JavaScript** are used to add interactivity to Web pages. If developers choose to use a mix of HTML and other technologies to deliver Web content, it is important that they design their Web pages so the pages work correctly when viewed with browsers that support these technologies as well as with browsers that do not. In this way, all users will have an acceptable experience using the Web site.

Web sites vary in their layout, but there are common themes for layouts that are widely used and understood users. Some example Web site layouts are shown in this figure:



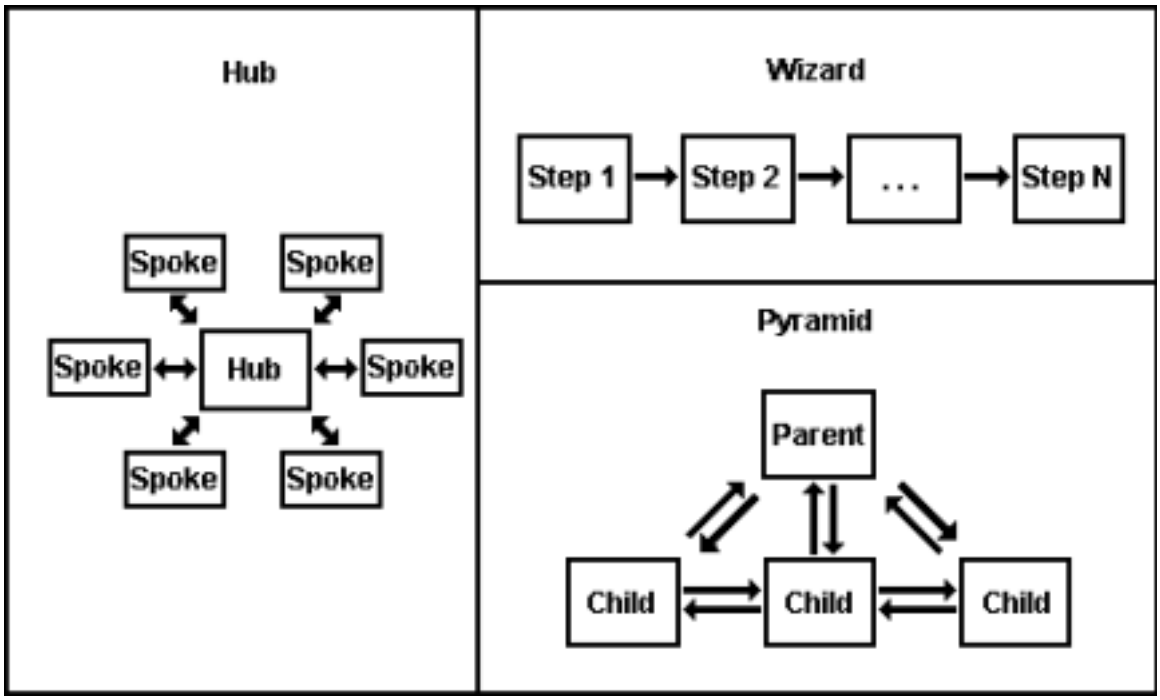
Web Applications

A Web site tends to be content-centric, but a Web application tends to be task-centric and organizes content around a hierarchy of tasks. An example user interface for a given task structure is shown in this figure:



A Web application often supports interactivity similar to that available in a desktop application but delivered to users within the framework of a browser. Because a Web application allows users to create, save, and delete data, it supports greater complexity in design and interactivity compared to a content-oriented site.

In addition to application structure, there are common navigation models that are well understood by users for Web application workflow. Some common examples are in this figure:



The "hub navigation metaphor" is often used for applications where a task consists of multiple independent steps that are performed in any order. The hub page present users with a collection of "spoke" pages that they access from a single page; when users submit their input, they are returned to the hub page.

The "wizard navigation" metaphor is often used when a task consists of multiple interdependent steps that are performed in a predefined order. In this metaphor, a wizard presents users with a collection of pages that they interact with sequentially; when the user submits their input, the user is presented with the next page

The "pyramid navigation" metaphor is often used when it is important to navigate to sibling, child, or parent pages while completing tasks; when the user submit their input, they are returned to the same page where they follow links to another adjacent page in the pyramid.

Web Portals

A portal is a type of Web application that provides a gateway from which users can access the information, resources, and services they need. A portal aggregates and organizes content from different sources within a Web page related to specific mission or business task. Sometimes a portal allows users to personalize what and how information is presented to them such as selecting and arranging the content presented on the portal page and to choosing the "look and feel" of the display.

The pages in a portal contain portlets that enable users to view and/or interact with Web-based information related to a specific function. A portlet provides more than a view of existing Web content, functioning instead as a complete application with multiple states and view modes.

Since portals are designed to contain portlets from various sources, it is important for portlet developers to develop portlets carefully to allow for a standard presentation and behavior when the portlet is deployed within the portal. Allowing for configuration for presentation such as fonts and colors allows for a common look and feel across all portlets within a portal. Developing portlets according to standards for user controls enables a better experience for the end user with respect to common portlet control behavior.

Guidance

- G1267: Use industry standard HTML data entry fields on Web pages.
- G1276: Do not modify the contents of the Web browser's status bar.
- G1277: Do not use tickers on a Web site.
- G1278: Use the browser default setting for links.
- G1284: Use only one font for body text.
- G1292: Use text-based Web site navigation.
- G1293: Use descriptive labels for all clickable graphics.
- G1294: Provide a site map on all Web sites.
- G1295: Provide redundant text links for linked images and each active region of an image map.
- G1566: Use **`alt`** attributes to provide alternate text for non-text items such as images.
- G1759: Use a style guide when developing Web portlets.

Best Practices

- BP1297: Structure a Web site hierarchy so users can reach important information and/or frequently accessed functions in a maximum of three jumps.
- BP1299: Include a link back to the home page on all Web pages.
- BP1042: Do not build a **Web page** where the horizontal width is greater than the screen (vertical scrolling is fine), planning for the lowest common denominator to be super-VGA resolution (800 x 600).
- BP1041: Do not change the default colors of the links.
- BP1038: Use a **sans serif** font (e.g., Arial, Verdana) in Web pages rather than a serif font (e.g., Times New Roman).
- BP1039: Do not underline any text unless it is a link.
- BP1768: Use design patterns for application navigation.

References

- R1091: "Web Portal Design Guide 1.1 July 2006," Space and Naval Warfare Systems Center San Diego
- R1144: "Seven Common Mistakes in Designing a Usable Portal," Iozzo, N., 2002 [http://www.tandemseven.com/pdf/T7_Seven_Common_Mistakes.pdf]
- R1149: "Common Presentation Layer Guide Standard 03-01," NAVSEA, September 2006
- R1151: "Web-Based Portal Computer-Human Interface Guidelines," Ahlstrom, V. & Allendoerfer, K., 2004 [http://hf.tc.faa.gov/products/bibliographic/tn0423.htm]
- R1152: "Web Application Design Handbook: Best Practices for Web-Based Software," Fowler, S. & Stanwick, V., San Francisco: Morgan Kaufmann Publishers, 2004.

P1111: Designing User Interfaces for Accessibility

Section 508 of the Rehabilitation Act of 1973, as amended, requires that individuals with disabilities have access to and use of information that is comparable to that provided to federal employees and members of the public who are not disabled. The standards created under Section 508 define technology accessibility requirements for all types of information technology in the federal sector, including Web-based intranet and Internet information and applications.

Federal accessibility standards focus on providing redundancy in information presentation and interaction so individuals with disabilities can use different modalities to access information. The scope of Section 508 is confined to the federal sector, with a limited exemption for systems used for military command, weaponry, intelligence, and cryptologic activities. The exemption does not apply to routine business and administrative systems used for other defense-related purposes or by defense agencies or personnel. A Web application or portal that will be used in these systems is required to comply with Section 508 standards.

Guidance

- G1044: Comply with Federal accessibility standards contained in Section 508 of the Rehabilitation Act of 1973 (as amended) when developing software user interfaces.

References

- R1154: "Federal IT Accessibility Initiative," [http://www.section508.gov/]
- R1155: "Electronic and Information Technology Accessibility Standards," Federal Register, [http://www.access-board.gov/sec508/508standards.pdf]
- R1156: "Web Content Accessibility Guidelines 1.0," W3C, [http://www.w3.org/TR/WAI-WEBCONTENT/]
- R1157: "Guidelines for Keyboard User Interface Design," Microsoft Corporation [http://msdn.microsoft.com/library/?url=/library/en-us/dnacc/html/ATG_KeyboardShortcuts.asp]

P1008: Browser-Based Clients

This complex perspective provides guidance for creating and interfacing to thin clients. It includes the following topics:

- XML Rendering
- Active Server Pages (ASP)
- Active Server Pages for .NET (ASP.NET)
- Java Server Pages (JSP)
- Style Sheets

Guidance

- G1035: Follow W3C standards for code which will generate a Web page display.
- G1043: Separate formatting from data through the use of **style sheets** instead of hard coded **HTML** attributes.
- G1271: Provide instructions and **HTML** examples for all style sheets.
- G1283: Use linked style sheets rather than embedded styles.

Best Practices

- BP1040: Use hex codes for all colors (e.g., #FFFF33), never the color name (e.g., yellow).
- BP1291: Use obvious navigation controls for moving between pages in search results that span multiple pages.
- BP1567: Use the <**`abbr`**> and <**`acronym`**> tags to specify the expansion of acronyms and abbreviations.
- BP1568: Use markup language (if available) and styles to represent mathematical equations.

References

- R1023: For answers to frequently asked questions about **cascading style sheets**, see http://www.blooberry.com/indexdot/css/topics/stylefaq.htm
- R1024: Don't Make Me Think by Steve Krug (ISBN 0-7897-2310-7)
- R1025: Creating Killer Interactive Web Sites by Adjacency (ISBN 1-56830-373-4)
- R1026: Designing Web Usability by Jakob Nielsen (ISBN 1-56205-810-X)

P1084: XML Rendering

XML can render display-device-neutral output to a particular output device given a set of display rules or a **style sheet**. The **XSLT** file is the decoupled output formatter that determines how the output device renders the data.

Guidance

- G1045: Define **XML** format information separately in **XSL**.

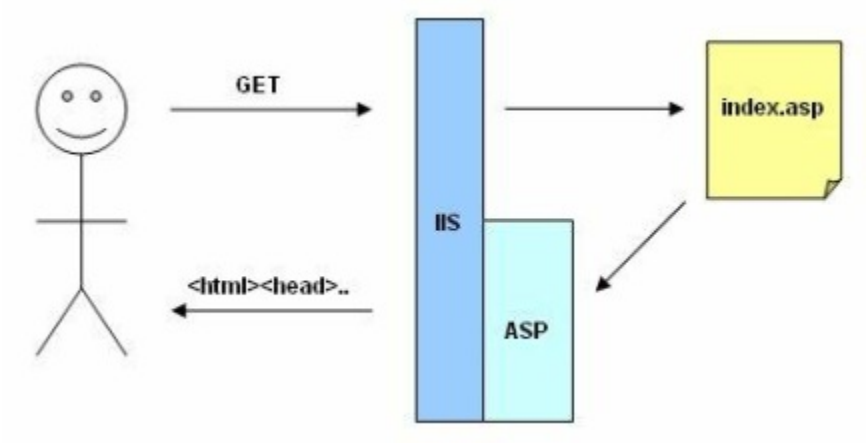
References

- R1007: XSL - definition of XSL in Encyclopedia - http://encyclopedia.laborlawtalk.com/XSL

P1001: Active Server Pages (ASP)

Active Server Pages (**ASP**) are scripts that are executed by Microsoft **Internet Information Services** (**IIS**). The output is returned to the **end user** as **HTML**. Typically, an ASP script generates a customized **Web page** on the fly before sending it to the end user.

- **Active Server Pages**:
  - Are specific to Microsoft
  - Only run on Internet Information Services (IIS) or Personal Web Server (PWS).
  - Can contain HTML, **Jscript**, and VBScript
  - Can access **Component Object Model** (**COM**) component



Guidance

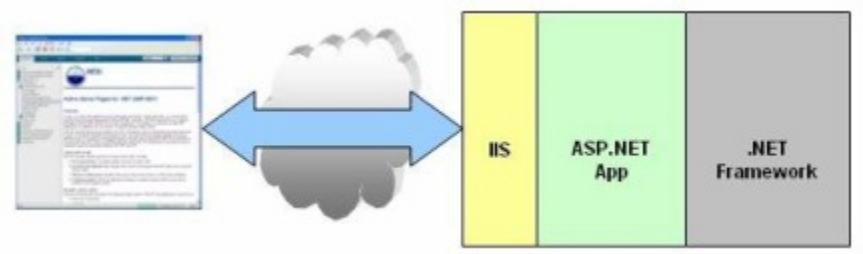
- G1049: Do not use **ActiveX** controls.
- G1050: In **ASP**, isolate the presentation tier from the middle tier using **COM** objects.
- G1058: Use the Model, View, Controller (MVC) pattern to decouple presentation code from other tiers.

References

- R1011: ActiveX Control definition - http://isp.webopedia.com/TERM/A/ActiveX_control.html
- R1012: Component Object Model definition - http://isp.webopedia.com/TERM/C/Component_Object_Model.html
- R1013: JScript definition - http://isp.webopedia.com/TERM/J/JScript.html
- R1014: IIS - definition of IIS in Encyclopedia - http://encyclopedia.laborlawtalk.com/IIS
- R1015: Personal Web Server - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci296469,00.html
- R1016: HTML - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212286,00.html
- R1017: VBScript definition from The Glossary of Internet Terms - http://www.strategicwebventures.com/definitions/Glossary/VBScript/

P1002: Active Server Pages for .NET (ASP.NET)

Microsoft .NET uses ASP.NET for Web applications. ASP.NET requires Microsoft **Internet Information Services** (**IIS**).



ASP.NET improves upon ASP. It has more features than **Java Server Page** (**JSP**), an extensible Web technology that uses static data, **JSP** elements, and server-side Java objects to generate dynamic content for a client. Typically, the static data are **HTML** or XML elements, and in many cases the client is a Web browser. An application responds to events, such as code-behind and event-driven Web controls.

Guidance

- G1052: Use the code-behind feature in ASP.NET to separate presentation code from the business logic.
- G1053: Do not embed HTML code in any code-behind code used by aspx pages.
- G1055: Use a fully qualified, registered **namespace** with identity information for all custom controls.
- G1056: Specify a versioning policy for **.NET** assemblies.
- G1058: Use the Model, View, Controller (MVC) pattern to decouple presentation code from other tiers.

References

- R1018: HTML - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212286,00.html
- R1019: What is JSP? - http://www.webopedia.com/TERM/J/JSP.html

P1040: Java Server Pages (JSP)

Java Server Page (**JSP**) technology enables Web developers and designers to develop and maintain information-rich, **dynamic Web pages** that leverage existing business systems rapidly and easily. As part of the Java technology family, JSP technology enables rapid development of platform-independent, Web-based applications. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

Java Server Pages:

- Are similar to **ASPs**.
- Can contain **HTML**, Java code, and JavaBean components
- Provide a powerful, **dynamic Web page** assembly mechanism
- Are platform-independent
- Are compiled into Servlets at runtime; on most application servers, this occurs only the first time they are invoked

Guidance

- G1060: Encapsulate Java code that is used in **JSP**(s) in tag libraries.
- G1058: Use the Model, View, Controller (MVC) pattern to decouple presentation code from other tiers.

References

- R1040: JavaBeans definition - http://isp.webopedia.com/TERM/J/JavaBeans.html
- R1041: Model-view-controller - a Whatis.com definition - http://whatis.techtarget.com/definition/0,,sid9_gci214607,00.html
- R1042: Java Servlets definition - http://www.fromallangles.com/glossary/web-hosting/terms/java-servlets.htm

P1070: Style Sheets

A **style sheet** is a template used to customize the layout of a **Web site**. Style sheets allow Web sites to present content in a consistent manner. Web designers can create custom tags to override default values:

```
h1,h2,h3 {
   font-family: verdana, arial, 'sans serif';
}
p,table,li {
   font-family: verdana, arial, 'sans serif';
   margin-left: 10pt;
}
```

Guidance

- G1043: Separate formatting from data through the use of **style sheets** instead of hard coded **HTML** attributes.
- G1283: Use linked style sheets rather than embedded styles.
- G1271: Provide instructions and **HTML** examples for all style sheets.

Best Practices

- BP1040: Use hex codes for all colors (e.g., #FFFF33), never the color name (e.g., yellow).
- BP1041: Do not change the default colors of the links.
- BP1038: Use a **sans serif** font (e.g., Arial, Verdana) in Web pages rather than a serif font (e.g., Times New Roman).

References

- R1058: CSS (Cascading Style Sheets) versions 1 (CSS1) and 2 (CSS2)
  [See http://www.w3.org/Style/CSS, http://www.w3.org/TR/REC-CSS1, http://www.w3.org/TR/REC-CSS2]

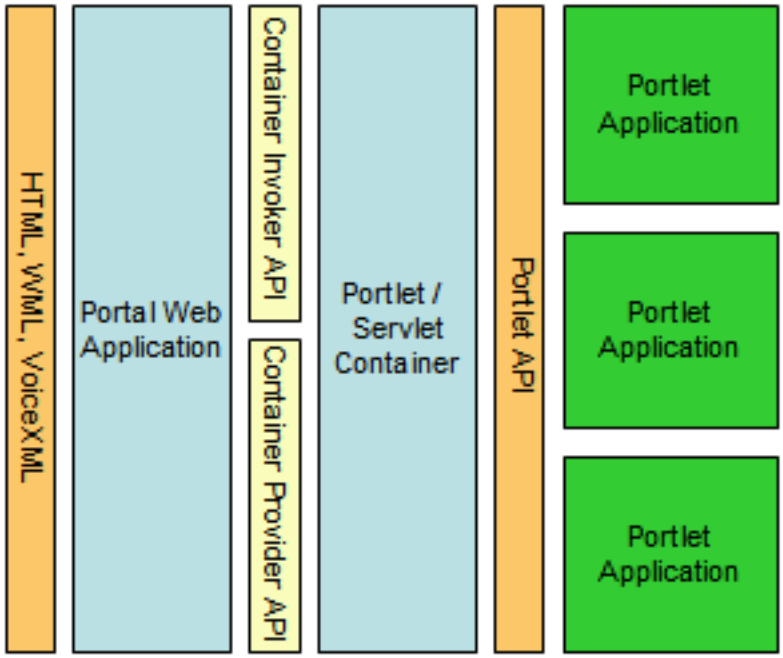
P1077: Web Portals

A **Web portal** is a **Web site** that provides a starting point or gateway to other resources on the Internet or an intranet. Access to a Web portal is typically via **HTTP** and can be in any number of formats including **HTML**, **Wireless Markup Language** (**WML**) or **VoiceXML**. A Web Portal often uses a **Web Application** that provides **single sign-on**, content **integration** and **aggregation** from different sources, **collaboration**, content and document management and personalization of the presentation. It hosts the presentation layer of different backend systems in a **single touch point**.

An attractive feature of a **portal** to an **enterprise** is to aggregate different applications into a single **page** with a common **Look and Feel** that enhances the portal **end user's** experience. A portal may also have sophisticated personalization features, which provide customized content to individual end users or to their roles within the enterprise. **Portal pages** can dynamically coordinate different **portlets** to create specialized content for different portal end users.

IBM's Websphere depicts the basic architecture of portals as a series of layers between the end user's environment such as **browsers**, mobile devices and phones. The portal processes an end user **client** request. A Web Application that interacts with the portlet to request the web page for the current end user is produced. The portal Web Application then uses the **portlet container** for each portlet to retrieve the requested content through the **Web Container Invoker API**. The portlet container calls the portlets through the Portlet API. The Container Provider Service Provider Interface (SPI) enables the Web Application to retrieve information from the portal through its portlet container.

The portlet container invokes the portlets, provides a runtime environment, and manages the lifecycle of the portlet. In addition, it provides persistence for the portlet to store end user information enabling the production of customized Web pages.



Guidance

- G1245: Isolate the Web service portlet from platform dependencies using the Web Services for Remote Portlets (**WSRP**) Specification protocol.

Best Practices

- BP1246: Base Java-based portlets on **JSR 168**.
- BP1247: Encapsulate Java-based **portlets** in a **.war** file.

References

- R1092: For information on WSRP access specifications, see http://www-106.ibm.com/developerworks/webservices/library/ws-wsrp/
- R1093: For information on JSR-168 access specifications, see http://www.jcp.org/aboutJava/communityprocess/final/jsr168/
- R1094: For information on portlets, JSR-168, and WSRP access, see http://www.portletsfactory.com/resources/portlets-jsr168-wsrp-portals-books.html
- R1095: The Navy Enterprise Application Development Guide (NEADG) provides developer's guidance for the Task Force Web (TFW) Navy Enterprise Portal (NEP).
- R1097: The Air Force Portal is built upon the BroadVision portal (http://www.broadvision.com/). For specifics on this portal, refer to http://www.gcss-af.com/.
- R1098: Other Air Force portal initiatives are based on the Plumtree portal (http://www.plumtree.com/).

P1074: Thick Clients

A thick client (often called "fat client") is a client machine in a client/server environment that performs most or all of the application processing with little or none performed in the server.

Guidance

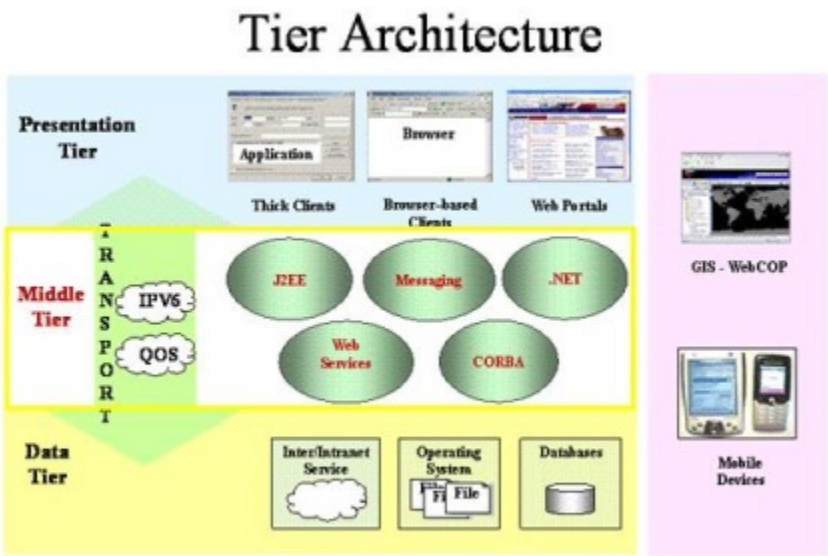
- G1030: Use a standard GUI **component** library.

References

- R1077: AWT - http://java.sun.com/products/jdk/awt/
- R1078: Swing - http://java.sun.com/products/jfc/
- R1079: Thick client - http://www.jargonsoft.com/m2/tech/JargonWhitePaper.html

P1052: Middle Tier

The middle tier provides process management services such as process development, monitoring, and resourcing that are shared by multiple applications.



Detailed Perspectives

Messaging

- Message-Oriented Middleware (MOM)
- Message-Based Applications
- Messaging with MSMQ

Web Services

- Web Services with.NET
- SOAP
- WS-I Compliance
- WSDL
- Insulation and Structure
- Error Handling
- Universal Description, Discovery, and Integration (UDDI)

Java EE Environment
.NET Framework
CORBA
Software Communication Architecture

DDS

P1047: Messaging

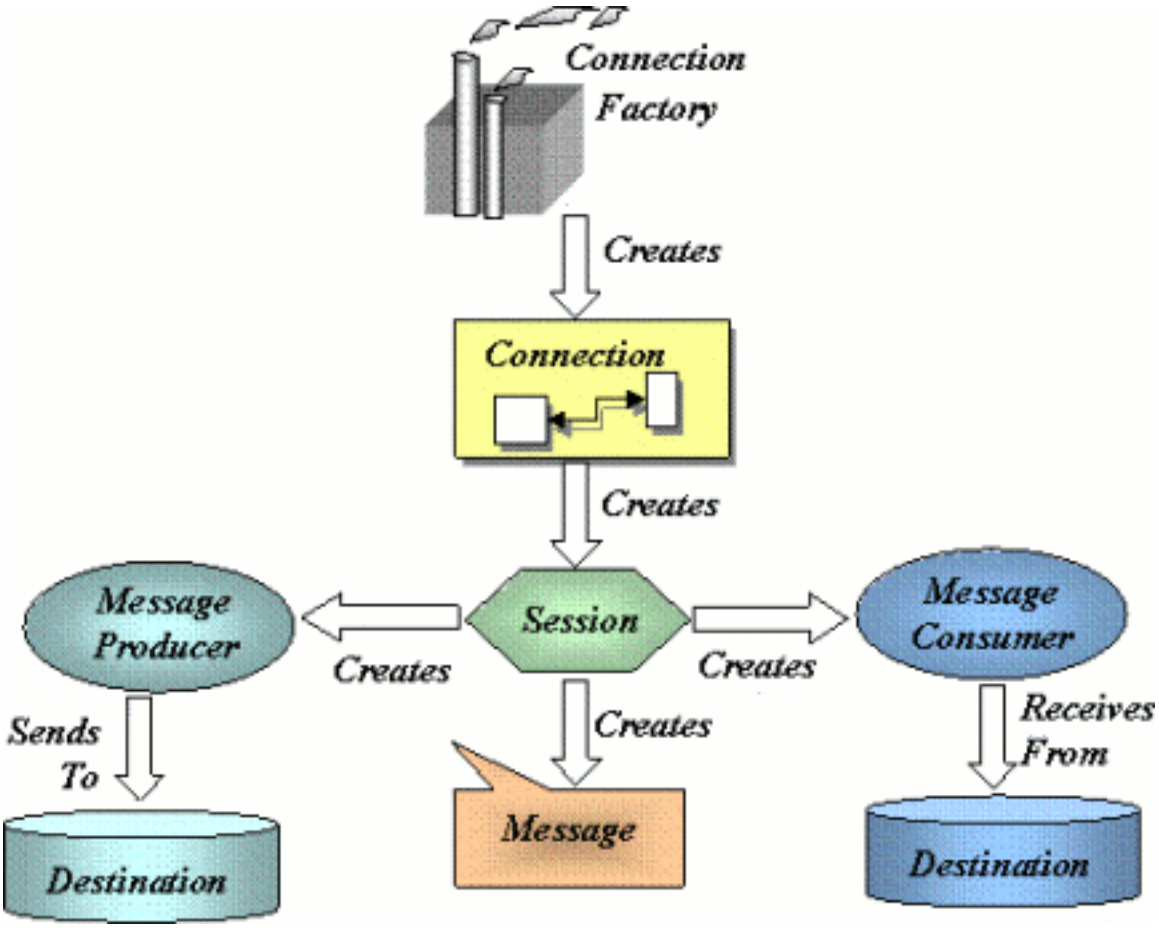
The explosion of the Internet required applications to communicate and interoperate with other applications and services. Messaging systems play an important role in enterprise applications because computers and networks are inherently unreliable and messaging systems are perfectly suited to operate in disconnected environments. They provide a reliable, secure, event-driven message-delivery communication mechanism. Unlike traditional **RPC**-based systems (**RMI** or **CORBA**), most message-oriented based systems operate peer-to-peer.

The messaging paradigm offers three major advantages:

- Allows applications to communicate asynchronously. This means the system sending the **message** does not have to wait around for a response.
- Provides more robustness and reliability; messages do not get lost if a **client** has crashed or is unavailable.
- Multiplexes messages and sends them to multiple clients.

There are other advantages such as transactional message support, message prioritization, load balancing, and firewall **tunneling**. However, these features usually depend on how the **Message-Oriented Middleware** (**MOM**) is implemented.

This diagram shows the relationship of the classes and interfaces in the **Java Message Service** (**JMS**) **API**. Developers use these classes and interfaces to create a JMS application.



P1046: Message-Oriented Middleware (MOM)

Message-oriented middleware acts as an arbitrator between incoming and outgoing **messages** to insulate producers and consumers from other producers and consumers A **MOM** typically is implemented using proprietary **protocols** and interfaces, which means that different implementations are usually incompatible. Using a single implementation of a MOM in a system typically leads to dependence on the MOM vendor for maintenance, support, and future enhancements. Maturing standards such as **Java Message Service** (**JMS**) and **SOAP Web services** are reducing vendor dependencies by standardizing message content and providing standard interfaces to the various MOM **APIs**.

Advantages

- A MOM provides a common reliable way for programs to create, send, receive, and read messages in any distributed enterprise system.
- A MOM ensures fast, reliable, asynchronous communications, guaranteed message delivery, receipt notification, and transaction control.
- A MOM increases the interoperability, portability, and flexibility of an application by allowing it to be distributed over multiple heterogeneous platforms.
- A MOM enables applications to exchange messages with remote programs without having to know on what platform or processor the other application resides.

Disadvantages

- A MOM does not help with interoperability directly, as applications need to agree on message content and format at development time.
- The current marketplace is filled with proprietary implementations of features, so moving between MOMs usually requires recoding; JMS and other standard interfaces help in this area but do not usually cover all of the vendor's extended functionality.

Features

Guaranteed message delivery	MOMs provide a message queue between interoperating processes. If the destination process is busy or offline, the message is held in a temporary storage location until it can be processed.
Asynchronous and synchronous communications	MOMs allow multitasking. Once an application sends out a message to a receiving application, the MOM allows the **client** application to handle other tasks without waiting for a response from the receiving application. Supports blocking method calls.
Transaction support	Most MOMs support transactions.
One-time, in-order delivery	MOMs guarantee that each message will be delivered once and that messages are received in the order in which they are sent.
Message routing services	MOMs support least-cost routing and can reroute around network problems.

Notification Services	MOMs provide audit trails, journaling, and notifications when messages are received.

Message models

The most important aspect of a message-based communication system is the message. The most common messaging models are the following:

- Point-to-Point (p2p)
- Publish/Subscribe (pub/sub)
- Request-Reply

References

- R1045: See the Microsoft MSDN document .NET Compact Framework for detailed comparisons between the **.NET Compact Framework** and the **.NET** Framework.

P1045: Message-Based Applications

Developers need to understand the types of applications that are best suited for **message**-based systems so they can understand how best to use messaging to enterprise applications. Three types of applications follow:

- Workflow
- Event-driven
- Disconnected

Guidance

- G1117: Isolate topic and queue names by not hard-coding them in **client** code.

Best Practices

- BP1116: If using **Java**-based messaging (e.g., **JMS**), register destinations in **Java Naming and Directory Interface** (**JNDI**) so **message clients** can use JNDI to look up these destinations.

Examples

Most **JMS** interoperability coding issues relate to the use of **JNDI** for resources. You can mitigate these issues by encapsulating resource definitions in a properties file or in **Java EE** as a **deployment descriptor**. The following table lists the vendor-specific syntax for specifying resources.

Vendor	JNDI properties
WebLogic 8.1 sp2	java.naming.factory.initial=WebLogic .jndi.WLInitialContextFactory java.naming.provider.url=t3://localhost:7001
JBoss 3.2.3	java.naming.factory.initial=org.jnp.interfaces.NamingContextFacto java.naming.provider.url=jnp://localhost:1099
WebSphere 5.1	java.naming.factory.initial=com.ibm.websphere.naming.WsnInitial java.naming.provider.url=iiop://localhost:2809
Sonic 5.0.2	java.naming.factory.initial=com.sonicsw.jndi.mfcontext.MFContex java.naming.provider.url=tcp://localhost:2506 com.sonicsw.jndi.mfcontext.domain=testdomain
Fiorano 7.2	java.naming.factory.initial=fiorano.jms.runtime.naming.FioranoInit java.naming.provider.url=http://localhost:1856 java.naming.security.principal=anonymous java.naming.security.credentials=anonymous
Joram 4.0	java.naming.factory.initial=fr.dyade.aaa.jndi2.client.NamingConte

	java.naming.provider.url=joram://localhost:16400

Using JMS

Creating a JMS sender and receiver application

The previous sections have reviewed basic **JMS** terminology and interfaces. We are ready to put it all together and see how to create a JMS sender and receiver application.

Process for creating a JMS Sender

To write a basic **JMS** sender application:

1. Perform a lookup through **Java Naming and Directory Interface** (**JNDI**) to get a connection factory.
2. Perform a lookup through Java Naming and Directory Interface (JNDI) to find a destination (Queue or Topic).
3. Using the connection factory obtained in step 1, create a connection to the JMS provider.
4. Create a session by using the connection created in step 3.
5. Create a **message** producer (or) using the session created in step 4 and the destination created in step 2.
6. Create and send the message with the message producer created in step 5. For a queue, use the send method. For a topic, use the publish method.

Process for creating a JMS Receiver

To write a basic **JMS** receiver application:

1. Perform a lookup through **Java Naming and Directory Interface** (**JNDI**) to get a connection factory.
2. Perform a lookup through Java Naming and Directory Interface (JNDI) to find a destination (Queue or Topic).
3. Using the connection factory you obtained in step 1, create a connection to the JMS provider.
4. Create a session by using the connection created in step 3.
5. Create a **message** consumer (or) using the session created in step 4 and the destination created in step 2.
6. For asynchronous operations, create a custom message listener. Attach it (set) to the desired message consumer (Queue or Topic). For synchronous operations, use the receive method of the Receiver.
7. When a message is available, the method of the message listener will be called for asynchronous operations. For synchronous operations, the blocking receive method will return a Message object.

JMS client

AbstractThread.java

```
package util;
public abstract class AbstractThread extends Thread {
   private boolean killed = false;
   private boolean paused = false;
   /**
```

```
     * Creates a new thread by calling corresponding
     * constructor in java.lang.Thread.
     */
    public AbstractThread() {
       super();
    } // End AbstractThread
    /**
     * Creates a new thread by calling corresponding
     * constructor in java.lang.Thread.
     */
    public AbstractThread ( String name ) {
       super( name );
    } // End AbstractThread
    /**
     * Creates a new thread by calling corresponding
     * constructor in java.lang.Thread.
     */
    public AbstractThread ( ThreadGroup group, String name ) {
       super( group, name );
    } // End AbstractThread
    /**
     * Replacement for the deprecated method stop().
     * Sets the killed property to true and notifies
     * all waiting threads.
     */
    synchronized public void kill() {
       killed = true;
       notifyAll();
    } // End kill
    /**
     * Replacement for the deprecated method suspend().
     * Sets the paused property to true.
     */ synchronized
    public void pause() {
       paused = true;
    } // End pause
    /**
     * Replacement for the deprecated method resume().
     * Sets the paused property to false and notifies
     * all waiting threads.
     */
    synchronized public void unpause() {
       paused = false;
       notifyAll();
    } // End unpause
    /**
    * This thread's wait method. Called to force the
     * thread to wait to be notified. It is meant to be
     * used in the wait/notify scheme for the current
     * thread.
     *
     * For example, this thread can wait when it has
     * nothing to do and when notified, can wake up,
     * process something, and then wait again.
     */
    synchronized public void waitToBeNotified() {
       try {
          wait();
       } catch(InterruptedException ie) {
       }
    } // End waitToBeNotified
    /**
     * Determines if the thread has been killed.
     */
    public boolean isKilled() {
       return killed;
    } // End isKilled
    /**
     * Determines whether the thread is currently paused.
     */
    public boolean isPaused() {
       return paused;
    } // End isPaused
 } // End AbstractThread
```

JmsConsumer.java

```
package client;
import util.AbstractThread;
import javax.naming.InitialContext;
import javax.jms.ConnectionFactory;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.TextMessage;
import javax.jms.Destination;
import javax.jms.Connection;
import javax.jms.Session;
import java.util.LinkedList;
/**
 * Standalone java jms consumer that receives
 * text messages from a test queue or a test
 * topic. This is just a sample consumer so it
 * uses default settings where possible and
 * does not account for advanced jms functionality.
 */
public class JmsConsumer
   extends AbstractThread
   implements MessageListener {
   private LinkedList inbox;
   private MessageConsumer consumer;
   private Connection connection;
   private TextMessage msg;
   /**
    * constructor - sets up jms connections.
    * All JNDI properties are configured using
    * the jndi.properties file. This file needs
    * to reside in the topmost directory of the
    * classpath because it has no package associated
    * with it.1
    * @param connectionFactory the JNDI name of
    * the jms connection factory
    * @param destinationName the JNDI name of the
    * jms topic or queue
    */
   public JmsConsumer ( String connectionFactory, String destinationName )
      throws Exception {

      // create thread safe list to hold jms messages
      inbox = new LinkedList();
      // The javax.naming.* package contains a mechanism
      // that automatically puts jndi parameters into the
      // initial context from a properties file.
      // The properties file should be named jndi.properties
      // and placed in the top level directory of the classpath.
      // see javax.naming.Context for further discussion
      InitialContext ictx = new InitialContext();

      // jms destination (topic or queue)
      System.out.println( JmsConsumer - looking up jms destination: + destinationName);
      Destination destination = (Destination) ictx.lookup ( destinationName );
      // jms factory
      System.out.println ( JmsConsumer - looking up jms connection factory: + connectionFactory );
      ConnectionFactory factory = (ConnectionFactory) ictx.lookup ( connectionFactory );

      // jms connection
      connection = factory.createConnection();

      // jms session
      // params = transactional, acknowledgement of
      // message received
      Session session =
         connection.createSession ( false, Session.AUTO_ACKNOWLEDGE );
      // jms consumer for given destination
      consumer = session.createConsumer(destination);
      consumer.setMessageListener(this);
      // create reusable text message
      msg = session.createTextMessage();
      // done with context ictx.close();
      // start connection - this only needs to be done
```

```
      // for consumers, not producers
      System.out.println ( JmsConsumer - starting jms connection );
      connection.start();
   } // End JmsConsumer
   /**
    * run
    */
   public void run() {
      boolean startFlag = true;
      while (!isKilled()) {
         // only here to print initial message
         if (startFlag) {
           System.out.println("JmsConsumer - done");
           System.out.println("******************************\n");
           startFlag = false;
         } // End if
         // check internal message queue and then wait for notify()
         // to be called from the jms callback onMessage() method
         if (isEmpty()) {
           waitToBeNotified();
           if (isKilled())
              break;
        } // End if
        try {
           TextMessage msg = (
             TextMessage)retrieveMessage();
           System.out.println( "JmsConsumer - got message (" + msg.getText() + ")");
        } // End try
        catch ( Exception exception) {
           System.out.println ( "JmsConsumer - error in run method" );
           System.out.println ( exception.toString() );
        } // End catch exception
      } // End while loop
   } // End run
   /**
    * kill
    */
   public void kill() {
      System.out.println ( "\n******************************" );
      System.out.println ( "JmsConsumer - thread stopping" );
      super.kill();
      try {
         connection.close();
      } // End try
      catch ( Exception exception ) {
         // Do nothing
      } // End catch Exception
      System.out.println ( "JmsConsumer done" );
      System.out.println ( "******************************\n" );
   } // End kill

   /**
    * finalize
    */
   public void finalize() {
      kill();
   } // End finalize
   /**
    * Adds a new object to the internal queue
    * @param obj the object to be added to the queue.
    */
   private synchronized void storeMessage ( Object messageObject ) {
     inbox.addLast ( messageObject );
   } // End storeMessage

   /**
    * Removes an object from the internal queue
    * @return the next object on the queue.
    */
   private synchronized Object retrieveMessage() {
     return inbox.removeFirst();
   } // End retrieveMessage
   /**
    * Is internal queue empty
    */
```

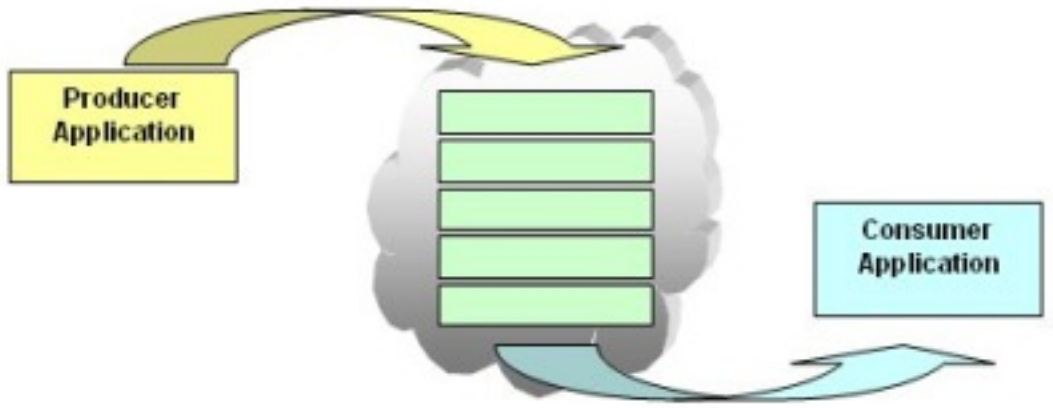
```
    private synchronized boolean isEmpty() {
      return inbox.isEmpty();
    } // End isEmpty
    /**
     * From MessageListener interface. This method
     * is called by jms when a message arrives on
     * the jms destination that this is subscribed to.
     * @param msg Message object from jms
     */
    public void onMessage ( javax.jms.Message msg ) {
      try {
        storeMessage(msg);
        // wake up and process
        synchronized ( this) {
          notify();
        } // End synchronized block
      } // End try
      catch ( Exception exception ) {
        System.out.println ( "JmsConsumer - error in onMessage method" );
        System.out.println ( exception.toString() );
      } // End catch Exception
    } // End onMessage
    /**
     * main
     */
    public static void main ( String argv[] ) {
      System.out.println ( "\n*****************************" );
      System.out.println ( "JmsConsumer starting" );
      JmsConsumer consumer = null;
      try {
        consumer = new JmsConsumer ( argv[0], argv[1] );
        consumer.start();
      } // End try
      catch (Exception exception ) {
        System.out.println ( "JmsConsumer - error in main method" );
        System.out.println ( exception.toString() );
        consumer.kill();
      } // End catch exception
    } // End main
  } // End JmsConsumer
```

References

- R1043: Java Naming and Directory Interface - http://java.sun.com/products/jndi/
- R1044: Erdogan, Levent. "Java Message Service (JMS) for J2EE." New Riders Publishing, August 14, 2002.

P1048: Messaging with MSMQ

Messaging in **.NET** uses Microsoft Message Queue (**MSMQ**). MSMQ is responsible for reliably delivering **messages** between applications inside and outside the enterprise. MSMQ ensures reliable delivery by placing messages that fail to reach their intended destination in a queue and then resending them once the destination is reachable.



MSMQ also supports transactions. It permits multiple operations on multiple queues, with all of the operations wrapped in a single transaction, thus ensuring that either all or none of the operations will take effect. Microsoft Distributed Transaction Coordinator (MSDTC) supports transactional access to MSMQ and other resources.

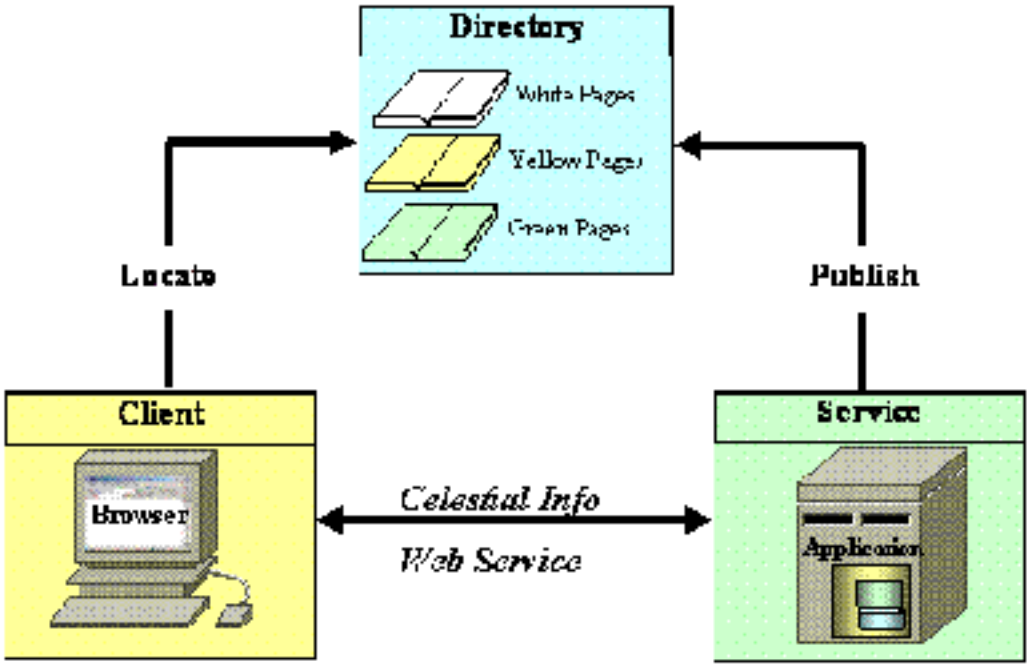
Best Practices

- BP1111: Mark all **Microsoft Message Queue** (**MSMQ**) messages as recoverable.
- BP1112: Specify all **Microsoft Message Queue** (**MSMQ**) queues as transactional if they support multiple-step processes.
- BP1227: Do not allow installation of **MSMQ**-dependent clients.
- BP1230: Do not use the **MSMQ** **`SupportLocalAccountsOrNT4`** feature.

P1078: Web Services

A **Web service** is an application that exists in a distributed environment, such as the **Internet**. A Web service accepts a request, performs its function based on the request, and returns a response. The request and the response can be part of the same operation, or they can occur separately in which case the consumer does not need to wait for a response. Both the request and the response usually take the form of **XML**, use a portable data-interchange format called **SOAP**, and are delivered over a **wire protocol**, such as **HTTP**.

A Web service can reside on top of existing legacy applications and expose services to the net. The Web services architecture illustrated below implements the **service-oriented architecture** pattern. For more information on design patterns, see "Web Service Patterns: Java Edition" by Paul B. Monday.

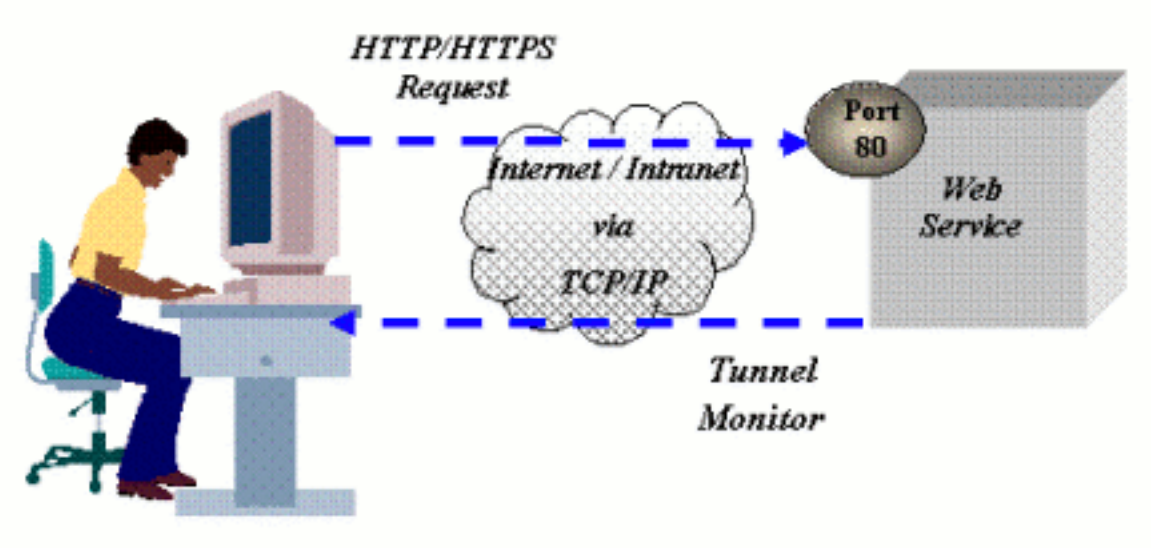


Web Service Models

Web services have traditionally been used to connect people to **services**. However, as the Web service **infrastructure** has matured, a new model has emerged, the service-to-service model.

Traditional Model

In a classic Web service, a request is usually made to a Web service using a **Web browser**. The request is submitted to the Web service using **HTTP** or **HTTPS** over the **Internet** or an **intranet**. The Web service processes the request and returns an **HTML** page that can be displayed in a Web browser.

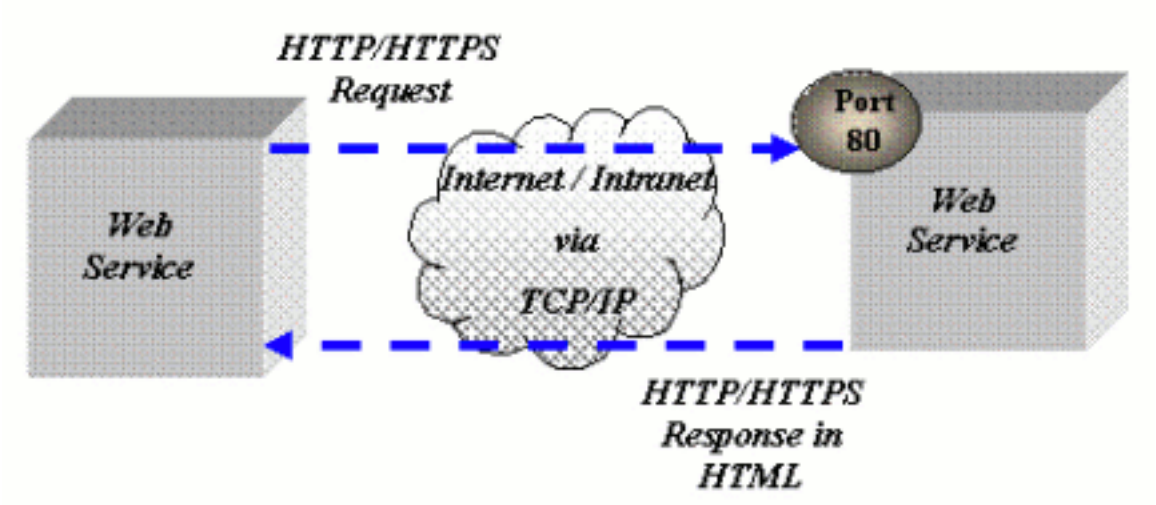


A classic Web service has the following characteristics:

- **Web pages** appear via a Web browser
- Connection is via **TCP/IP**
- Transport is HTTP/HTTPS
- Message format is HTML

Service-to-Service model

Application servers used to be responsible for providing machine-to-machine services. Now **Web servers** can handle similar work. The Web server can pass a request as an **XML** payload embedded in a TCP/IP and HTTP request, process the data, and respond. The response is typically in the form of an HTML Web page or an XML payload that a **client** application can use.



Machine-to-machine Web services have the following characteristics:

- Two independent applications

- Two independent **servers**
- Connection is via TCP/IP
- Transport is HTTP (port 80)
- Message format is XML payload in **SOAP** format

Key characteristics

Some key characteristics of Web services include the following:

- High-overhead interactions; may be too heavy for some applications
- **Loosely coupled** collaborators (e.g., client/server)
- Multiple layers of **parsing**, **marshalling**, and un-marshalling
- Non-standard content
- Standard interaction **protocol**
- No support for **services** such as **messaging** and security
- Infant technology
- No support for pass-by-reference

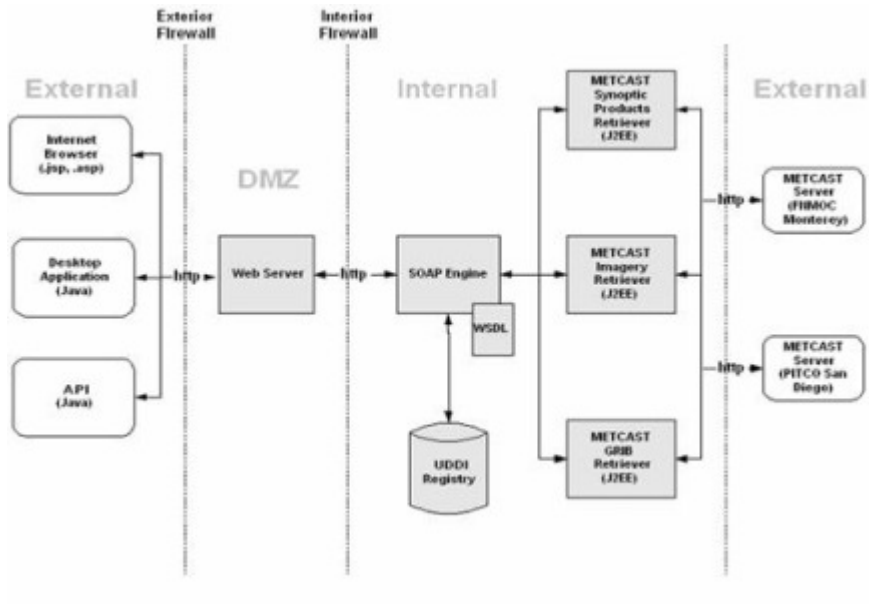
Guidance

- G1087: Validate all **Web Services Definition Language** (**WSDL**) files that describe **Web services**.
- G1088: Use isolation design patterns such as **facade**, **proxy**, or **adapter** to isolate the application from the connection and manipulation of **SOAP** messages.
- G1090: Do not hard-code a **Web service's endpoint**.

Examples

Navy operational example: Exposing Web services for METOC

The following figure shows a simplified example of the architecture, illustrating a METOC metcast application that uses **SOAP** as a **proxy** to legacy content.



References

- R1099: **SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
- R1100: Web Service Definition Language (**WSDL**) - http://www.w3.org/TR/wsdl
- R1101: Adapter pattern - http://c2.com/cgi/wiki?AdapterPattern
- R1102: Design patterns: Proxy - http://www.dofactory.com/Patterns/PatternProxy.aspx
- R1103: Façade pattern - http://c2.com/cgi/wiki?FacadePattern

P1079: Web Services with .NET

.NET Web services use ASP.NET to expose the middle tier's **API** via **SOAP**. .NET Web services also support the **WSDL** 1.1 specification and use a WSDL document to describe it. In this case, however, the WSDL document contains an **XML namespace** that uniquely identifies the Web service's **endpoints**. .NET provides the following:

- A **client**-side **component** that lets an application invoke web service operations described by a WSDL document.
- A **server**-side component that maps Web service operations to method calls as described by a WSDL and a Web Services Meta Language (WSML) file, which is needed for Microsoft's implementation of SOAP.

References

- R1104: For information on .NET vs. J2EE Web services, see http://www.webservicesarchitect.com/content/articles/hanson01.asp.

P1068: SOAP

SOAP is an **XML** message-based **protocol**. It uses **HTTP** to send text commands to **Web services** across the internet. SOAP is lighter weight and requires less programming than similar **protocols** such as **CORBA** and **Distributed Component Object Model** (**DCOM**). The extensible messaging framework is independent of programming models and other implementation-specific semantics.

The **World Wide Web Consortium** (**W3C**) provides this description of SOAP:

"SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics."

Two major design goals for SOAP are simplicity and extensibility. SOAP attempts to meet these goals by omitting distributed-system features from the messaging framework. Such features include but are not limited to reliability, security, correlation, routing, and Message Exchange Patterns (MEPs). While it is anticipated that many features will be defined, this specification provides specifics only for two MEPs. Other features are left to be defined as extensions by other specifications.

Key characteristics

SOAP is **RPC**-based. It offers an **XML**-RPS with extensibility mechanisms; for instance, it allows schemas to define types.

SOAP is an XML document.

SOAP is text-based, providing a standard mechanism for passing through firewalls via the HTTP ports.

There are many SOAP language bindings, and new ones are frequently announced.

SOAP is a **wire protocol** and does not have an activation mechanism. It is inherently stateless.

SOAP does not implement security.

SOAP is case-sensitive and white-space-sensitive.

Message formats

Message styles

The **W3C WSDL** 1.1 Specification identifies two message styles: Document and RPC. The purpose of the styles determines how the content of the SOAP message body is formatted.

Document	The SOAP Body contains one or more child elements called parts. There are no SOAP formatting rules for what the SOAP Body contains; it contains whatever the sender and the receiver agree upon. ***Note:*** *There is a Wrapped form of this style that is required to interoperate with Microsoft* ***Web services*** *using Document style. There is no specification that defines this style.*
RPC	**RPC** implies that the SOAP Body contains an element with the name of the method or remote procedure being invoked. This element in turn contains an element for each parameter of that procedure.

Serialization formats

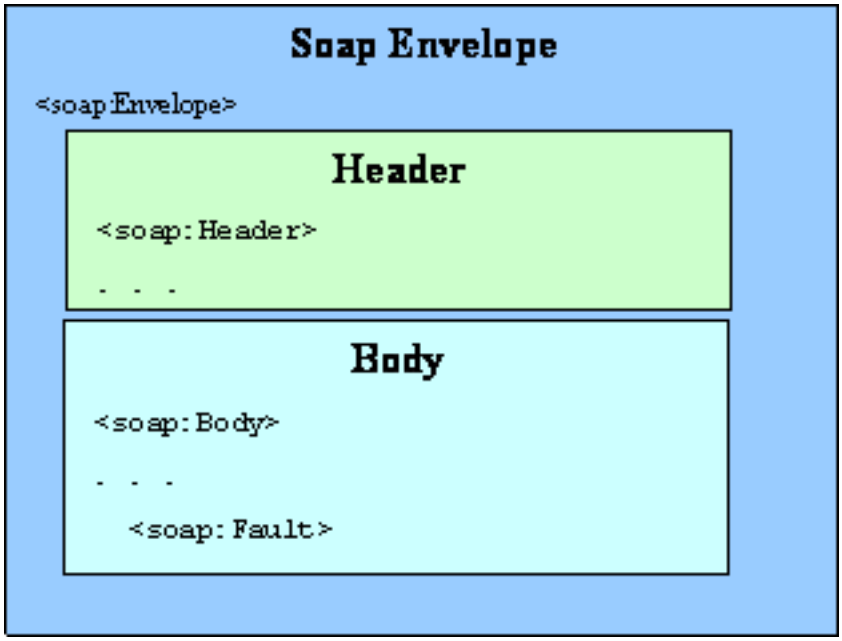
For applications that use **serialization**/**deserialization** to abstract away the data wire format, there is one more choice to be made: the serialization format. The following table describes the two most popular serialization formats today.

SOAP encoding	SOAP encoding uses a set of rules to serialize the data transferred between the **client** and the **server**. The rules are defined in section 5 of the **WSDL** 1.1 Specification. These rules are also referred to as "section 5 encoding." The rules specify how to serialize objects, structures, arrays, and object graphs and directly use the predefined **XML Schema** data types. Generally, an application using SOAP encoding should use the **RPC mssage** style.
Literal	Data is serialized according to an independent external schema. There are no preset rules for serializing objects, structures, and graphics, etc., in the literal encoding style. The industry is overwhelmingly embracing XML Schemas.

Note: *Document style can be interpreted as either an **XML** string or as a W3C **Document Object Model** (**DOM**) Document Element. Microsoft has a technique called Wrapped that encapsulates the information being exchanged, regardless of the style.*

Structure

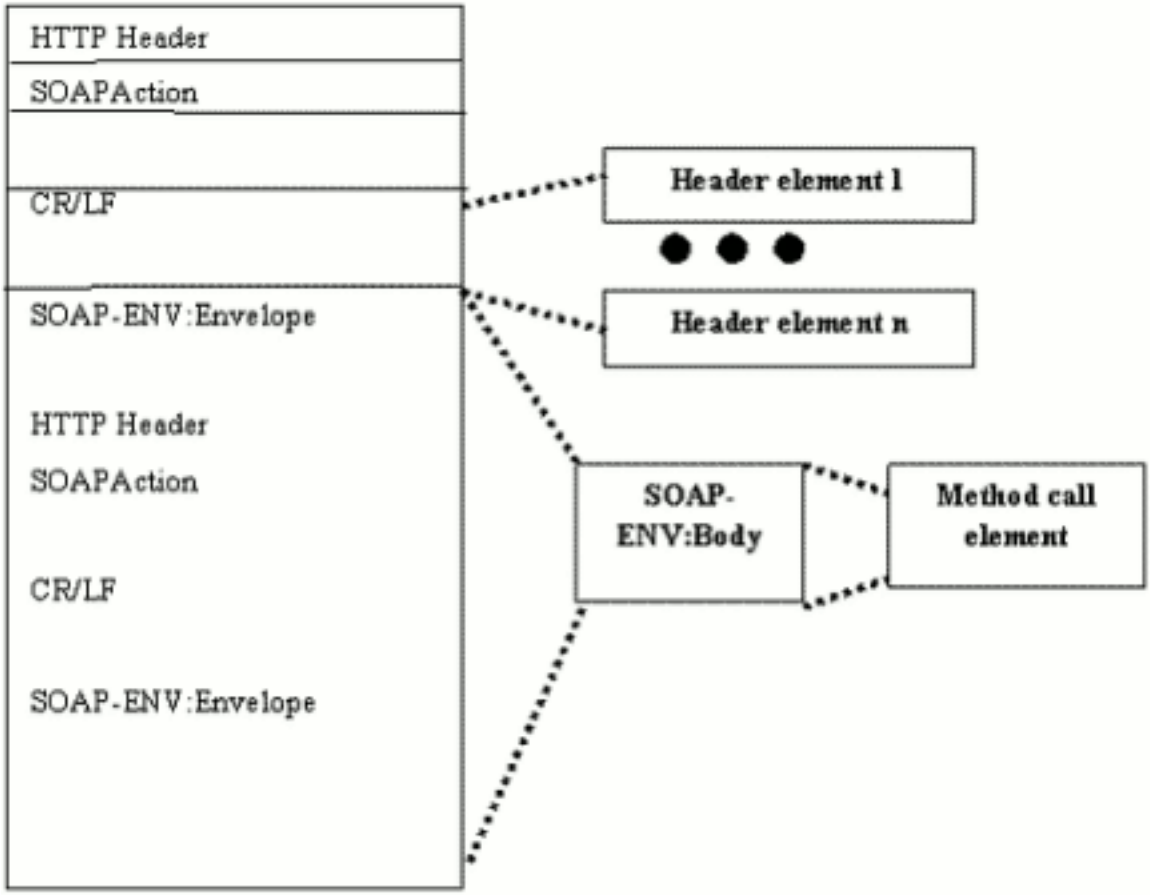
A **SOAP** message comprises three parts: an envelope, an optional header, and a required body. The envelope encapsulates the other two elements. The optional header contains one or more header elements that contain meta-information about the method calls.



Envelope	The Envelope is the root of the SOAP request. At a minimum, it defines the SOAP namespace for SOAP 1.2. The envelope may define additional namespaces.

Header	The Header contains auxiliary information as SOAP blocks, such as authentication, routing information, or transaction identifier. The header is optional.
Body	The Body contains the main information in one or more SOAP blocks; for example, a SOAP block for **RPC** call. The body is mandatory and it must appear after the header.
Fault	The Fault is a special block that indicates a protocol-level error. If present, it must appear within a Body element.

The SOAP payload is encapsulated within the SOAP envelope, which is part of the HTTP payload. The following figure shows an HTTP payload that contains a SOAP message.



Guidance

- G1082: Use the document-literal style for all data transferred using **SOAP** where the document uses the **World Wide Web Consortium** (**W3C**) **Document Object Model** (**DOM**).
- G1084: Validate documents transferred using **SOAP** against the **W3C XML** Standard by an **XML Schema Definition** (**XSD**) defined by the **Community of Interest** (**COI**).
- G1088: Use isolation design patterns such as **facade**, **proxy**, or **adapter** to isolate the application from the connection and manipulation of **SOAP** messages.
- G1093: Ensure **Web services** handle **SOAP** exceptions and faults.
- G1095: Use **W3C** fault codes for all **SOAP** faults.

Examples

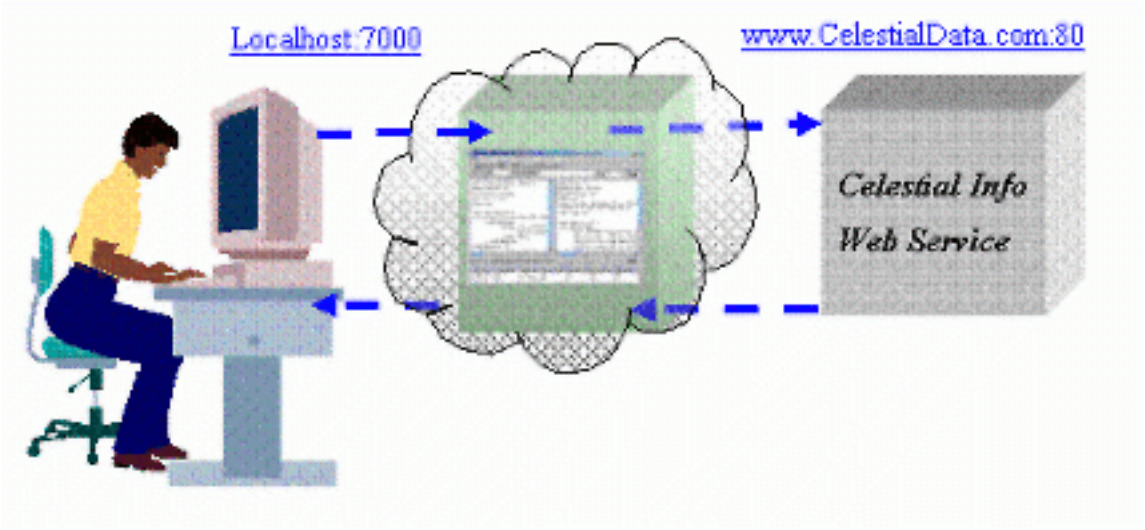
The following is an example of a **Web service client** requesting celestial information about a particular location and receiving the results. Both the request and the response are made using the **WS-I** document literal style of send and receiving **XML SOAP messages**.

These listings are the results of using a **tunnel** monitoring utility called NetTool available from the SourceForge site http://sourceforge.net/projects/nettool/. The tunnel monitoring tool basically interjects itself between the Web service client and the Web service producer. The client connects to the tunnel monitor instead of connecting directly to the producer. The tunnel tool then displays or logs the traffic and forwards it onto the producer. The producer returns the response to the tunnel tool monitor. The response is also displayed or logged and forwarded back to the client.

Monitoring

Without Tunnel

With Tunnel



Request

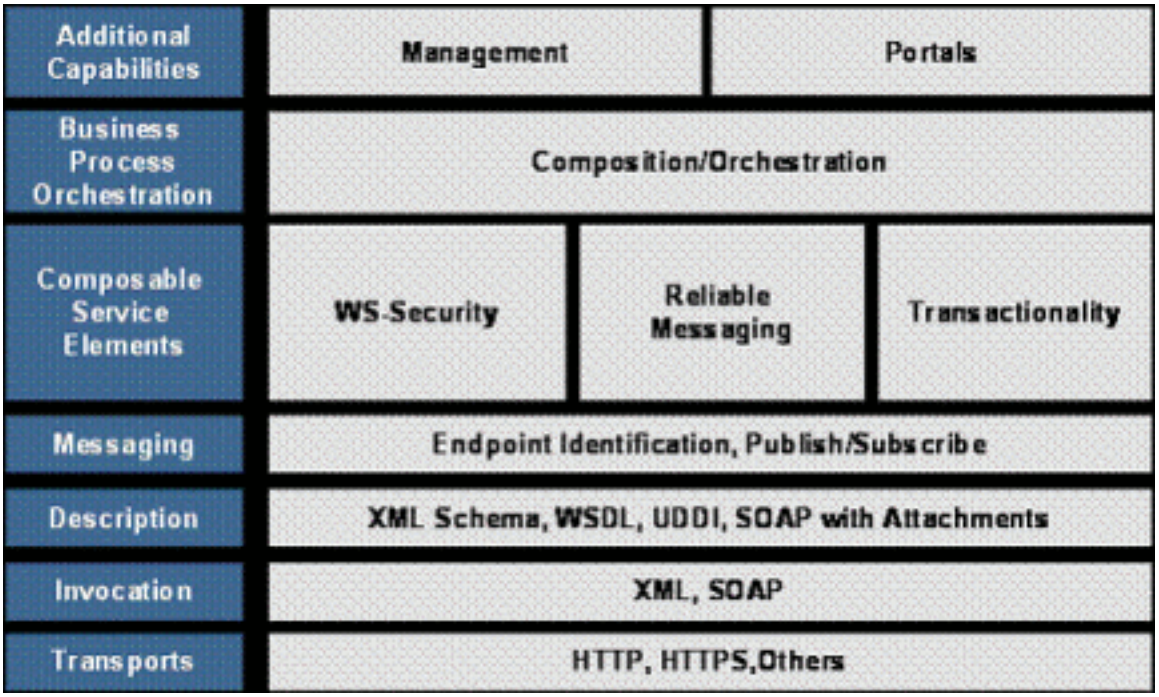
```
POST /DocClientWebProject/BeaServers/CelestialInfoDocDoc.jws
HTTP/1.0Content-Type: text/xml; charset=utf-8
```

```
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: 192.168.2.8:7003
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 597
<xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsi:type="ns1:Document"
POST /DocClientWebProject/BeaServers/CelestialInfoDocDoc.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: 192.168.2.8:7003
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 597
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope


  >
 <soapenv:Body>
   <in0
      xsi:type="ns1:Document"

      xmlns:ns1="http://xml.apache.org/xml-soap">
     <DocumentRequestData>
       <city>San Diego</city>
       <stateOrProvince>California</stateOrProvince>
       <country>USA</country>
       <documentName>CelestialInfoRpt</documentName>
     </DocumentRequestData>
   </in0>
 </soapenv:Body>
</soapenv:Envelope>
```

Response

P1081: WS-I Compliance

The **Web Services Interoperability Organization** (**WS-I**) is an open industry effort. Its charter is to promote **Web services** interoperability across platforms, applications, and programming languages. Its goal is to be a standards integrator to help Web services advance in a structures, coherent manner. WS-I has organized the standards that will affect the interoperability of Web services into a "stack" based on functionalities.



There are many standards that need to be coordinated to address basic Web service interoperability issues and the standards are all being developed in parallel and independently. To overcome these issues, the WS-I has developed the concept of a profile. The WS-I defines a profile as follows:

a set of named Web services specifications at specific revision levels, together with a set of implementation and interoperability guidelines recommending how the specifications may be used to develop interoperable Web services. [http://webservices.sys-con.com/read/39947.htm]

Guidance

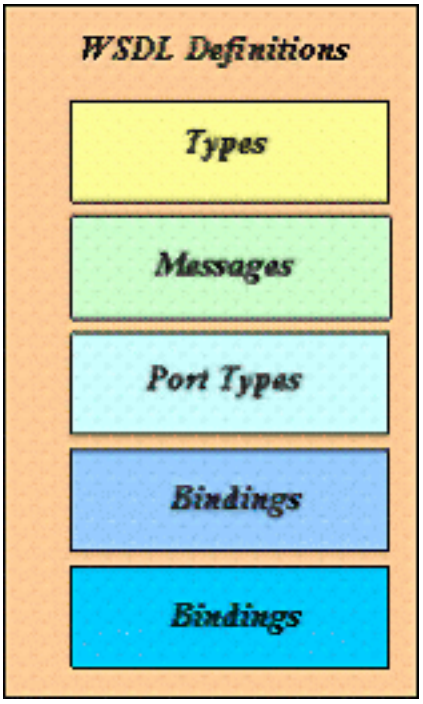
- G1080: Adhere to the Web Services-Interoperability Organization (**WS-I**) Basic Profile specification for **Web Service** environments.
- G1082: Use the document-literal style for all data transferred using **SOAP** where the document uses the **World Wide Web Consortium** (**W3C**) **Document Object Model** (**DOM**).
- G1083: Do not pass **Web Services-Interoperability Organization** (**WS-I**) **Document Object Model** (**DOM**) documents as strings.

References

- R1001: DOM – a Whatis.com definition - http://whatis.techtarget.com/definition/0,,sid9_gci213910,00.html.
- R1105: **SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html

P1082: WSDL

The Web Services Description Language (**WSDL**) is an **XML**-based language that is used to describe a **Web service**. It describes the operations that are available from the Web service and it describes the data that flows between the **client** or consumer of the Web service and the producer of the Web service. In addition, it describes the **endpoint** The **URL** or location of the Web service on the internet of the Web service provider.



Guidance

- G1085: Establish a **registered namespace** in the **XML Gallery** in the **DoD Metadata Registry** for all DoD Programs.
- G1087: Validate all **Web Services Definition Language** (**WSDL**) files that describe **Web services**.
- G1084: Validate documents transferred using **SOAP** against the **W3C XML** Standard by an **XML Schema Definition** (**XSD**) defined by the **Community of Interest** (**COI**).

Examples

The following **Java** interface file:

...can be used to generate the following **WSDL** file:

References

- R1002: **SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
- R1003: Web Service Definition Language (WSDL) - http://www.w3.org/TR/wsdl
- R1004: XSD - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0%2c%2csid26_gci831325%2c00.html

P1035: Insulation and Structure

Insulating the user of **Web services** from the implementation of the services enhances the maintainability and portability of the overall system and aids in the migration to net-centricity. Application developers can use the facade or adapter design pattern for Web services to insulate applications from the implementation details of the service. Services can then change over time to match changing requirements and deployments. Legacy functionality can be similarly wrapped via a service. It is important to not directly expose vendor-specific functionality via the services interface to enable the ready reimplementation of the service if necessary.

Guidance

- G1087: Validate all **Web Services Definition Language** (**WSDL**) files that describe **Web services**.
- G1088: Use isolation design patterns such as **facade**, **proxy**, or **adapter** to isolate the application from the connection and manipulation of **SOAP** messages.
- G1236: Do not hard-code the **endpoint** of a **Web service** vendor.
- G1237: Do not hard-code the configuration data of a **Web service** vendor.

References

- R1029: **SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
- R1030: Web Service Definition Language (WSDL) - http://www.w3.org/TR/wsdl
- R1031: Adapter pattern - http://c2.com/cgi/wiki?AdapterPattern
- R1032: Design patterns: Proxy - http://www.dofactory.com/Patterns/PatternProxy.aspx
- R1033: Façade pattern - http://c2.com/cgi/wiki?FacadePattern

P1022: Error Handling

One of the most sensitive areas for interoperability is handling errors. No one ever plans on having errors, but designing a system which does not handle errors in a common and standard way can be disastrous.

Guidance

- G1093: Ensure **Web services** handle **SOAP** exceptions and faults.
- G1095: Use **W3C** fault codes for all **SOAP** faults.
- G1094: Catch all exceptions for application code exposed as a **Web service**.

Examples

Handling Web service faults

Web service exceptions, known as faults, are handled using standard **XML** tags as discussed in the **W3C SOAP** specification.

> ***Note:*** *The latest version of the SOAP specification (currently 1.2), covers SOAP faults and fault codes.*

The examples in this section show the response from throwing system and SOAP exceptions using **.NET**, BEA WebLogic, and an Axis **client**.

Assumptions

Web services are generated automatically using vendor tools, like an **Integrated Development Environment** (**IDE**). When generating the web service, it is the vendor's responsibility to add a layer that converts standard software-based exceptions to the proper **XML** fault tags before sending the response back to the **client**.

Catch exception block

This is the Catch block that receives the error and generates the sample output shown in these examples.

```
try
{ . . . /// Some code here
} // End try
catch ( Exception exception)
{ System.out.println(exception.getClass().getName());
  org.apache.axis.AxisFault fault
    = (org.apache.axis.AxisFault) exception;
  System.out.println ( "Fault Code: "   + fault.getFaultCode().toString() );
  System.out.println ( "Fault Node: "   + fault.getFaultNode() );
  System.out.println ( "Fault Reason: " + fault.getFaultReason() );
  System.out.println ( "Fault Role: "   + fault.getFaultRole() );
  System.out.println ( "Fault String: " + fault.getFaultString() );
} // End catch Exception
```

Throwing a system exception

The examples on this page show the response from throwing a system exception to an Axis **client** from a .NET **Web service** and a BEA WebLogic Web service.

.NET Web service throwing a fault to an Axis client

This C# code shows a general system exception being thrown from a Web service method.

```
throw new System.Exception
  ( "Fault Occurred" );
The client receives an error like this:
[java] org.apache.axis.AxisFault
[java] Fault Code: {http://schemas.XMLsoap.org/soap/envelope/}Server
[java] Fault Node: null
[java] Fault Reason: System.Web.Services.Protocols.SoapException: Server was unable to
process request. ---> System.Exception: Fault Occurred
[java] Fault Role: null
[java] Fault String: System.Web.Services.Protocols.SoapException: Server was unable to process
request. ---> System.Exception: Fault Occurred
```

BEA WebLogic Web service throwing a fault to an Axis client

This **Java** code shows a general system exception being thrown from a Web service method.

```
throw new System.Exception
  ( "Fault Occurred" );
The client receives an error like this:
[java] org.apache.axis.AxisFault
[java] Fault Code: {http://schemas.xmlsoap.org/soap/envelope/}Server
[java] Fault Node: null
[java] Fault Reason: System.Web.Services.Protocols.SoapException: Server was unable to
process request. ---> System.Exception: Fault Occurred
[java] Fault Role: null
[java] Fault String: System.Web.Services.Protocols.SoapException: Server was unable to process
request. ---> System.Exception: Fault Occurred
```

BEA WebLogic Web service throwing a fault to an Axis client

This Java code shows a general system exception being thrown from a Web service method.

```
throw new java.lang.Exception
  ( "Fault Occurred" );
The client receives an error like this:
[java] org.apache.axis.AxisFault
[java] Fault Code: {http://www.bea.com/2003/04/jwFaultCode/}JWSError
[java] Fault Node: null
[java] Fault Reason:
[java] <xml-fragment
[java]     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
[java]
[java]
[java]    >
[java]    <faultcode
[java]        >fc:JWSError
[java]    </faultcode>
[java]    <faultstring>
[java]         Fault Occurred
[java]    </faultstring>
[java]    <detail>
[java]        <jwErr:jwErrorDetail
[java]            >
[java]             java.lang.Exception: Fault Occurred
[java]             at test.exceptions.ex.thisWillThrowException()V(ex.jws:13)
[java]        </jwErr:jwErrorDetail>
[java]    </detail>
[java] </xml-fragment>
[java] Fault Role: null
[java] Fault String:
[java] <xml-fragment
[java]     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
[java]
[java]
[java]    >
[java]    <faultcode
[java]        >fc:JWSError
```

```
[java]    </faultcode>
[java]    <faultstring>
[java]         Fault Occurred
[java]    </faultstring>
[java]    <detail>
[java]        <jwErr:jwErrorDetail
[java]            >
[java]             java.lang.Exception: Fault Occurred
[java]             at test.exceptions.ex.thisWillThrowException()V(ex.jws:13)
[java]        </jwErr:jwErrorDetail>
[java]    </detail>
[java] </xml-fragment>
```

Throwing a SOAP exception

.NET Web service throwing a SOAP exception to an Axis client

This C# code shows a **SOAP** exception being thrown from a **Web service** method.

```
throw new System.Web.Services.Protocols.SoapException
  ( "Fault Occurred",
    System.Web.Services.Protocols.SoapException.ClientFaultCode,
    Context.Request.Url.AbsoluteUri
  );
```

The **client** receives an error like this:

```
[java] org.apache.axis.AxisFault
[java] Fault Code: {http://schemas.xmlsoap.org/soap/envelope/}Client
[java] Fault Node: null
[java] Fault Reason: System.Web.Services.Protocols.SoapException: Fault Occurred
[java] Fault Role: http://localhost:15623/server/CelestialInfoDocDocImpl.asmx
[java] Fault String: System.Web.Services.Protocols.SoapException: Fault Occurred
```

BEA WebLogic Web service throwing a SOAP exception to an Axis client

This Java code shows a SOAP exception being thrown from a Web service method.

```
throw new javax.xml.rpc.soap.SOAPFaultException
  ( new javax.xml.namespace.QName("", "Client"),
    "Fault Occurred",
    "",
    Null
  );
```

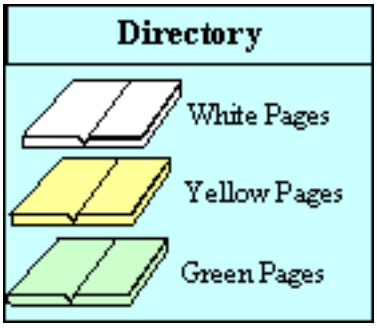
The client receives an error like this:

```
[java] org.apache.axis.AxisFault
[java] Fault Code: {http://www.bea.com/2003/04/jwFaultCode/}JWSError
[java] Fault Node: null
[java] Fault Reason:
[java] <xml-fragment xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
[java]
[java]
[java]    >
[java]    <faultcode>
[java]         Client
[java]    </faultcode>
[java]    <faultstring>
[java]         Fault Occurred
[java]    </faultstring>
[java]    <faultactor/>
[java] </xml-fragment>
[java] Fault Role: null
[java] Fault String:
```

```
[java] <xml-fragment xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
[java]
[java]
[java]    >
[java]    <faultcode>
[java]         Client
[java]    </faultcode>
[java]    <faultstring>
[java]         Fault Occurred
[java]    </faultstring>
[java]    <faultactor/>
[java] </xml-fragment>
```

P1075: Universal Description, Discovery, and Integration (UDDI)

The **Universal Description, Discovery, and Integration** (**UDDI**) standard is an industry initiative for a **Web services** registry. It enables businesses to access a universal pool of Web services. The UDDI registry contains yellow pages, white pages, and so-called "green pages," like a phone book.



White pages	List point of contact information, such as • Name • Address • Phone • Fax • email
Yellow pages	List services that are available from businesses, such as • Weather data • Software development • Project management
Green pages	List service properties, such as • Business processes • Service descriptions • Binding information • Categorization of services • XML version, type of encryption, and Document Type Definition (DTD)

UDDI is a platform-independent, open framework that allows automated consumers and suppliers to find each other, assess mutual compatibilities, negotiate terms, and build the relationship. It supports human interaction as well as machine-to-machine communication. People can use a UDDI browser to review services and find point-of-contact information (white pages), and business information (yellow pages).

Like the **Domain Name System** (**DNS**), the UDDI registry comprises a network of **servers** on the internet. It is a **SOAP**-based mechanism. The **API** specification focuses on the storage, organization, and architecture of the registry.

The UDDI project takes advantage of **World Wide Web Consortium** (**W3C**) and **Internet Engineering Task Force** (**IETF**) standards such as **eXtensible Markup Language** (**XML**) and **HTTP** and Domain Name System (DNS) **protocols**.

Guidance

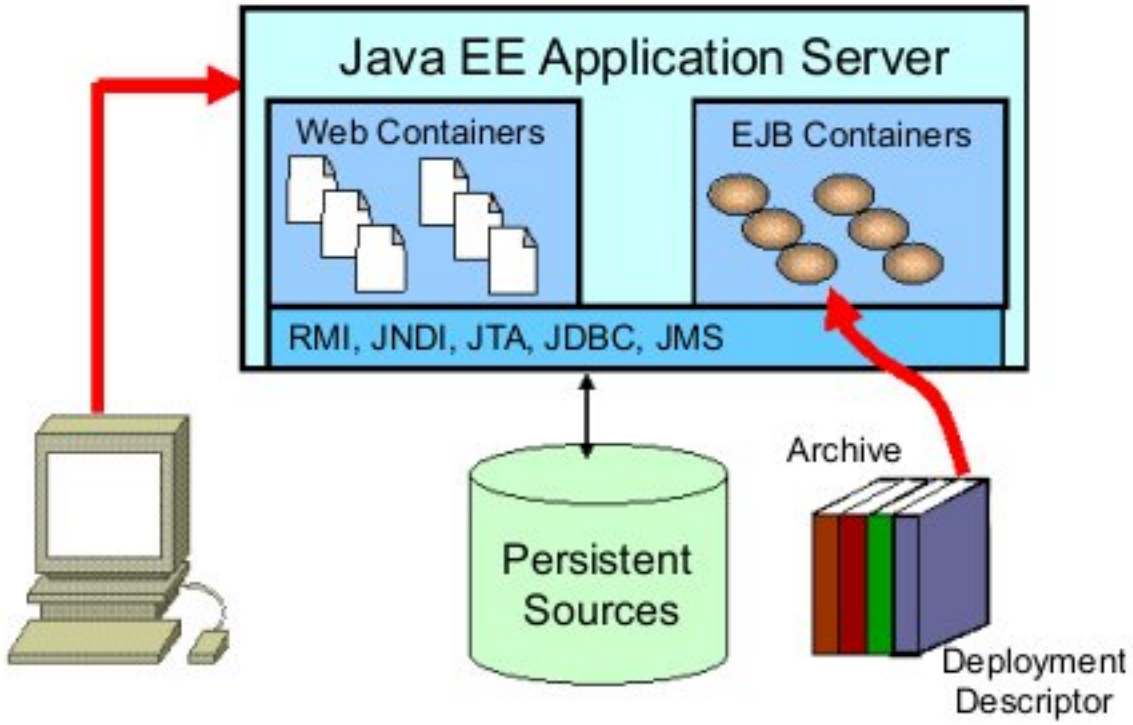
- G1127: Use a **UDDI** specification that supports publishing discovery services.
- G1131: Use industry standard Universal Description, Discovery, and Integration (**UDDI**) **APIs** for all UDDI inquiries.

References

- R1080: OASIS – a Whatis.com definition - http://searchvb.techtarget.com/gDefinition/0%2c294236%2csid8_gci527425%2c00.html
- R1081: What is **UDDI**? - http://www.webopedia.com/TERM/U/UDDI.html

P1037: Java EE Environment

Java has been extended to handle the complexity of **enterprise** computing through the **Java Enterprise Edition** (**Java EE**, formerly termed **Java 2 Enterprise Edition** or **J2EE**). In the Java EE environment, packaging and **deployment** is done using a Java archive file. A Java archive file is a self-contained module that contains all of an application's **Java class files**, static files, and **deployment descriptor** files. **Java archive** files are created using a **jar** utility. There are multiple deployment descriptors that correspond to the type of modules being deployed as indicated in the table below using the Java EE specification.



The table below shows the Java EE standard deployment descriptor files and the specific applications to which they apply. See http://java.sun.com/dtd/ for details of each XML file.

Component or Application	Scope	Deployment descriptors	Packaging Archives
Web application	Java EE	web.XML	.**war**
Enterprise bean	Java EE	ejb-jar.XML	.**jar**
Resource adapter	Java EE	ra.XML	.**rar**
Enterprise application	Java EE	application.XML	.**ear**
Client application	Java EE	application-client.XML	

The format for a deployment descriptor is defined in both the **EJB** specification and the **servlet** specification. The Sun standards are defined at the following locations:

Java EE environment applications	http://java.sun.com/products/ejb/docs.html
Non-JavaEE or standard Webapplications	http://java.sun.com/products/servlet/download.html

> ***Note:*** *Some vendors have extensions to the Java EE deployment descriptors or have specific additional descriptors for their products. Refer to specific vendor documentation for these details.*

Guidance

- G1078: Document the use of non-**Java EE**-defined **deployment descriptors**.
- G1079: Isolate tailorable data values into the **deployment descriptors** for **Java EE** applications.
- G1209: For Java, use **JDK** logging facilities.

Best Practices

- BP1076: When **deploying** a new application to a WebLogic **application server** (e.g., **ear**, **war**, **rar**), do not edit the WebLogic startup file to add application-specific information. This file is used for **server** startup only and should not contain application-specific logic. The system administrator must approve and coordinate all updates to this file.
- BP1077: Do not edit the **`config.xml`** file manually.

Examples

Environment entries

Enterprise JavaBeans (**EJB**) **environment values** are defined in the **deployment descriptor** using the env-entry element. Use **Java EE** provider utilities to modify these values during or after **deployment**.

A bean can access the environment entries with a similar code to the following:

Resource references

Use resource references to define and use environment entries. By default, the initial Java EE environment context is java:comp/env/. Consequently, it is best to classify all resources into subcontexts of the default. For example, classify all **JDBC** definitions using the default context with a JDBC subcontext appended to it. For example:

```
java:comp/env/jdbc
In the standard deployment descriptor, the declaration of a resource reference to a JDBC connection
factory is:
 <resource-ref>
   <res-ref-name>jdbc/JTMDS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
 </resource-ref>
```

And the **EJB** accesses the data source as in the following:

Resource Environment References

- The **`resource-env-ref`** describes administered objects, as opposed to objects that are better maintained programmatically. Administered objects help define objects that are likely to change between implementations: for example, **JMS** or database implementations. It is best to administer these objects along with other administrative tasks that vary from provider to provider and not within the application. This makes the code more portable.

The code to access the administered object follows:

Example Deployment Descriptors

ejb-jar.xml

web.xml

```
/* Descriptor for Application named: HelloWorld.jsp */
MyWebApp/  (public directory)
   HelloWorld.jsp
WEB-INF/
  Web.XML
  Classes/myBean
<?XML version="1.0" encoding="UTF-8"?>
<web-app>
 <display-name>HelloWorldJSP</display-name>
 <servlet>
   <servlet-name>HelloWorld</servlet-name>
   <display-name>HelloWorld</display-name>
   <jsp-file>/HelloWorld.jsp</jsp-file>
 </servlet>
 <session-config>
   <session-timeout>30</session-timeout>
 </session-config>
 <ejb-ref>
   <ejb-ref-name>ejb/helloejb</ejb-ref-name>
   <ejb-ref-type>Session</ejb-ref-type>
   <home>HelloHome</home>
   <remote>Hello</remote>
 </ejb-ref>
</web-app>
    Contact.class
```

References

- R1034: Java EE - http://java.sun.com/javaee/
- R1035: EJB - http://java.sun.com/products/ejb/
- R1036: .jar - http://java.sun.com/developer/Books/javaprogramming/JAR/
- R1037: .war - http://access1.sun.com/techarticles/simple.WAR.html
- R1038: .ear - http://java.sun.com/J2EE/tutorial/1_3-fcs/doc/Overview4.html
- R1039: .rar - http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Connector2.html

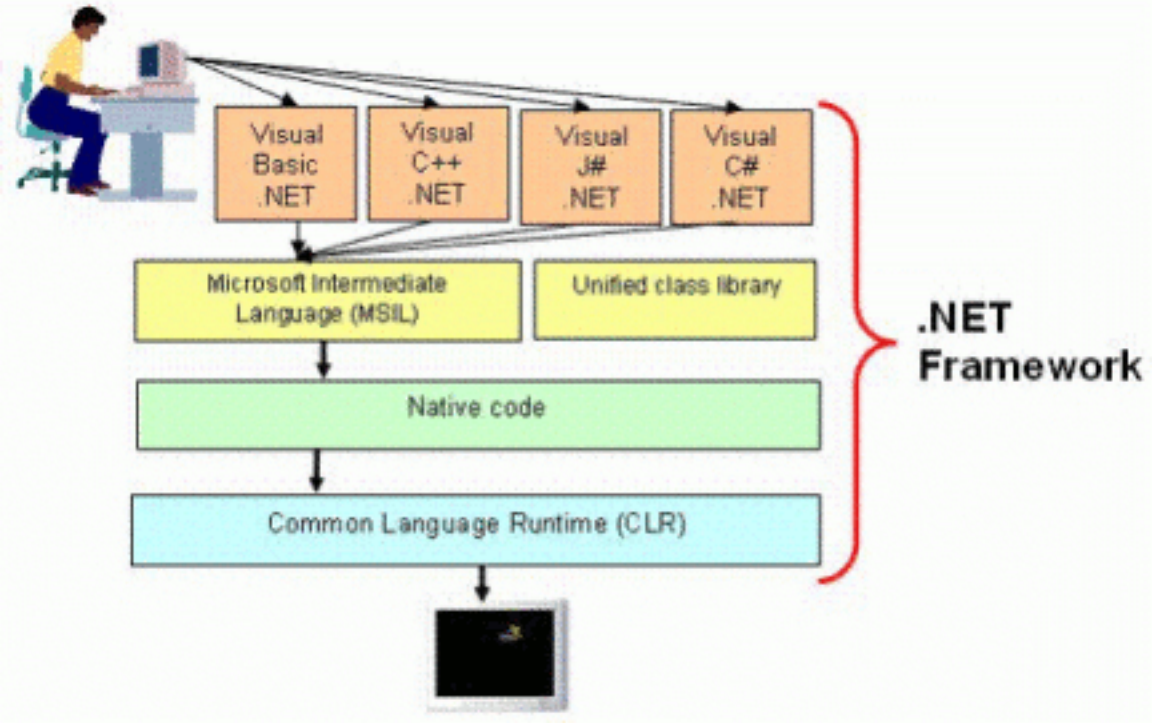
P1086: .NET Framework

To address the confusing maze of computer languages, libraries, tools, and toolkits that were necessary for creating multi-tier applications, Microsoft developed the **.NET** Framework and integrated it into Microsoft Windows as a **component**. It supports building and running multi-tier and Service-Oriented Architectures (**SOAs**), including **Web services** and **client** and **server** applications. It simplifies the process of designing, developing, and testing software, allowing individual developers to focus on core, application-specific code.

Microsoft summarizes the .NET Framework as

- A consistent, language-neutral, **object-oriented programming** environment.
- A code-execution environment that minimizes software deployment and versioning conflicts, guarantees safe execution of code, and eliminates the performance problems of scripted or interpreted environments.
- A consistent development environment.
- A framework composed of two key parts: the **Common Language Runtime** (**CLR**) and the **Unified Class Libraries**.

In the Microsoft .NET development environment, a programmer writes software in any one of several Visual .NET languages. These use a single, unified, object-oriented, hierarchical, and extensible set of class libraries to access the system and common services such as **XML** web services, enterprise services, ADO.NET, and XML. Next, the language source code is compiled into an intermediate **Microsoft Intermediate Language** (**MSIL**), which is later translated into platform-specific native code that uses the CLR.



Guidance

- G1101: Use **Web services** to bridge **Java EE** and **.NET**.
- G1210: For **.NET**, use Debug and Trace from the **`System.Diagnostics` namespace**.

Best Practices

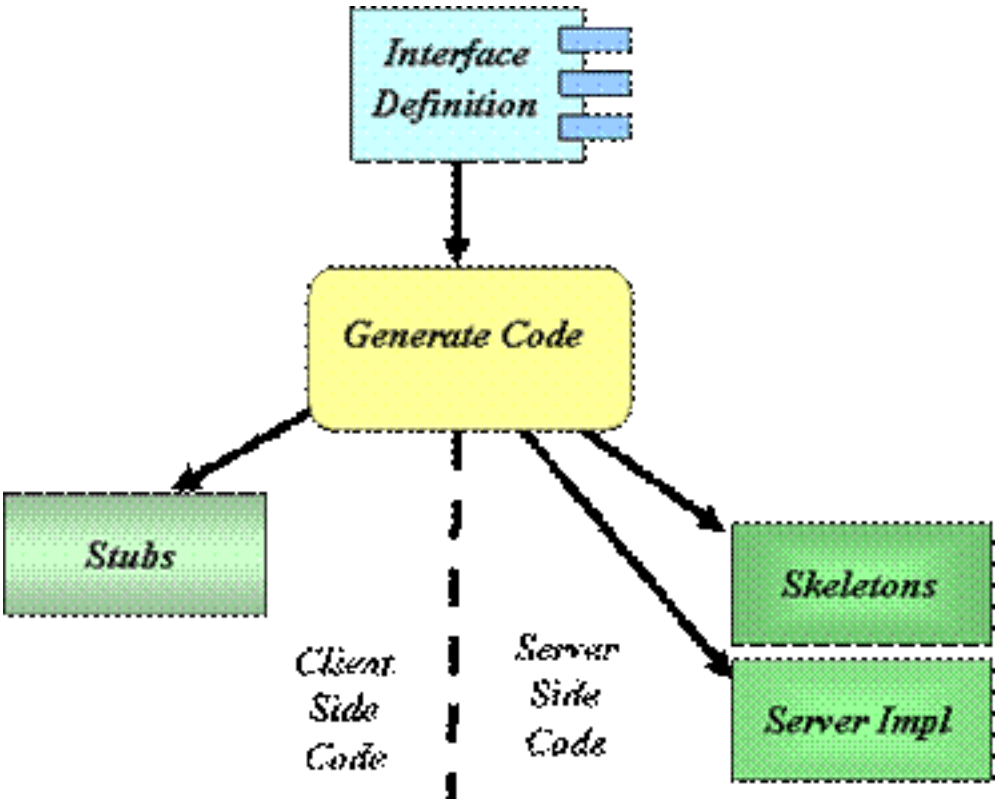
- BP1097: Use the **`System.Text.StringBuilder`** class for repetitive string modifications such as appending, removing, replacing, or inserting characters.
- BP1098: Write all **.NET** code in C#.
- BP1100: Compile all **.NET** code using the .NET **Just-In-Time compiler**.

P1011: CORBA

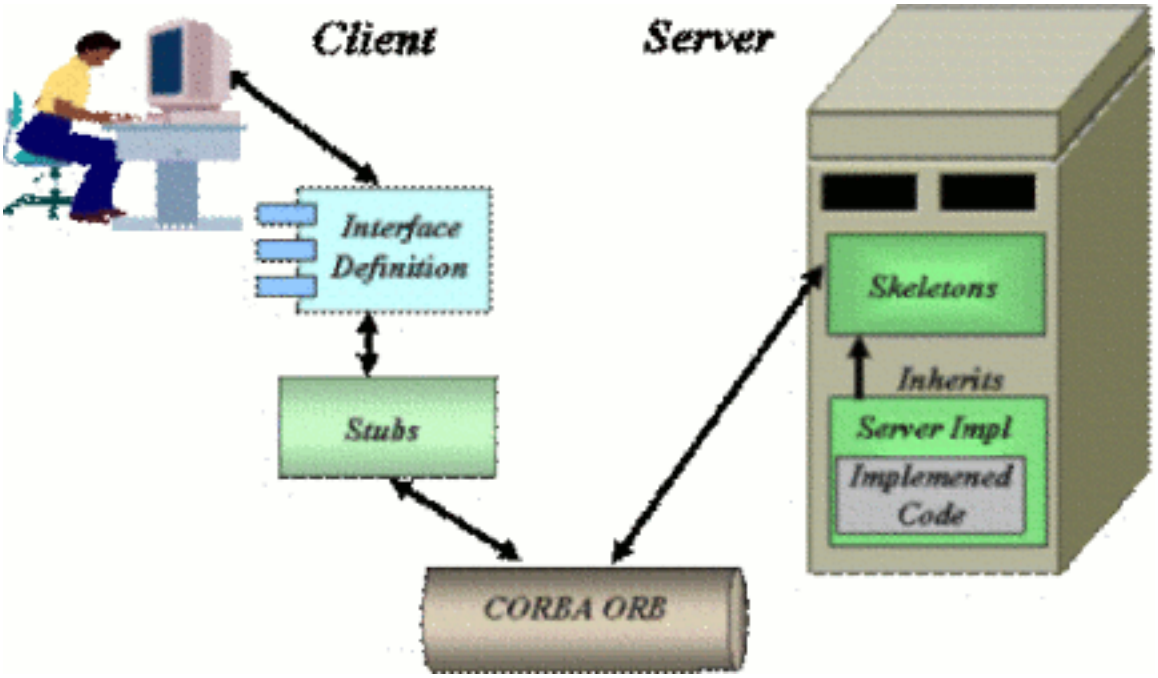
CORBA is the acronym for **Common Object Request Broker Architecture**. It is the **Object Management Group** (**OMG**) open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the Internet InterORB **Protocol** (**IIOP**), a CORBA-based program from any vendor, on almost any computer, operating system, programming language, or network, can interoperate with a CORBA-based program from the same or another vendor on almost any other computer, operating system, programming language, or network.

In general, the code that needs to be created to access an object remotely using CORBA can be implemented using well established and well understood design patterns. Consequently, it is not difficult to write but it is tedious and subject to human error during the writing process because much of it is of a cut-and-paste nature. Therefore, most **Object Request Broker** (**ORB**) vendors have developed code generators that can auto-generate the required infrastructure code given the definition of the interface between a **client** and a **server**. The use of these auto-generators is strongly encouraged.

The following diagram illustrates auto-generation of the infrastructure code from an interface defined using the CORBA **Interface Definition Language** (**IDL**).



This diagram illustrates how the generated code is used within the **CORBA** infrastructure.



Key features

Some of the key features of interest in the **CORBA** specifications are:

- Internet InterORB **Protocol** (IIOP)
- Dynamic Invocation Interface (DII)
- Dynamic Skeleton Interface (DSI)
- Interface Repository (IFR)
- Objects by Value (OBV)
- **CORBA** Component Model (CCM)
- Portable Object Adapter (POA)
- General InterORB **Protocol** (GIOP)
- **Java** to **IDL** mapping

Guidance

- G1118: Localize **CORBA** vendor-specific source code into separate modules.
- G1202: Use the **CORBA Portable Object Adapter** (**POA**) instead of the **Basic Object Adapter** (**BOA**).
- G1119: Isolate user-modifiable configuration parameters from the **CORBA** application source code.
- G1204: Create configuration services to provide distributed user control of the appropriate configuration parameters.
- G1205: Use non-source code persistence to store all user-modifiable **CORBA** service configuration parameters.
- G1121: Do not modify **CORBA** Interface Definition Language (**IDL**) compiler auto-generated stubs and skeletons.

- G1123: Use the Fat Operation Technique in **IDL** operator invocation.
- G1203: Localize frequently used CORBA-specific code in modules that multiple applications can use.

Best Practices

- BP1231: Use **`CORBA::String_var`** in **IDL** to pass string types in C++.
- BP1232: Do not pass or return a zero or null pointer; instead, pass an empty string.
- BP1233: Do not assign **`CORBA::String_var`** type to **`INOUT`** method parameters.
- BP1234: Assign string values to **`OUT`** , **`INOUT`** , or **`RETURN`** parameters using operations to allocate or duplicate values rather than creating and deleting values.
- BP1235: Assign string values to returned-as-attribute values using operations to allocate or duplicate values rather than creating and deleting values.

References

- R1027: OMG - http://www.omg.org/gettingstarted/gettingstartedindex.htm

P1087: Software Communication Architecture

The **Software Communications Architecture** (**SCA**) establishes an implementation-independent framework with baseline requirements for the development of software for an established hardware platform, such as software defined radios. The SCA is an architectural framework that was created to maximize portability, interoperability, and configurability of the software while still allowing the flexibility to address domain specific requirements and restrictions. Constraints on software development imposed by the framework are on the interfaces and the structure of the software and not on the implementation of the functions that are performed.

The framework places an emphasis on areas where reusability is affected and allows implementation unique requirements to determine a specific application of the architecture. SCA specifications incorporate accepted industry standards such as a subset of the **Portable Operating System Interface** (**POSIX**) specification and the **Object Management Group** (**OMG**) **CORBA** specification.

SCA includes a real-time operating system functionality to provide multi-threaded support for all software executing on the system. Software can include SCA applications, devices, and services. The exact functionality supported by the Operating Environment is described by the **Application Environment Profile** (**AEP**) which is a subset of the POSIX specification.

The OMG Domain Special Interest Group for Software Radios (SWRADIO DSIG) and Software Defined Radio Forum (SDRF) are working together towards building an international commercial standard based on the SCA.

The purpose of this perspective is to provide guidance and reference material for Programs providing products and services using SCA in order to increase interoperability and net-centricity.

Guidance

- G1713: Use an Operating Environment (OE) for all SCA applications that includes middleware that, at a minimum, provides the services and capabilities specified by Minimum CORBA Specification version 1.0.

- G1714: Develop SCA application to only use Operating Environment functionality defined by the SCA Application Environment Profile.

Best Practices

- BP1715: Design SCA log services according to the OMG Lightweight Log Service Specification.

- BP1716: Develop applications for SCA-compliant systems using a standard higher order language.

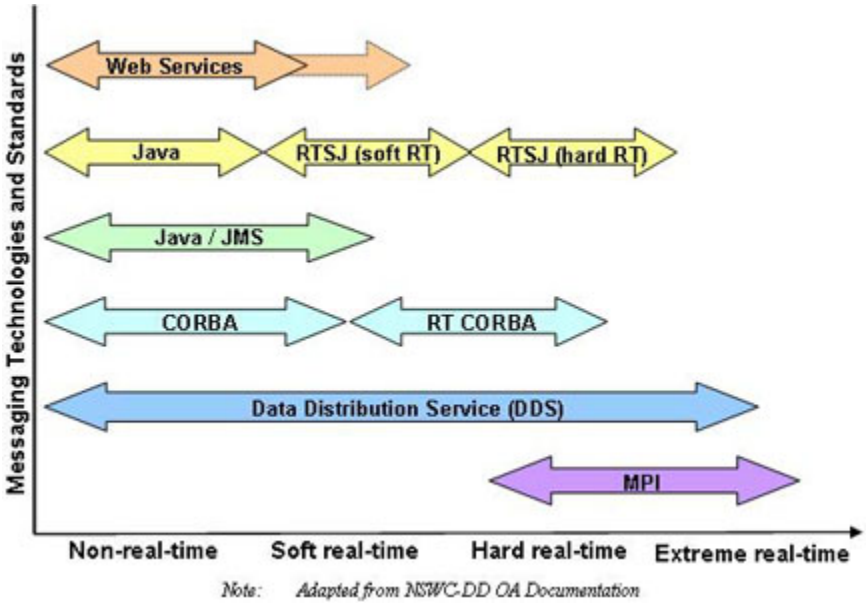
References

- R1108: Software Communications Architecture
- R1109: Minimum CORBA v1.0
- R1110: OMG Lightweight Log Service
- R1111: Software-based Communications DTF
- R1112: Software Communications Architecture (Wikipedia)

P1190: Data Distribution Service (DDS)

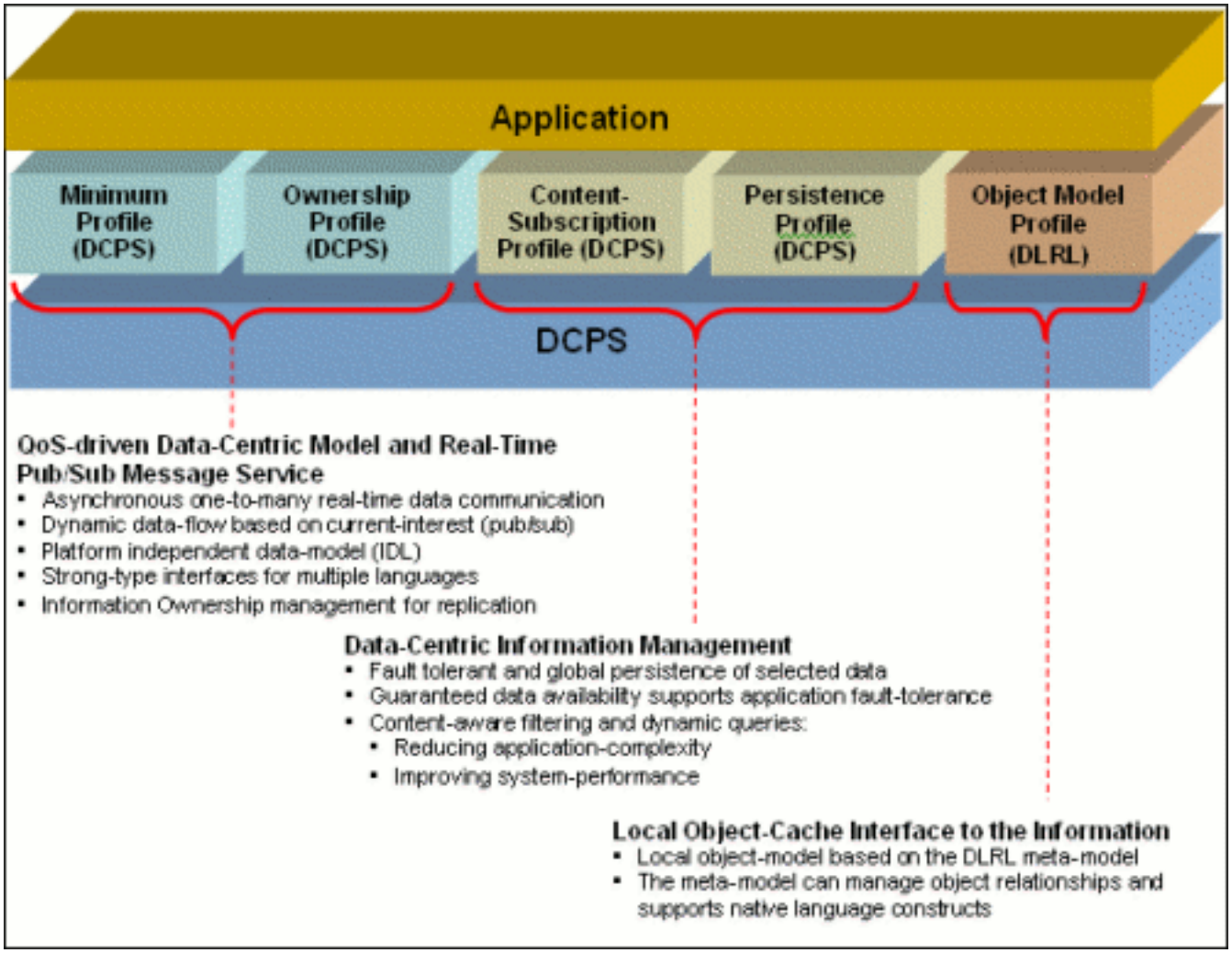
Data Distribution Service for Real-time Systems is an **Object Management Group** (OMG) specification for distributing data messages using the **Publish-Subscribe** design pattern. It defines a common application programming interface (API) that cleanly separates the data distribution functionality from the application functionality. DDS also simplifies the complexity associated with application programming by separating the details of publishing data messages from those for subscribing to data messages using a **Quality of Service** (QoS) approach. The implementation of the interface effectively creates a data distribution service that applications can access.

The use of QoS makes DDS especially appealing as an integration middleware in heterogeneous systems. DDS QoS allows fine-grained tuning of the properties for each information flow including the lowest level data writer and data reader. Therefore, the system can devote its resources to the more critical flows ensuring they are achievable. Also, the use of QoS combined with the inherent real-time nature of the DDS allows DDS solutions to span the complete spectrum from Enterprise (non-real-time) to hard real-time applications as shown in the following figure.



DDS Profiles

The specification divides the complexity of the full data distribution functionality into five profiles (Minimum, Ownership, Content Subscription, Persistence, and Object Model) to help applications meet their individual requirements. The applications can use any or all of the profiles to access the Data Distribution Service.

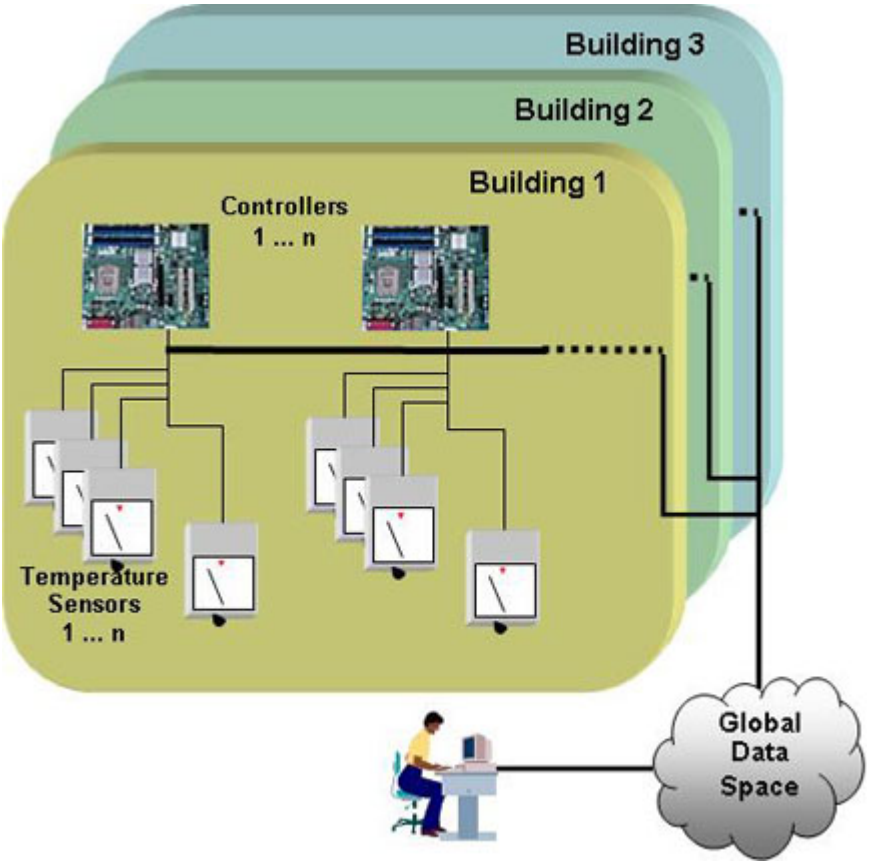


DDS Compliance Profiles

Minimum	This profile contains just the mandatory features of the **DCPS** layer. None of the optional features are included.
Ownership	This profile adds the following: • the optional setting **`EXCLUSIVE`** of the **`OWNERSHIP`** kind • support for the optional **`OWNERSHIP_STRENGTH`** policy • the ability to set a depth > 1 for the **`HISTORY`** QoS policy.
Content-Subscription	This profile adds the optional classes **`ContentFilteredTopic`**, **`QueryCondition`**, and **`MultiTopic`**. This profile also enables subscriptions by content.
Persistence	This profile adds the optional QoS Policy **`DURABILITY_SERVICE`** as well as the optional settings **`TRANSIENT`** and **`PERSISTENT`** of the **`DURABILITY`** QoS Policy kind. This profile enables saving data into either transient memory, or permanent storage so that it can survive the lifecycle of the **`DataWriter`** and system outings.
Object Model	This profile includes the **DLRL** and also includes support for the **`PRESENTATIONaccess_scope`** setting of **`GROUP`**.

Example

The following diagram depicts using a data-oriented approach to solve a typical distributed system problem. The goal in this example is to maintain the temperature in many buildings, using embedded controllers each connected to a number of sensors. Each of these sensors and control processes are connected through a transport mechanism such as Ethernet and use basic protocols such as **TCP-UDP/IP** to provide standardized communication.



To achieve data integrity and fail-over capabilities, multiple controllers and sensors are deployed in each building. Controllers within a building collaborate in the process of collecting data from the various sensors. Applications access and manipulate the data through the use of a global data space.

Data-centric technologies such as databases and Service-Oriented Architecture **Web service**-based applications can interoperate seamlessly with the embedded sensors. These technologies provide a standards-based way for external applications to get, process and manipulate real-time sensor data with out having to know the specifics of the real-time data infrastructure. Furthermore, decoupling the data from the technology that manipulates the data contributes to developing a truly data-centric application. In this example, the external access and monitoring applications can simply receive real-time updates from any sensor as well as issue commands to the various controllers via DDS, **SQL**, etc., to maintain suitable temperatures.

Data Model

For simplicity, this example will focus on the data the sensors send to their controller and how they can be distributed throughout the entire system. The first step in a data-centric approach is to describe the data format carefully in a standards-based way, either IDL or XML, and give it a **Topic** name. Topics are the element of the DDS middleware publish-subscribe standard which identify the data objects and provide the basic connection between **publishers** and **subscribers**. Subscribers (the Controllers in this example) register Topics with the middleware that they wish to receive. Publishers (the individual sensors in this example) register Topics with the middleware that they will send. If the Topics do not match, effective communication does not take place.

Topics enable one to find specific information sources when architecting a loosely coupled system; that is, one which does not know a priori how many sensors or controllers there are going to be or where they all are. The Controller can simply subscribe to **`TempSensor`**, the Topic's name, and receive all the sensor updates for that building. Similarly, a sensor does not need to know if it is sending its data to one or multiple Controllers or even an external data store.

Specification of the Topic's name is a key element in a **data-centric** approach to creating open real-time systems. One could name each sensor's Topic based on its unique location in the building, **`Floor12Room3Sensor14`** for example, but the Controller would then need to be configured every time a sensor is added or removed from the system. Topics (name and type) define the standard interface for the distributed system; chose them appropriately.

Data Type

Specification of the Topic's data type is equally important as the Topic's name. DDS specifies the use of a subset of the **Interface Definition Language** (IDL) for specifying a Topic's data type.

Note: *IDL readily maps to XML and SQL semantics.*

```
struct SensorData
{
  long    id; //@key
  float   temp;
};
```

In the definition of the Topic's type, chose one or more data elements to be a ***Key***. Keys provide scalability and the communication infrastructure can use the key to sort and order data from many sensors. In this example, without keys, one would need to create individual Topics for each sensor. Topic names for these topics might be **`Sensor_1`**, **`Sensor_2`**, and so on. Therefore, even though each Topic is comprised of the same data type, there would still be multiple Topics. With keys, there is only one topic, **`TempSensor`**, used to report temperatures.

New sensors can be added without creating a new Topic. The publishing application would just need to set a new id when it was ready to publish. An application can also have a situation where there are multiple publishers of the same Topic with the same key defined. This enables the application to provide redundancy. Per this example, two sensors in the same room using the same Key value will measure the same piece of information. Managing the redundancy, should one or both sensors report to the controller, is accomplished though Quality-of-Service (QoS).

Domains and Partitions

A **Domain** is the basic DDS construct used to bind individual **publications** and **subscriptions** together for communication. A distributed application can elect to use single or multiple DDS Domains for its data-centric communications. A Partition is a way to separate Topics logically within a DDS Domain.

In the context of the example, Partitions can group sensors on different floors. For example, to divide the building into different zones where each zone is controlled by a dedicated Controller, the Sensor and Controller could set the Partition to **`Floor 1`** and **`Floor 1-6`**, respectively. The Controller will receive data from all Sensors on Floors 1 through 6. Using Partitions makes it easy to group which Sensors are ***hooked*** to a Controller and a Controller can take over a different zone by changing or adding to its Partition list.

In the example, different buildings map to different DDS Domains. Domains isolate communication, promote scalability and segregate different classifications of data.

Quality of Service

The following briefly details how one might leverage a few of the DDS QoS Policies for this example.

Ownership

The Ownership QoS specifies whether or not multiple publishers can update the same data object and is how to achieve fault-tolerance using DDS.

Returning to the example, having multiple sensors in the same room and only wanting to get data from the primary (as long as it is functioning), then the Ownership QoS policy is set to Exclusive, stating that only one sensor can update that keyed value. Setting the Ownership QoS value to Shared indicates that there can be multiple sensors in the same room all reporting the same piece of keyed data. In this case the Controller would get all updates from all sensors and treat the values as the same measurement.

Durability

The Durability QoS specifies whether past samples of data will be available to newly joining subscribers.

Considering the example, if a Controller were to reboot, rather than require all sensors to resend their data, or require the data to be sent at a periodic rate in case the systems reboots, one simply gets the latest published value for every attached sensor. This effectively decouples the system in time and provides a high degree of data integrity.

History

History specifies how many data samples are stored for later delivery.

In the case of the example, a rebooted controller may want the last 5 samples from its sensors, so that it can make sure that readings are consistent.

Reliability

The Reliability QoS may be set on a per Topic basis and informs the middleware that the Subscription should receive all data (no missed samples) from a Publication even over non-reliable transports. Generally for periodic publications Reliability doesn't need to be set, since it can just get the updated value one sample period later. Although periodic sensor data doesn't need to be delivered reliably, synchronization commands between Controllers in this example could be.

Summary

This simply stated example is surprisingly complex, containing many elements of real-time messaging, data integrity and failover capabilities, integration with databases, web services, as well as scalability and modularity concerns while remaining data-centric.

Detailed Perspectives

- Decoupling Using DDS and Publish-Subscribe
- DDS Quality of Service (QoS)
- DDS Data-Centric Publish-Subscribe (DCPS)
- DDS Data Local Reconstruction Layer (DLRL)

P1191: Decoupling Using DDS and Publish-Subscribe

A fundamental tenet of data-centricity and DDS is the decoupling between information providers and consumers. The decoupling is conceptually anonymous in that the producers do not need to know who the consumers are, and similarly the consumers do not need to know who the producers are. They are in fact each communicating independently using the DDS **Domain** (i.e., **Global Data Space**). Persistence services in the Global Data Space allow data written by an application to be available to late joining applications, even if the original application is no longer present.
While communications can precede anonymously, DDS does offer the means for an application to detect its communication partner. A ***Writer*** can see who the matched Readers are, and similarly a ***Reader*** can identify the matched Writers. If so requested, the application is given notification of new matches and can even "veto" specific Readers or Writers.

Decoupling and anonymity is accomplished using the publish-subscribe paradigm. Applications that want to provide information indicate their intent to publish by creating a **DataWriter** and specifying the offered **Quality of Service** (QoS) and a **Listener**. Applications that want to access information indicate their intent to subscribe by creating a **DataReader** and specifying the requested QoS and a Listener.

Publishers are matched with **subscribers** by DDS using the **Topic** and the QoS, and DDS automatically sets up the needed communication paths and resources such that information (data updates) can flow directly with the highest possible performance. **Listeners** are used to indicate to the application that certain events of interest have taken place, such as the arrival of new information for **`DataReaders`**, violations in the QoS contracts, matching of new Publishers/Subscribers or other middleware-observed events.

QoS contracts provide the means for applications/components to remain modular and independent from each other while at the same time having some control over how the information is provided or delivered. For example, a reading application may have some minimum requirements regarding reliability, ordering, coherence, or frequencies of updates, and a writing application may have some resource limits with regards to how much history it can maintain or how many readers it can handle. The QoS contract can specify these requirements and DDS checks and monitors them. In addition QoS can configure resources, message priorities, history, etc. The ability to fine-tune separately the behavior of each **`DataWriter`** and **`DataReader`** is one of the reasons why DDS can span the range from real-time to near-time to enterprise systems.

Guidance

- G1807: Check the return values of **Data Distribution Service** (DDS) functions.
- G1802: Catch all **Data Distribution Service** (DDS) events.
- G1809: Handle all **Data Distribution Service** (DDS) events using one of the **subscriber access APIs**.
- G1810: Use **data models** to document the data contained within the **Data Distribution Service** (DDS) **Data-Centric Publish Subscribe** (DCPS).

Best Practices

- BP1811: Isolate all use of vendor specific extensions to the **Data Distribution Service** (DDS).
- BP1825: Use the **`ignore_participant`** operation on the **DomainParticipant** to deny access to another DomainParticipant trying to join a **Data Distribution Service** (DDS) **Domain**.
- BP1827: Use the **`ignore_publication`** and **`ignore_subscription`** on the **DomainParticipant** to deny access to a **Data Distribution Service** (DDS) **Topic** by a specific **DataWriter** or **DataReader**.
- BP1830: Use the **Data Distribution Service** (DDS) Content Profile to tailor subscription message data.
- BP1831: Use the **Data Distribution Service** (DDS) Persistence Profile to ensure durable data delivery.

References

- R1202: **OMG** Data Distribution Service for Real-time Systems Version 1.2

P1192: DDS Quality of Service

Quality of Service (QoS) is a general concept that specifies the behavior of a service. Programming service behavior by means of QoS settings offers the advantage that the application developer only indicates what is wanted rather than how to achieve the specific QoS. Generally speaking, QoS is comprised of several QoS policies. Each QoS policy is then an independent description that associates a name with a value. Describing QoS by means of a list of independent QoS policies gives rise to more flexibility.

> ***Note:*** *As Service-Oriented Architecture (SOA) systems evolve and become richer in the number of publishers and subscribers supported with time, the use of well defined and specific QoS parameters becomes essential in managing the complexity of the system and the loosely coupled nature of the services.*

Data-centric communication using DDS provides the ability to specify various parameters like the rate of publication, rate of subscription, how long the data is valid, and many others. These QoS parameters allow system designers to construct a distributed application based on the requirements for, and availability of, each specific piece of data. A data-centric environment allows a communication mechanism that is custom tailored to the distributed application's specific requirements yet remains a loosely coupled design and architecture.

The ability to set QoS on a per-entity basis is a significant capability provided by DDS. Being able to specify different QoS parameters for each **Topic**, **Publisher** or **Subscriber** gives developers many options when designing their systems. Through the combination of these parameters, a system architect can construct a distributed application to address an entire range of requirements, from simple communication patterns to complex data interactions.

Guidance

- G1771: Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of a **publisher**.
- G1801: Explicitly define a **Topic Quality of Service** (QoS) for each **Data Distribution Service** (DDS) Topic within a DDS **Domain**.
- G1803: Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe real-time messaging criteria for **Publishers**.
- G1804: Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe **DataWriter**.
- G1805: Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of the **Subscriber**.
- G1806: Explicitly define the Request-Offered **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of the **DataReader**.
- G1808: Handle all **Data Distribution Service** (DDS) **Quality of Service** (QoS) contract violations using one of the **Subscriber access APIs**.

Best Practices

- BP1812: Use the **`RELIABILITY`** **Quality of Service** (QoS) kind **`BEST_EFFORT`** for **Data Distribution Service** (DDS) **Topics** that are written frequently where missing an update is not important because new updates occur soon thereafter.
- BP1813: Use the **`RELIABILITY`** **Quality of Service** (QoS) kind **`RELIABLE`** for **Data Distribution Service** (DDS) **Topics** written sporadically or where it is important that the current data in the Topic is received reliably.

- BP1814: Use the **DEADLINE** **Quality of Service** (QoS) to for **Data Distribution Service** (DDS) **DataWriters** for which data is published at a constant rate.

- BP1815: Use the **DEADLINE** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **DataReaders** that expect data to be sent to them at a constant rate.

- BP1816: Use the **LIVELINESS** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **Topics** where data is not sent sporadically; that is, it is sent with no fixed period.

- BP1817: Use the **MANUAL_BY_TOPIC** setting of the **LIVELINESS** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **Topics** where the presence and health of the **DataWriter** is critical to the proper operation of the system.

- BP1818: Use the **HISTORY** **Quality of Service** (QoS) kind **KEEP_LAST** for **Data Distribution Service** (DDS) **Topics** that represent system state, in that new data-values replace the old values for each Keyed data-object.

- BP1819: Use the **HISTORY** **Quality of Service** (QoS) kind **KEEP_ALL** for **Data Distribution Service** (DDS) **Topics** that represent events or commands where all values written should be delivered to the readers (i.e., new values do not replace old values).

- BP1820: Use **TIME_BASED_FILTER** **Quality of Service** (QoS) to protect **DataReaders**that cannot handle all the traffic that could be written by the writers on that **Data Distribution Service** (DDS)**Topic** and just need periodic updates on the most current data-values.

- BP1821: Use the **Data Distribution Service** (DDS) **LIFESPAN** **Quality of Service** (QoS) to indicate that data is only valid for a finite time period and stale data is discarded after a certain expiration time elapses.

- BP1822: Use the **PARTITION** **Quality of Service** (QoS) to limit the scope of the data written/read on a **Data Distribution Service** (DDS) **Topic** to only the writer/readers that have a common partition.

- BP1823: Use the **Data Distribution Service** (DDS) **RESOURCES_LIMITS** **Quality of Service** (QoS) in platforms with limited memory or in real-time systems to properly configure the resources that will be utilized and avoid exhaustion of system resources at run-time.

- BP1824: Use the **USER_DATA** **Quality of Service** (QoS) to communicate metadata on the **DomainParticipant** that may be used to authenticate the application trying to join the Data **Distribution Service** (DDS) **Domain**.

- BP1826: Use the **USER_DATA** **Quality of Service** (QoS) on the **DataWriters** and **DataReaders** to communicate metadata that may provide application-specific information of the entity writing/reading data in a **Data Distribution Service** (DDS) **Domain**.

- BP1828: Use the **Data Distribution Service** (DDS) **OWNERSHIP** **Quality of Service** (QoS) kind set to **SHARED** when each unique data-object within a DDS **Topic** to which multiple **DataWriters** can write.

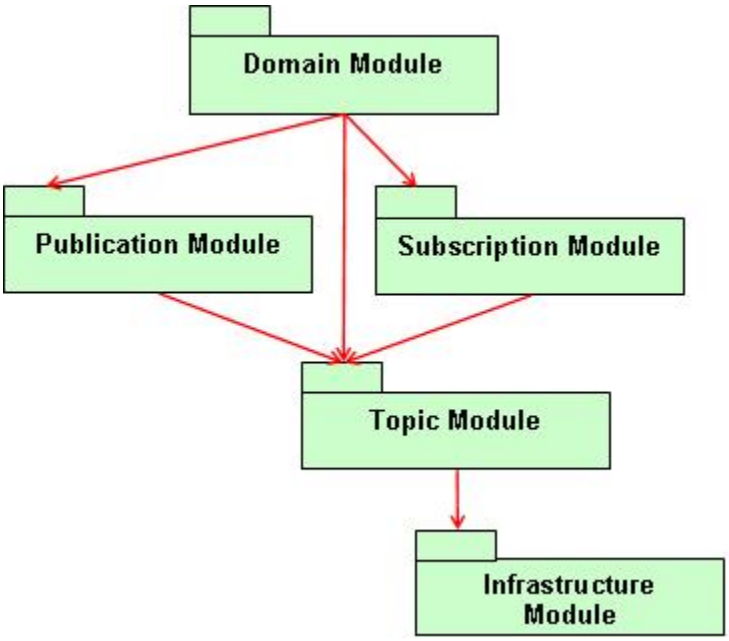
- BP1829: Use the **Data Distribution Service** (DDS) **OWNERSHIP** **Quality of Service** (QoS) kind set to **EXCLUSIVE** when multiple **DataWriters** cannot write each unique data-object within a DDS **Topic** simultaneously.

P1193: DDS Data-Centric Publish-Subscribe (DCPS)

The **Data-Centric Publish-Subscribe** (DCPS) interface is targeted toward the efficient delivery of the proper information to the proper recipients. It provides the application with a **data-centric** information model and is responsible for controlling the lower level layer of the DDS infrastructure targeted toward the efficient and reliable delivery of the information to its intended recipients. The DCPS architecture is comprised of five modules. The modules build upon each other in a hierarchical inheritance structure. The following table captures the purpose of each of the five modules.

Infrastructure Model	Defines the abstract classes and the interfaces that are refined by the other modules; also provides support for the two interaction styles (notification- and wait- based) within the middleware
Domain Module	Contains the **DomainParticipant** class that acts as an entrypoint of the Service and acts as a factory for many of the classes; the **`DomainParticipant`** also acts as a container for the other objects that make up the Service
Topic-Definition Module	Contains the **Topic**, **`ContentFilteredTopic`**, and **`MultiTopic`** classes, the **`TopicListener`** interface, and more generally, all that is needed by the application to define Topic objects and attach **QoS** policies to them
Publication Module	Contains the **Publisher** and **DataWriter** classes as well as the **`PublisherListener`** and **`DataWriterListener`** interfaces, and more generally, all that is needed on the publication side
Subscription Module	Contains the **Subscriber**, **DataReader**, **`ReadCondition`**,and **`QueryCondition`** classes, as well as the **`SubscriberListener`** and **`DataReaderListener`** interfaces, and more generally, all that is needed on the subscription side

The following is a UML Class diagram that represents the five modules and how they relate to each other.



Detailed Perspectives

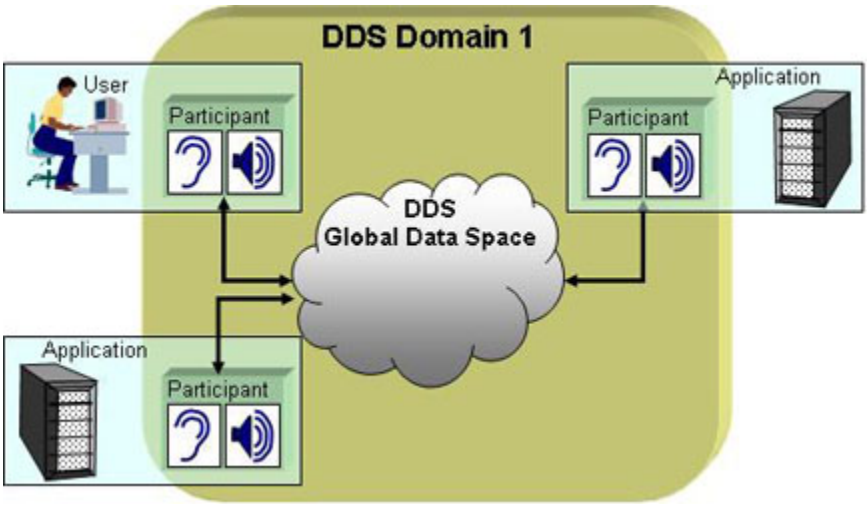
- DDS Domains - Global Data Spaces

- Reading/Writing Objects within a DDS Domain
- Messaging within a DDS Domain

P1194: DDS Domains - Global Data Spaces

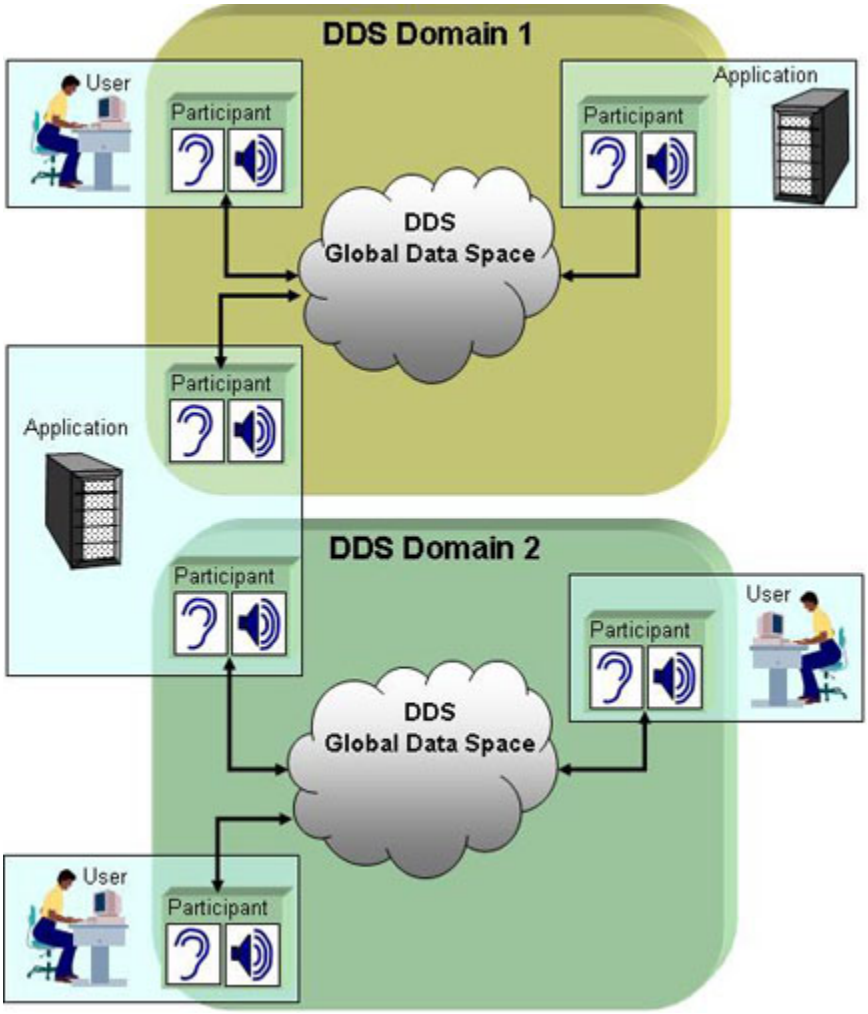
DDS allows application developers to create a collection of virtual shared **Global Data Spaces** where separate application processes can share data anonymously. Processes can access (read and/or write) data in the Global Data Space as well as exchange messages on the associated DDS **Domain**.

A DDS Global Data Space (called a DDS Domain) is identified by a domainId that represents an isolated Data Space. The Data Space exchanges no information or messages with other domains. The operating system maintains isolation between DDS Domains by using different port numbers. Each computer process (running on behalf of some user or application) must attach to the desired DDS Domain by creating a DDS **DomainParticipant**. Each **`DomainParticipant`** is owned by the creating process and is only accessible to it.



> ***Note:*** *The centralized image of a Global Data Space is just a convenient metaphor. In reality the DDS specification mandates that there should be no centralized implementation of the global data and data updates must flow directly from the writer to the readers.*

A distributed system may employ multiple DDS Domains (i.e., Global Data Spaces), each identified by a different **`domainId`**. A single application process may access multiple Global Data Spaces by creating multiple **`DomainParticipants`**, each associated with one of the Global Data Spaces.



Guidance

- G1770: Explicitly define the **Data Distribution Service** (DDS) **Domains** for the system.
- G1772: Assign a unique identifier for each **Data-Distribution Service** (DDS) **Domain** within the system.

P1195: Reading/Writing Objects within a DDS Domain

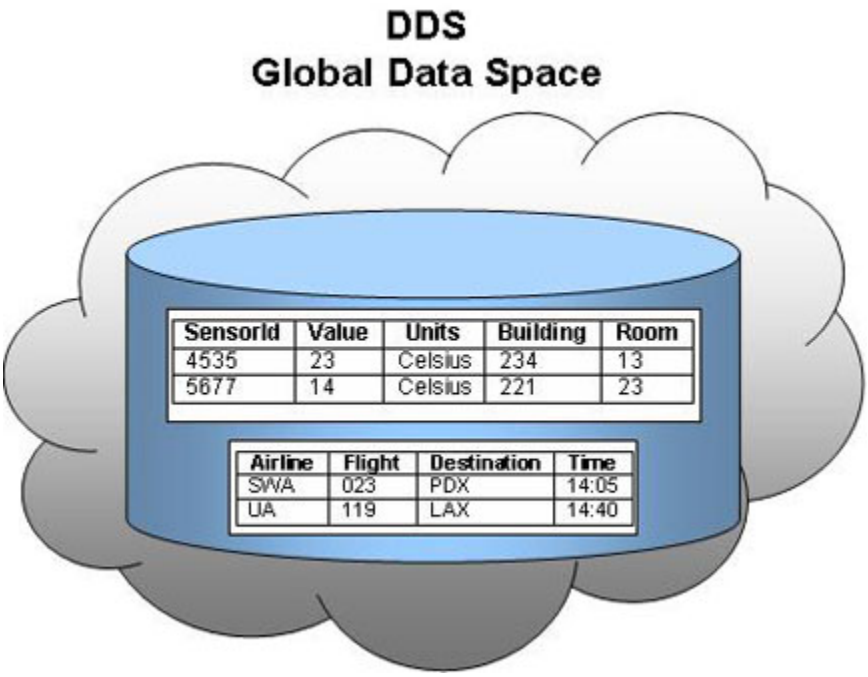
Address the Data Objects in the **Global Data Space** by means of a **Topic** (an application-chosen string that encodes a homogeneous collection of objects) and a ***Key*** (a set of fields inside the data object that uniquely identifies the object within the collection). A DDS Topic is an application-chosen string (such as **`Temperature`**) that has an associated schema or format representing the type of the data objects (for example the sensor ID, the value, the units, the location of the sensor, the time-stamp, etc.). The DDS Key is specific to each DDS Topic and uniquely identifies each Data Object within the Topic.

Pictorially one could think of each Topic in the Global Data Space representing a table of related data objects where each row represents the value of an individual data object the columns define the schema (data type of the object), and the key is the column(s) that defines the identity of each object. The table below depicts this concept for the hypothetical **`Temperature`** Topic.

SensorId (Key)	***Value*** : float	***Units*** : string	***Location*** : string	***Timestamp***
4535	23	Celsius	Building 234, Room 13	Tue Oct 31 15:47:42 PST 2006
5677	12	Celsius	Building 121, Furnace 23	Tue Oct 31 15:44:42 PST 2006

Another example is an Airport Information application that defines the Topic DepartingFlights with a schema consisting of fields containing the following information: Airline, flight number, destination airport, departure terminal, gate, scheduled departure time, expected departure time, and status. In this case the combination of fields Airline and Flight Number provides the Key that uniquely identifies each flight. Updates to the global data space will provide new estimated departure times, departing dates, etc. A display application may read this topic to show all the flights departing in the next three hours.

Airline (Key)	***Flight Number*** (Key)	***Destination***	***Departure Terminal***	***Departure Gate***	***Scheduled Departure***	***Expected Departure***	***Status***
SWA	023	PDX	A	12	10:30	14:05	Departed
UA	119	LAX	A	06	14:27	14:40	Boarding
AS	543	ANC	A	03	14:10	14:20	Boarding
KLM	006	AMS	A	14	14:35	14:35	Boarding
SQ	012	SIN	B	03	15:00	15:20	Go to Gate
JL	001	NRT	B	33	15:45	15:45	Go to Gate
LOT	007	WAW	B	02	16:30	16:30	Wait



Guidance

- G1810: Use **data models** to document the data contained within the **Data Distribution Service** (DDS) **Data-Centric Publish Subscribe** (DCPS).
- G1141: Use standard **data models** developed by **Communities of Interest** (**COI**) as the basis of program or project data models.
- G1146: Include information in the **data model** necessary to generate a **data dictionary**.
- G1147: Use **domain analysis** to define the constraints on input data validation.
- G1148: **Normalize** data models.

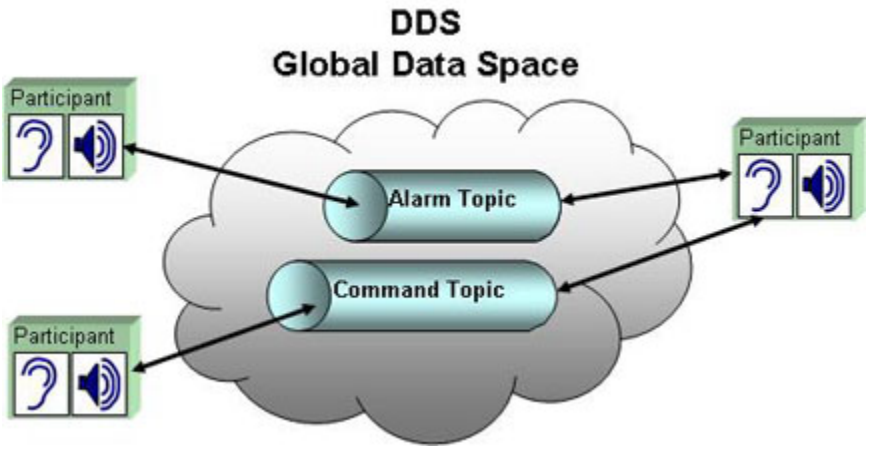
Best Practices

- BP1145: Use vendor-neutral **conceptual/logical models**.
- BP1254: For **command-and-control** systems, use the names defined in the **C2IEDM** for data exposed to the outside communities.
- BP1397: For new systems, identify and develop use cases or reuse existing use cases as appropriate as early in the data engineering process as possible to support **data model** development.
- BP1404: For DoD Programs requiring a **data model**, the **NATO** Generic Hub v.5 model (**LC2IEDM**) is an example of a successful **COI**-developed model.

P1196: Messaging within a DDS Domain

A DDS **Topic** acts like a virtual message-queue or pipe when DDS is used for messaging. Writers send messages though the Topic and readers access messages using the same Topic.
Topics for DDS messages are bound to an application-defined schema in advance; for example, an ***Alarm*** message where the schema consists of source identifier, the kind of alarm, the location, a time-stamp, and the urgency level. **DomainParticipants** can publish and subscribe messages by specifying the Topic and the associated contents.

The Topics used for messaging also live within a DDS **Domain** (i.e., **Global Data Space**) identified by a unique **`DomainId`**. Similar to the data-object paradigm, the middleware keeps the messaging Topics separated within different DDS Domains by using different port numbers.



> ***Note:*** *The centralized image of a pipe is only a convenient concept. In reality, the DDS specification mandates that there should be no centralized implementation of a pipe in DDS. Messages must flow directly from the sender to the receivers.*

The distinction between reading/writing data and receiving/sending messages is essentially a property of the Topic. Some Topics represent data (if the identify certain fields as Keys) and others represent messages (if they do not contain specify Keys). In addition, use different **Quality of Service** settings to attain the proper semantics. For example, associate Topics representing data with a **`HISTORY`** QoS setting of **`KEEP_LAST`** whereas Messages typically use a **`HISTORY`** setting of **`KEEP_ALL`**.

> ***Note:*** *For more details on this subject please refer to the introductory material on DDS available at the OMG DDS Portal.*

Guidance

- G1796: Explicitly define all the **Data Distribution Service** (DDS) **Domain Topics**.
- G1798: Explicitly define all the **Data Distribution Service** (DDS) **Domain data types**.
- G1799: Explicitly associate data types to the **Data Distribution Service** (DDS) **Topics** within a DDS **Domain**
- G1800: Explicitly identify Keys within the **Data Distribution Service** (DDS) **data type** that uniquely identify an instance of a data object.

- G1801: Explicitly define a **Topic Quality of Service** (QoS) for each **Data Distribution Service** (DDS) Topic within a DDS **Domain**.

P1197: DDS Data Local Reconstruction Layer (DLRL)

The **Data Local Reconstruction Layer** (DLRL) is an optional part of the Data-Distribution Service (DDS) specification that provides a local ***object-cache*** abstraction built upon the core **DCPS** layer and requires application objects to comply with the DLRL object metamodel which includes collections and relationships.

> ***Note:*** *The DLRL, a recent addition to the DDS specification, is particularly rich; implementations using this upper-level profile of the specification are emerging.*

Application developers use the DLRL to do the following:

- Describe classes of objects with the associated methods, data fields and relations
- Attach data fields to Data-Centric Publish-Subscribe (DCPS) entities
- Use native language constructs to manipulate objects (i.e., create, read, update, delete) using native language constructs to seamlessly interact with the DCPS layer
- Manage objects and pointers to objects in a cache

Best Practices

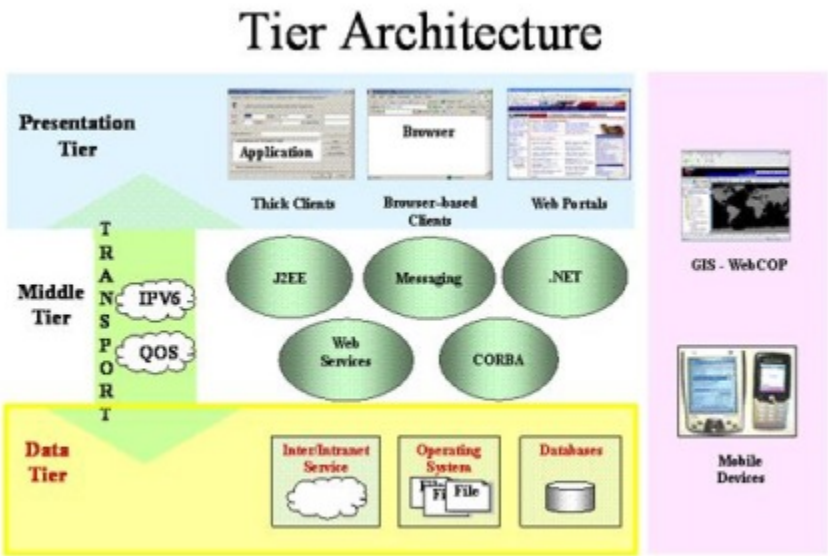
- BP1832: Handle all **Data Distribution Service** (DDS) **Data Local Reconstruction Layer** (DLRL) Exceptions.
- BP1833: Use the **Data Distribution Service** (DDS) Object Model Profile for accessing message data as objects.

References

- R1202: **OMG** Data Distribution Service for Real-time Systems Version 1.2
- R1203: **OMG** Data Distribution Portal (http://portals.omg.org/dds)

P1015: Data Tier

The data tier is responsible for storing data. It does not (should not) contain any business logic (which belongs in the middle tier) and handles only that processing required to access data and maintain its integrity.



Current guidance is in the following perspectives:

- Decouple from Applications
- Database Implementations
- Database Development
- RDBMS Internals

Most modern multi-tiered systems need to collect, store, retrieve and manage persistent data. This data persistence is the responsibility of the data tier. In essence, the data tier functionality is accomplished with modern **COTS** Database Management Systems (**DBMSs**) such as MySQL, Oracle, **SQL** Server, or Sybase Adaptive Server Enterprise (ASE).

P1017: Decouple from Applications

To promote database independence, access the database only through **open-standard** interfaces. The goal is to swap out data sources and/or connect to multiple data sources without affecting the application or increasing software maintenance costs. Data-level adapters allow applications to access data through database calls that are native to the requesting application. At this point, the **business logic** can be shared with other data sources. This positions the application to move business logic from the database to the middle tier to support database independence.

Guidance

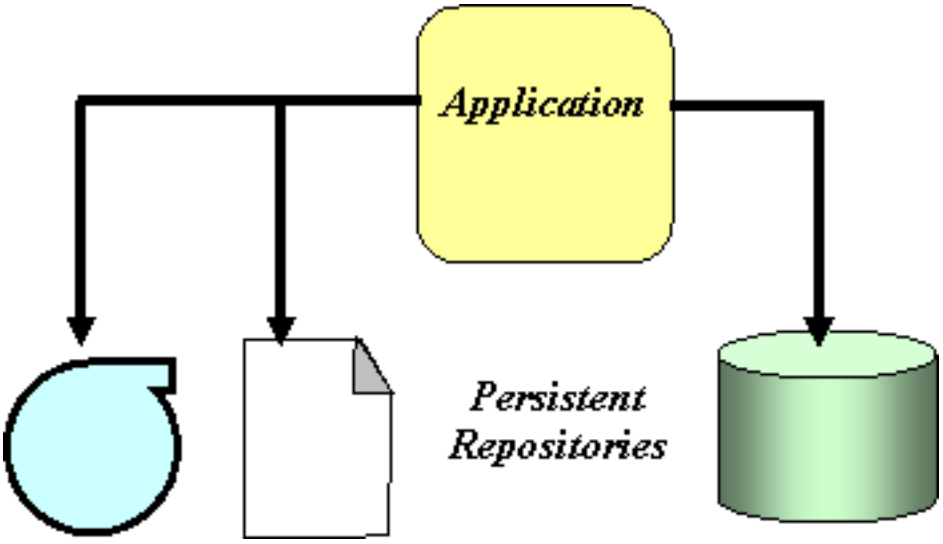
- G1014: Access databases through **open standard** interfaces.

P1014: Database Implementations

The data tier is simply a repository for persistent data. There are many ways that data can be persisted:

- **OS File Systems**
- **Hierarchical Databases**
- **Object-oriented Databases**
- **Niche Databases**
- **Native XML Databases**
- **Relational Databases**

Commercial off-the-shelf (**COTS**) database management systems (**DBMS**) are mature technical products, the capabilities of which are being continually expanded to adapt to and accommodate new technologies.



Guidance

- G1132: Implement the data tier using readily available **COTS RDBMS** products that implement the **SQL** standard and provide a rich set of generic capabilities such as row-level locking, **stored procedures**, **triggers**, and a high-level language **API** interface.

References

- R1062: Won Kim. Introduction to Object-Oriented Databases. Computer Systems. MIT Press, Cambridge, MA, 1990.
- R1069: Native XML database vendors: http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native

P1013: Database Development

The end products of **data modeling** can be **XML** schemas or **RDBMS** schema definitions. See the Data Modeling perspective. The following guidance applies to the data modeling in support of the data tier.

Guidance

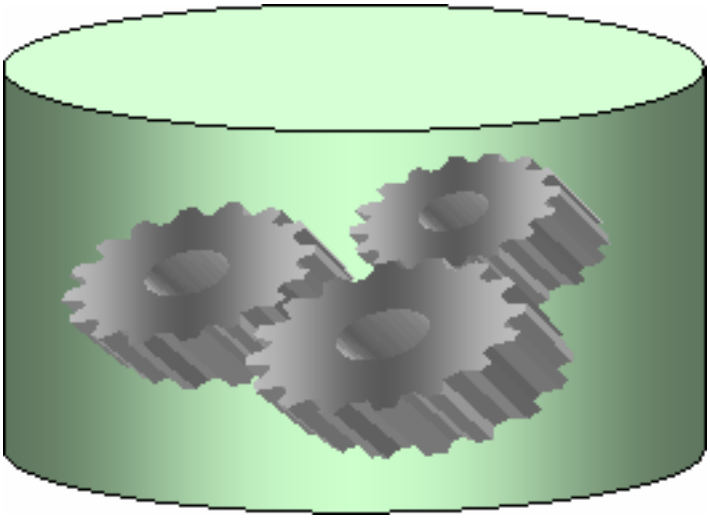
- G1144: Develop two-level database models: one level captures the **conceptual** or logical aspects, and the other level captures the **physical** aspects.
- G1147: Use **domain analysis** to define the constraints on input data validation.
- G1148: **Normalize** data models.
- G1141: Use standard **data models** developed by **Communities of Interest** (**COI**) as the basis of program or project data models.
- G1151: Define declarative **foreign keys** for all relationships between tables to enforce **referential integrity**.

Best Practices

- BP1256: Use surrogate keys as the **primary key**.
- BP1143: Use a **database modeling** tool that supports a two-level model (**Conceptual**/Logical and **Physical**) and **ISO-11179** data exchange standards.
- BP1254: For **command-and-control** systems, use the names defined in the **C2IEDM** for data exposed to the outside communities.

P1063: RDBMS Internals

An **RDBMS** is a collection of data items organized as a set of formally-described tables. This permits accessing and reassembling data in many different ways without having to reorganize the database tables. It is important to ensure data quality and to access data quickly, using simple, easily understood dynamic queries. Towards these ends, an **RDBMS** offers such services as **triggers**, **stored procedures**, indices, constraints, **referential integrity**, efficient storage, and **high availability** features.



Guidance

- G1146: Include information in the **data model** necessary to generate a **data dictionary**.
- G1153: Support n-tier architectures for efficient and accurate maintenance operations.
- G1155: Use **triggers** to enforce **referential** or **data integrity**, not to perform complex **business logic**.
- G1151: Define declarative **foreign keys** for all relationships between tables to enforce **referential integrity**.
- G1154: Use **stored procedures** for operations that are focused on the insertion and maintenance of data.

Best Practices

- BP1248: Follow a naming convention.
- BP1249: Do not use generic names for database objects such as databases, schema, users, tables, views, or indices.
- BP1250: Use case-insensitive names for database objects such as databases, schema, users, tables, views, and indices.
- BP1251: Separate words with underscores.
- BP1252: Do not use names with more than 30 characters.
- BP1253: Do not use the **SQL:1999** or SQL:2003 reserved words as names for database objects such as databases, schema, users, tables, views, or indices.
- BP1256: Use surrogate keys as the **primary key**.

- BP1257: Place a **unique key constraint** on the **natural key** fields.
- BP1260: Define a **primary key** for all tables.
- BP1261: Monitor and tune indexes according to the response time during normal operations in the production environment.
- BP1262: In the case of Oracle, define indexes against the **foreign keys** (**FK**) columns to avoid contention and locking issues.
- BP1263: Gather storage requirements in the planning phase, and then allocate twice the estimated storage space.
- BP1264: For **high availability**, use hardware solutions when geographic proximity permits.
- BP1254: For **command-and-control** systems, use the names defined in the **C2IEDM** for data exposed to the outside communities.
- BP1258: Explicitly define the encoding style of all data transferred via **XML**.
- BP1255: Use **surrogate keys**.
- BP1259: Use indexes.
- BP1140: Use SQL-2003 features in preference to **SQL-92** or **SQL-99**.
- BP1139: Do not use proprietary **SQL** extensions.
- BP1143: Use a **database modeling** tool that supports a two-level model (**Conceptual**/Logical and **Physical**) and **ISO-11179** data exchange standards.
- BP1145: Use vendor-neutral **conceptual/logical models**.
- BP1227: Do not allow installation of **MSMQ**-dependent clients.

P1059: Overarching Concepts

This section of NESI guidance includes the following complex perspectives:

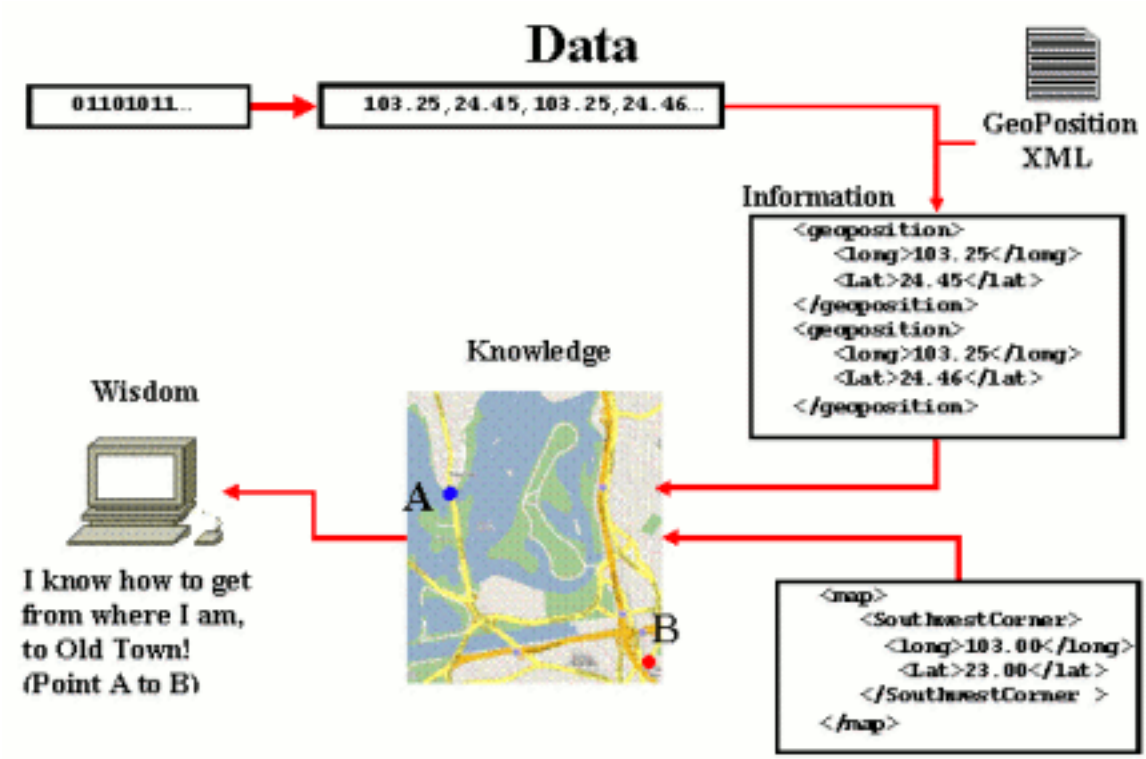
- Data
- Application Security
- Programming Languages

P1012: Data

There are several common definitions of data; the NESI Glossary definition includes the following points:

- **Data** is unprocessed **information**.
- Data is information without context.

But both of these definitions rely on the term "information" which can be a circular definition back to data. To clarify this, the following model helps create definitions of **Information**, **Knowledge** and **Wisdom**. Data flows into the **system** as a set of zeros and ones. The system transforms this initial data into other data that is more understandable from a human perspective (i.e., a list of double precision, floating point numbers). If the numbers are placed into a context such as it is a geographic position, then the data starts to become Information. As information is combined together, the result is referred to as Knowledge (i.e., the knowledge of where one is). When the knowledge can support making decisions, the results are Wisdom (i.e., how to get from point A to point B).



Within NESI, the term Data covers the entire data spectrum (i.e., Information, Knowledge and Wisdom) with a focus is on the transfer of data between components. There have been several major efforts within the **DoD** that have addressed the need to understand, control and document the flow of data between components. NESI is not in competition with these efforts nor is it intended to render these efforts obsolete. NESI provides detailed guidance intended to verify that the concepts and **tenets** of these efforts are met.

Generic data guidance statements include guidelines relative to basic functions associated with the definition of data and the most general categories of data types. Examples of the most basic data functions include **data modeling** and **domain analysis**. The most general categories of data types include **relational database** data and **XML**.

Data Exposure defines the steps necessary to set up the **metadata** infrastructure associated with a net-centric data strategy. This infrastructure permits the exposure (i.e., visibility) of net-centric data to the user community. This infrastructure will be set up once but maintained to include the following:

- Registry where the metadata will reside

- Repository where the data will reside
- Rules applicable to the tagging of data

Tagging and metadata rules follow from **Data Categorization**. Generic Data Categorization includes data types that adhere to **XML Schema** rules. Specialty Data Categories, such as **Electronic Data Interchange** (**EDI**) and **Binary XML** include data types that do not fit in the current XML paradigm but for which special XML extensions may be developed.

Data Publishing defines the steps necessary to make data available within the net-centric data strategy infrastructure. It requires the project to have a **Community of Interest** (**COI**), a model of the data associated with the project and an **ontology** which taken together can be used as a basis for structural metadata. Based on the Data Categorization rules promulgated in the data exposure section appropriate tags are determined and applied to the data

Detailed Perspectives

- XML
- Metadata Registry
- Data Modeling
- ASD(NII) Net-Centric Checklist
- Metadata

P1083: XML

The **Extensible Markup Language** (**XML**) is a **World Wide Web Consortium** (**W3C**) initiative that allows encoding **data** and information with meaningful structure and **semantics** into a document that computers and humans can read easily. XML is ideal for information exchange and is easily extended to include other data types. The ubiquitous nature of XML within existing and proposed DoD projects has spawned a lot of activity to capture guidelines and requirements that facilitate net-centricity and interoperability. Many of these activities have not been finalized and are "emerging" from a NESI viewpoint. This NESI Perspective leverages the work done by Roger Costello and colleagues at xFront.com. It is by no means complete, but it does provide a starting point for additional DoD XML work.
There are two key measures of XML instance document correctness: being well-formed and valid. Those concepts and others are introduced in the following perspectives:

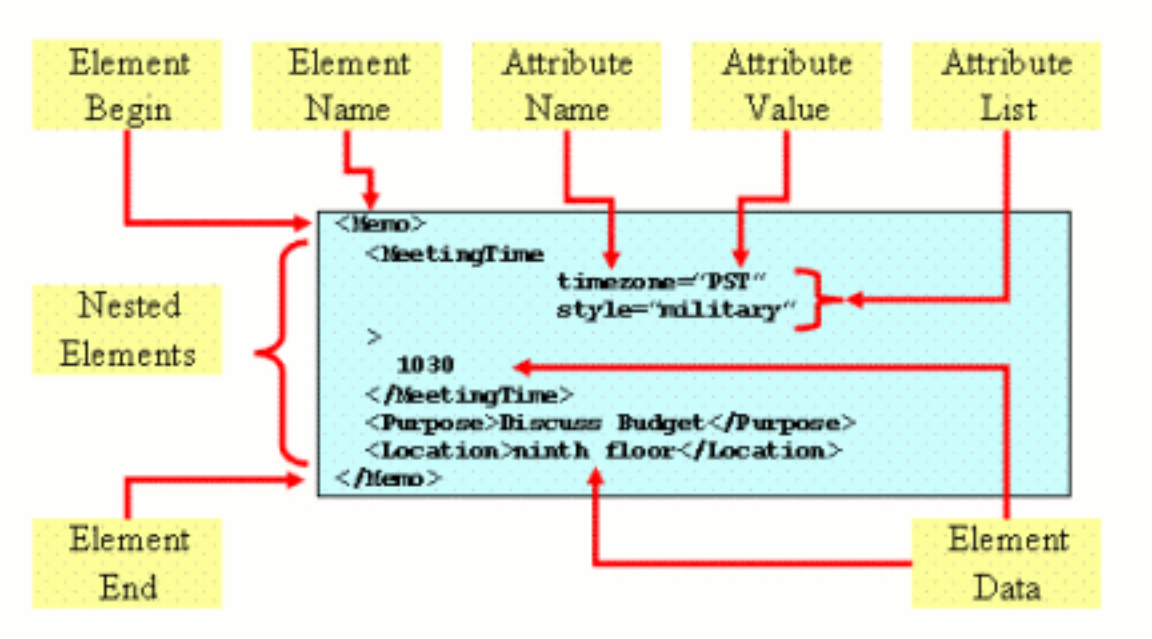
- XML Syntax
- XML Semantics
- XML Processing

References

- R1116: W3C Extensible Markup Language (XML)
- R1117: http://xfront.com/BestPracticesHomepage.html
- R1118: Microsoft Standards Reference - http://msdn2.microsoft.com/en-us/library/ms256177.aspx
- R1119: Component Organization and Registration Environment - https://www.collab.core.gov/CommunityBrowser.aspx?id=2234
- R1120: Federal XML Naming and Design Rules - http://xml.coverpages.org/Federal-NDR-20050609.pdf

P1095: XML Syntax

The syntax of an **XML document** is a hierarchical collection of elements that identify the name of the **data** within the XML document and the value associated with the element. Elements can have **attributes** and be nested within other elements. The following is a simplistic XML document displayed in **ASCII** with the major syntactical **components** labeled.



Guidance

- G1724: Develop XML documents to be well formed.

Best Practices

- BP1258: Explicitly define the encoding style of all data transferred via **XML**.
- BP1752: Place dynamic element data within a CDATA section.

Examples

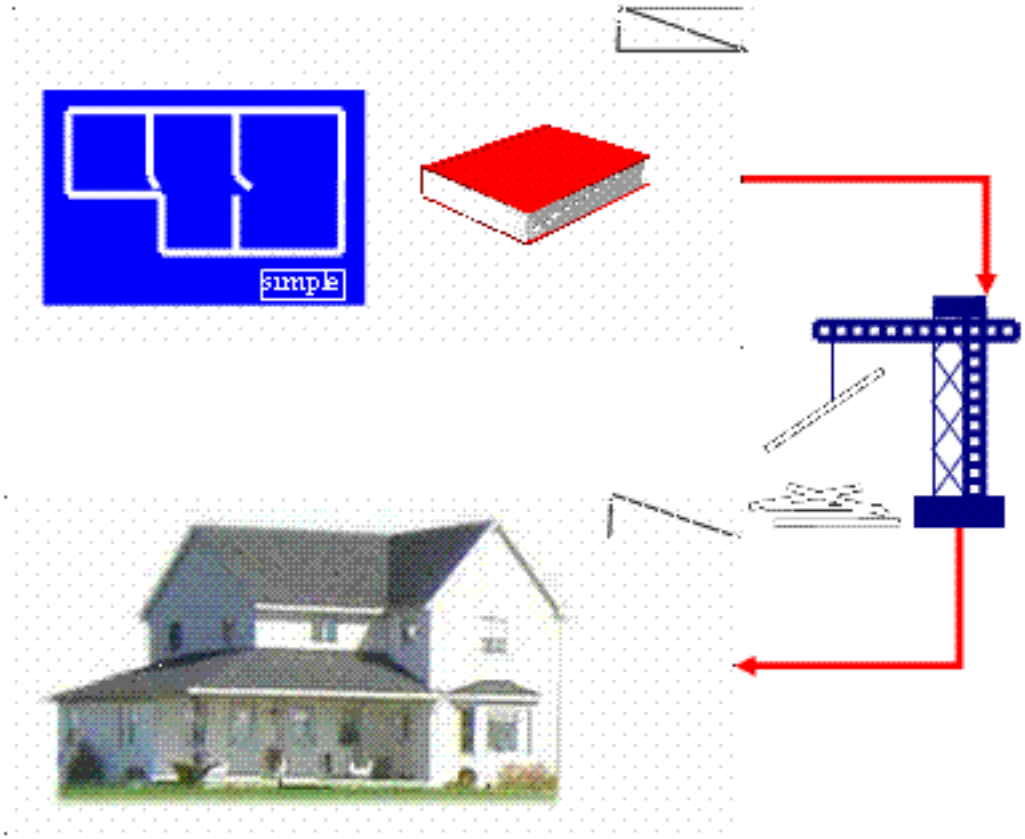
An example of an XML instance document is the following weather information XML. It can be thought of as a complex data structure that contains a weather station's data.

References

- R1121: W3C Extensible Markup Language (XML) 1.0 (Fourth Edition) - http://www.w3.org/TR/2006/REC-xml-20060816/
- R1122: W3 Schools XML Syntax Rules Tutorial - http://www.w3schools.com/xml/xml_syntax.asp
- R1123: W3 Schools XML Validator Tutorial - http://www.w3schools.com/xml/xml_validator.asp

P1096: XML Semantics

The semantics of an **XML document** are limited to the structural composition of data, the relationships of the structures to each other, and the rules governing data content. A full semantic interpretation of the **XML** content must be left to humans or tools that humans have written that connote some meaning to the data. For example, the semantics captured by XML might define a weather station that is comprised of air temperature, soil temperature, anemometer and hygrometer and the values and units associated with these values. XML does not capture what this data means semantically to a pilot or soldier.



The semantics of any XML instance document are captured in another XML document called the schema which is also defined using XML. Therefore, the semantics discussion is divided into two sub-perspectives:

- XML Schema Documents
- XML Instance Documents

References

- R1124: http://www.w3.org/2001/XMLSchema.xsd
- R1125: http://www.xfront.com/xml-schema.html

P1104: XML Instance Documents

An **XML instance document** is an **XML document** which is defined by an **XML Schema** but is populated with the actual data whereas the schema is the definition of the structure and semantics of data (**metadata**).

Guidance

- G1725: Develop XML documents to be valid XML.
- G1736: Separate document schema definition and document instance into separate documents.

Best Practices

- BP1742: Use the xsi qualifying prefix for XML Schema instance namespace uses.
- BP1743: Use .xml as the file extension for files that contain XML Instance Documents.

P1097: XML Schema Documents

An **XML Schema** is a **W3C** specification for defining the **semantics** and structure of **XML documents**. For a discussion of the grammar that governs **XML** see the XML Syntax perspective. The semantics are limited to the structural composition of data, the relationships of the structures to each other, and the rules governing data content. The discussions of the schema documents are broken down into schema subject areas:

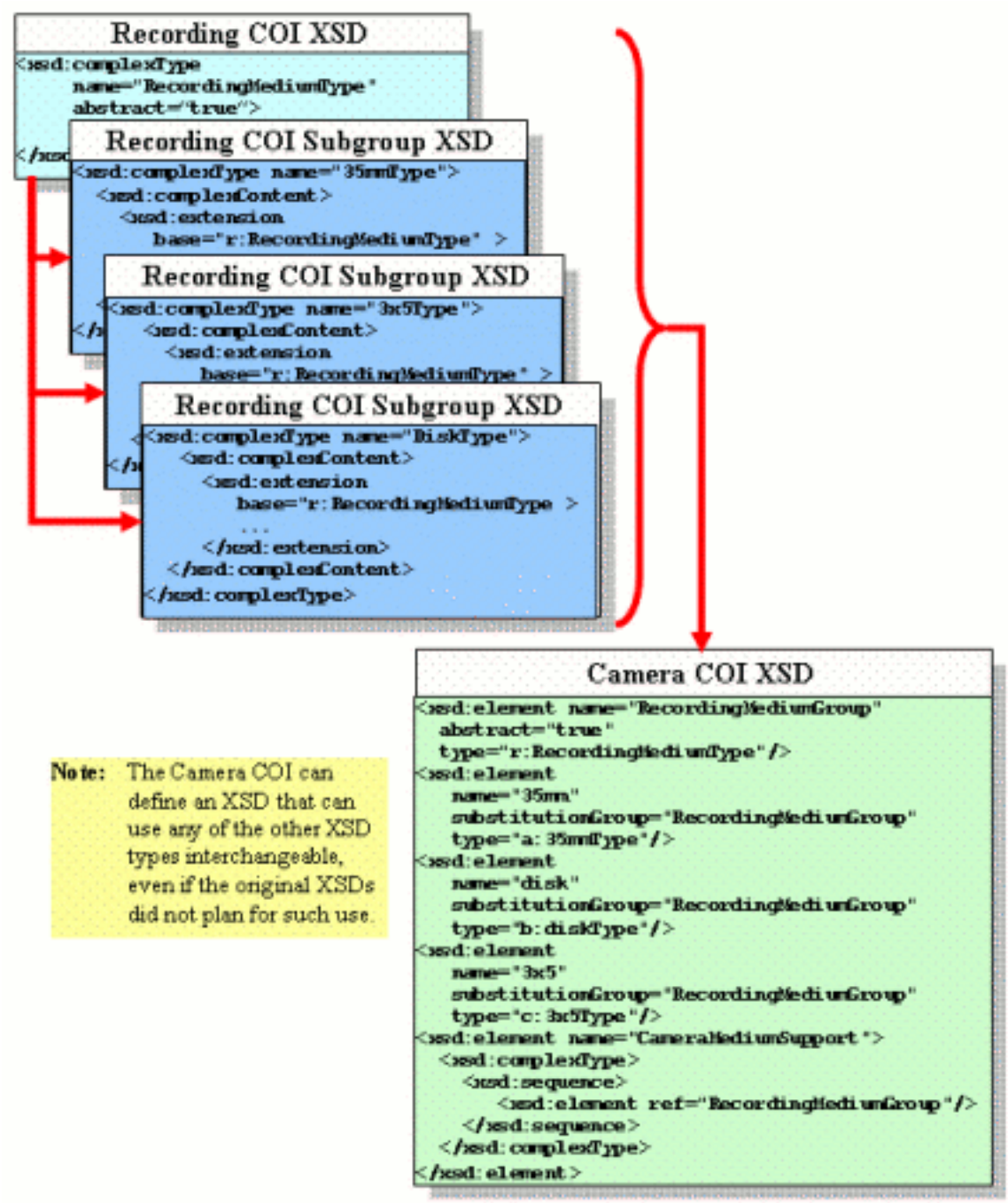
- Defining XML Schemas
- XML Schema Files
- Using XML Namespaces
- Defining XML Types
- Using XML Substitution Groups
- Versioning XML Schemas

References

- R1124: http://www.w3.org/2001/XMLSchema.xsd
- R1126: http://www.xfront.com

P1102: Using XML Substitution Groups

Substitution groups allow using elements defined in externally defined and controlled schemas as interchangeable elements in new schemas. The members of the substitution group do not have to be derived from the same type. This allows any of the element members' substitution group elements to participate as a member of a more abstract concept. For example, in the following **XML**, **`RecordingMedium`** is the name of the substitution group. The members of the group are the **`RecordingMedium`** element itself and **`35mm`**, **`disk`** and **`3x5`**. Anywhere that **`RecordingMedium`** is used as a reference, **`35mm`**, **`disk`** and **`3x5`** can also be used. For a complete example study the following diagram that defines a **`CameraMediumSupport`** element that has a single sequence comprised of the **`RecordingMediumGroup`** substitution group.



Guidance

- G1731: Only reference Elements defined by a Type in substitution groups.
- G1744: Only reference abstract Elements in substitution groups.
- G1745: Append the suffix Group to substitution group element names.

References

- R1132: Composition versus Subclassing - http://www.xfront.com/composition-versus-subclassing.html

P1099: XML Schema Files

Schema definitions are usually captured in files. The following guidance applies to those files which actually contain the schema definitions.

Guidance

- G1735: Use the .xsd file extension for files that contain XML Schema definitions.
- G1736: Separate document schema definition and document instance into separate documents.

Examples

```
<?xml version="1.0"?>
<xsd:schema xmlns: xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.camera.org"
            xmlns: nikon="http://www.nikon.com"
            xmlns: olympus="http://www.olympus.com"
            xmlns: pentax="http://www.pentax.com"
            elementFormDefault="unqualified">
 <xsd:import namespace="http://www.nikon.com"/>
 <xsd:import namespace="http://www.olympus.com"/>
 <xsd:import namespace="http://www.pentax.com"/>
 <xsd:element name="Camera">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="body"
                    type="nikon:BodyType"/>
       <xsd:element name="lens"
                    type="olympus:LensType"/>
       <xsd:element name="ManualAdapter"
                    type="pentax:manual_adapter_type"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

P1098: Defining XML Schemas

While it is possible to use **Document Type Definitions** (**DTD**) to convey much of the same information as the **XML Schema Definition** (**XSD**), XSDs have several distinct advantages which are very useful in terms of interoperability. **XML Schemas** have richer support for defining and using types than DTDs which capture domain information such as allowable ranges and units. For example, XSDs can define an elevation type with values limited to meters in the range of 0 to 12,000.

Guidance

- G1725: Develop XML documents to be valid XML.
- G1726: Define XML Schemas using **XML Schema Definition** (XSD).
- G1730: Follow an XML coding standard for defining schemas.
- G1045: Define **XML** format information separately in **XSL**.

Best Practices

- BP1732: Follow the Upper Camel Case (UCC) naming convention for XML Type names.
- BP1733: Follow the Upper Camel Case (UCC) naming convention for XML Element names.
- BP1734: Follow the Lower Camel Case (LCC) naming convention for XML Attributes.

References

- R1120: Federal XML Naming and Design Rules - http://xml.coverpages.org/Federal-NDR-20050609.pdf
- R1125: http://www.xfront.com/xml-schema.html

P1100: Using XML Namespaces

A **namespace** defines the scope for schema components and de-conflicts the use of schema components. Qualifying prefixes simplify the use of namespaces in names by appending a qualifier onto the beginning of the name that is mapped to a particular schema. Namespaces can become quite confusing if they are not used consistently.

Guidance

- G1737: Define a target namespace in schemas.
- G1738: Define a qualified namespace for the target namespace.
- G1385: Identify **XML Information Resources** for registration in the XML Gallery of the **DoD Metadata Registry**.
- G1383: Use a **registered namespace** in the XML Gallery in the **DoD Metadata Registry**.
- G1085: Establish a **registered namespace** in the **XML Gallery** in the **DoD Metadata Registry** for all DoD Programs.
- G1384: Review **XML Information Resources** in the **DoD Metadata Registry**, using those which can be reused.

Best Practices

- BP1739: Use the xsd qualifying prefix for XML Schema namespace.
- BP1741: Do not provide a schema location in import statements in schemas.
- BP1742: Use the xsi qualifying prefix for XML Schema instance namespace uses.

References

- R1120: Federal XML Naming and Design Rules - http://xml.coverpages.org/Federal-NDR-20050609.pdf
- R1127: http://www.xfront.com/ZeroOneOrManyNamespaces.pdf
- R1128: http://www.xfront.com/DefaultNamespace.pdf

P1103: Versioning XML Schemas

XML Schemas capture the **semantics** of the **data** that the schemas define. As the understanding of the data and its interrelationships evolves, the need to redefine the semantics captured by the schema is inevitable. This evolution can have a wide ranging ripple effect throughout a large widely distributed system or family of systems. Therefore, the uniform managing of schema versions is essential.

Guidance

- G1753: Declare the XML schema version with an attribute in the root element of the schema definition.
- G1754: Give each new XML schema version a unique URL.
- G1727: Provide names for type definitions.
- G1004: Make public **interfaces** backward-compatible within the constraints of a published **deprecation** policy.
- G1019: Deprecate public interfaces in accordance with a published deprecation policy.

P1101: Defining XML Types

The **W3C** defined datatype as follows:

"A datatype is a 3-tuple, consisting of a) a set of distinct values, called its value space, b) a set of lexical representations, called its lexical space, and c) a set of facets that characterize properties of the value space, individual values or lexical items."
[See W3C "XML Schema Part 2: Datatypes Second Edition," Section 2.1, http://www.w3.org/TR/xmlschema-2/#typesystem]
There are two kinds of datatypes definable within XML: Primitive and Derived. Primitive datatypes are not defined in terms of other datatypes while Derived datatypes are defined in terms of other datatypes. All datatypes can be further classified as Built-in and User-derived. Built-in datatypes are those which have been defined by the W3C in XML Schema Part 2: Datatypes Second Edition. User-derived datatypes are those defined by individual schema designers.

The guidance included in this perspective is for primitive and derived datatypes designed by individual schema designers.

Guidance

- G1727: Provide names for type definitions.
- G1728: Define types for all elements.
- G1729: Annotate type definitions.
- G1740: Append the suffix Type to type names.

Best Practices

- BP1732: Follow the Upper Camel Case (UCC) naming convention for XML Type names.

References

- R1129: Extensible Content Models - http://www.xfront.com/ExtensibleContentModels.pdf
- R1130: Element versus Type - http://www.xfront.com/ElementVersusType.pdf
- R1131: XML Schema Part 2: Datatypes Second Edition - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes

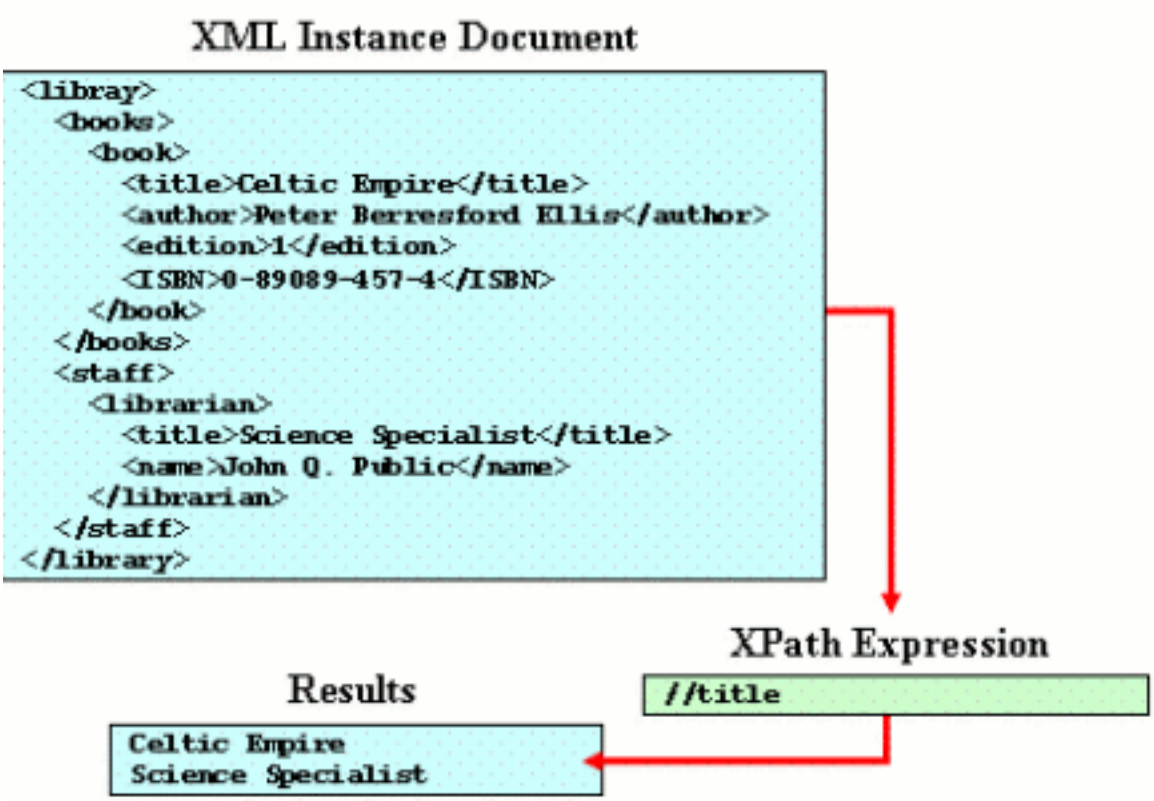
P1105: XML Processing

One of the primary benefits of using **XML** is that it can be read by humans or processed by software. The following perspectives pertain to XML processing:

- XSLT
- XPath
- Parsing XML
- XML Validation

P1107: XPath

A valid **XML Document** is a representation of a **Document Object Model** (**DOM**) tree structure. Each of the XML elements is considered a node with the tree. **XML Path Language** (**XPath**) is a succinct and elegant way of addressing the individual nodes (i.e., elements) within the tree (i.e., document) or to perform basic computations on the Element Data within the document. The following is a very simplistic example of how an XML Document and XPath work together. The XML instance document contains the data and the XPath provides the instructions on how to traverse the document.



For a more detailed description of XPath, see the following W3C location: http://www.w3.org/TR/xpath; there also is an XPath tutorial at http://www.w3schools.com/xpath/default.asp.

Guidance

- G1756: Isolate XPath expression statements into the configuration data.

Best Practices

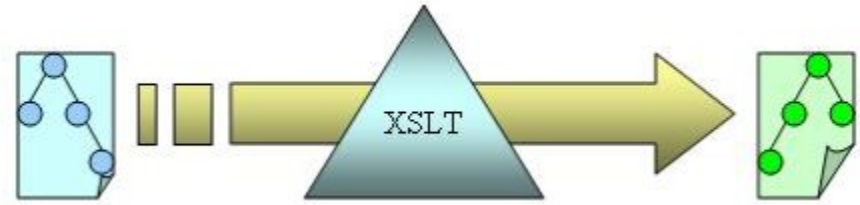
- BP1757: Do not ignore namespace prefixes in XPath expressions.
- BP1758: Make names in descendant expressions unique within an XML document.

References

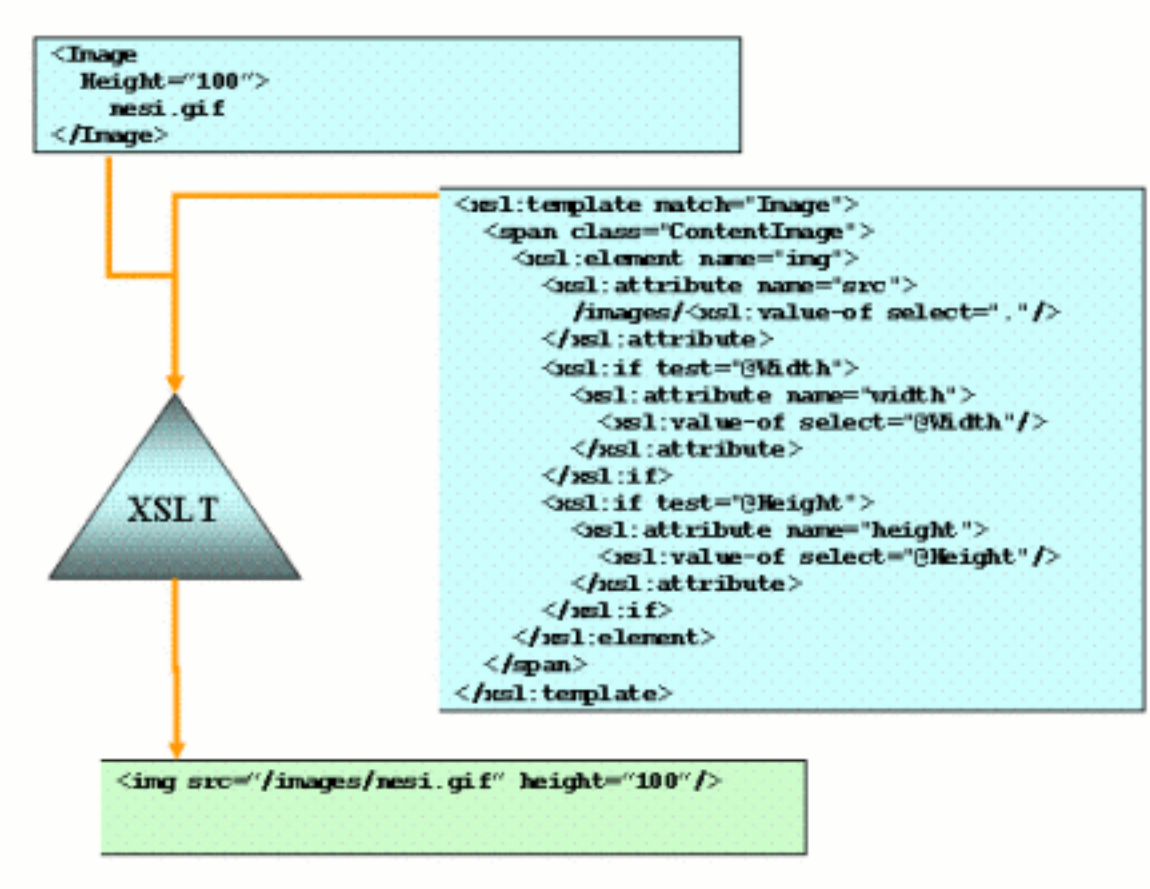
- R1133: http://www.w3.org/TR/xpath
- R1134: http://www.w3schools.com/xpath/default.asp
- R1135: XPath (Wikipedia) - http://en.wikipedia.org/wiki/XPath
- R1136: HOWTO: Write Namespace-Agnostic XPath and XSLT - http://jcooney.net/archive/2005/08/09/6517.aspx

P1106: XSLT

eXtensible Stylesheet Language Transformation (**XSLT**) allows **XML** data transformation using the functional **eXtensible Stylesheet Language** (**XSL**).



XSL is dependent on **XML Path Language** (**XPath**) to address nodes within the input document. For XPath guidance and best practices see the XPath perspective. The following example produces **HTML** image tag from an image XML element with optional height and width attributes.



Templates

Use templates to transform particular sections of an XML document tree. XSLT requires at least one template which matches to an absolute path of an element (e.g., `/`). Inside of a template, match other templates by using **`xsl:apply-templates`**. Passing an XPath query to the select parameter of **`xsl:apply-templates`** constructs a list of nodes by which templates are compared and executed.

XSLT 2.0

XSLT 2.0 improves on XSLT 1.0 and adds functionality that was previously only achieved through proprietary language extensions.

Some of the more significant improvements include the following:

- Backwards-compatibility
- Improved XPath functions
- Regular expressions
- Schema validation to temporal and result trees
- Multiple outputs
- Aggregation
- Strong data typing

Guidance

- G1746: Develop XSLT stylesheets that are XSLT version agnostic.
- G1751: Document all XSLT code.
- G1755: Use accepted file extensions for all files that contain XSL code.

Best Practices

- BP1747: Use the xsl qualifying prefix for XSLT namespace.
- BP1748: Separate static content from transformational logic in XSLTs.
- BP1749: Use xsl:include for including XSL transforms.
- BP1750: Use xsl:import for reusing XSL code.

References

- R1054: XSL Transformations (XSLT) Version 2.0
- R1056: The Extensible Stylesheet Language Family http://www.w3.org/Style/XSL

P1110: XML Validation

One advantage of **XML** is that it allows for validation of **XML instance documents**. Validation can occur at the **producer** and/or **consumer** or anywhere in-between.

Guidance

- G1725: Develop XML documents to be valid XML.

Best Practices

- BP1265: Validate **XML** idocuments during document generation.

P1109: Parsing XML

One advantage of **XML** is that a variety of standard **parsers** are available to parse documents. Another advantage is that the consumer of the XML document is free to choose the type of parser to use.

A couple of common types of XML parsers include the **Document Object Model** (**DOM**) and Simple API for XML (SAX) parsers. The DOM parser uses a tree-based approach, while the SAX parsers use an event-based approach. Both approaches have advantages and disadvantages depending the application.

In addition to the various types of XML parsers, there are multiple implementations of each types of parser. This provides the developer great flexibility in choosing an XML parser implementation. To take advantage of this flexibility, the developer must take care when developing software to allow for changing the XML parser throughout the life-cycle of the software. One way to do this is to provide a wrapper or adapter class that isolates the XML parser implementation allowing for changes to the XML parser during development or deployment.

Best Practices

- BP1769: Provide wrapper or adapter classes to isolate XML parser implementations.

References

- R1137: http://www.w3.org/DOM/
- R1138: http://www.saxproject.org/

P1050: Metadata Registry

A Metadata Registry is a central repository for storing and maintaining **metadata** definitions. A Metadata registry typically has the following characteristics:

- It is a protected area where only approved individuals may make changes
- It stores **data elements** that include both semantics and representations
- The semantic areas of a metadata registry contain the meaning of a **Data Element** with precise definitions
- The representational areas define how the data is represented in a specific format such as within a database or a structure file format such as **XML**

Metadata Registries often are stored in an international format called **ISO-11179**.

A Metadata Registry is frequently set up and administered by an organization's **Data architect** or data modeling team.

The **DoD Metadata Registry** provides a common source of data information required to promote interoperability in the Net-Centric Data Environment.

"Defense Information Systems Agency (DISA) is responsible for data services and other data-related infrastructures that promote interoperability and software reuse in the secure, reliable, and networked environment planned for the DoD's Global Information Grid (GIG). The Metadata Registry and Clearinghouse's primary objective is to provide software developers access to data technologies to support DoD mission applications. Through the Metadata Registry and Clearinghouse, software developers can access registered XML data and metadata components, COE database segments, and reference data tables and related meta-data information such as Country Code and US State Code. These data technologies increase the DoD's core capabilities by integrating common data, packaging database servers, implementing transformation media and using Enterprise data services built from "plug-and-play" components and data access components."
[http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal]

In the Net-Centric Data Strategy, data sources are called **Data Assets** which are divided into two generic areas:

The **data** area includes the following:

- **XML** stored in repositories (files)
- **Database data**
- Data services
- Data streams (real time)
- Sensor data
- **Message** data (includes **EDI**)

The **metadata** area includes the following:

- Metadata Stored in Registries
  - **UDDI**
  - **ebXML**
  - DoD Metadata Registry

    - Other **ISO/IEC 11179 Registries**
    - Discovery metadata stored in Catalogs
- DoD Discovery Metadata Standard (**DDMS**)
- Interface Metadata (**WSDL**)
- Structural Metadata (**XSD**)

Data comes in many forms. It can be simple or complex; structured or unstructured in nature.

Simple Structured Data has an uncomplicated data structure. All requisite metadata is provided and simple data types only are used (e.g., integers, long integers, strings, and simple lists).

Simple Unstructured Data has uncomplicated data structure but not all requisite Metadata is provided.

Complex Structured Data has well-defined metadata. It includes data represented in **XML documents** with deeply hierarchical and recursive structures. Complex data can be represented in a complex data structure or can be mapped into a relational or flat structure with additional metadata provided to represent the complex relationships. Although Complex structured data is generically a property of object oriented databases, the Complex Data Structures can be filled from any source.

- Data
    - XML files
    - defined by **XML Schemas** (**XSDs**)
        - **Interface**
- Metadata stored in DoD Repository
    - XML Schemas (XSDs)
    - Discovery metadata
        - **WSDL**
        - **UDDI**
    - Web Service Source Code
    - XSDs include element validation and descriptions
    - XSDs may import other XSDs
    - XSDs are validated
    - **Complex Structured Data** follows all of the **XML** rules.

Note: *The source of this data can be any.*

Complex Semi-Structured Data has partial metadata. It includes data defined in **COBOL** copybooks and Electronic Data Interchange standards **ANSI X.12** and Health Level 7 (HL7). Semi-structured data can be as complex or more so as any Complex Structured data. It can map into or be XML. It may also be missing some Metadata or an XSD.

Complex Unstructured Data has little or no metadata. It includes data in binary files, spreadsheets, documents, and print streams.

Guidance

- G1382: Be associated with one or more **Communities of Interest** (**COIs**).
- G1383: Use a **registered namespace** in the XML Gallery in the **DoD Metadata Registry**.
- G1384: Review **XML Information Resources** in the **DoD Metadata Registry**, using those which can be reused.
- G1385: Identify **XML Information Resources** for registration in the XML Gallery of the **DoD Metadata Registry**.
- G1386: Review predefined commonly used **data elements** in the **Data Element Gallery** of the **DoD Metadata Registry**, using those in the **relational database** technology which can be reused in the Program.
- G1387: Identify **data elements** created during Program development for registering in the **Data Element Gallery** of the **DoD MetaData Registry**.
- G1388: Use predefined commonly used database tables in the **DoD Metadata Registry**.
- G1389: Publish database tables which are of common interest by registering them in the **Reference Data Set** Gallery of the **DoD Metadata Registry**.
- G1390: Standardize on the terminology published by relevant **COIs** listed in the **Taxonomy Gallery** of the **DoD Metadata Registry**.
- G1391: Identify **taxonomy** additions or changes in conjunction with the **COIs** during the Program development for potential inclusion in the **Taxonomy Gallery** of the **DoD Metadata Registry**.
- G1392: Adhere to a common mechanism of service location.
- G1125: Use the **Department of Defense Metadata Specification** (**DDMS**) for standardized tags and taxonomies.

Best Practices

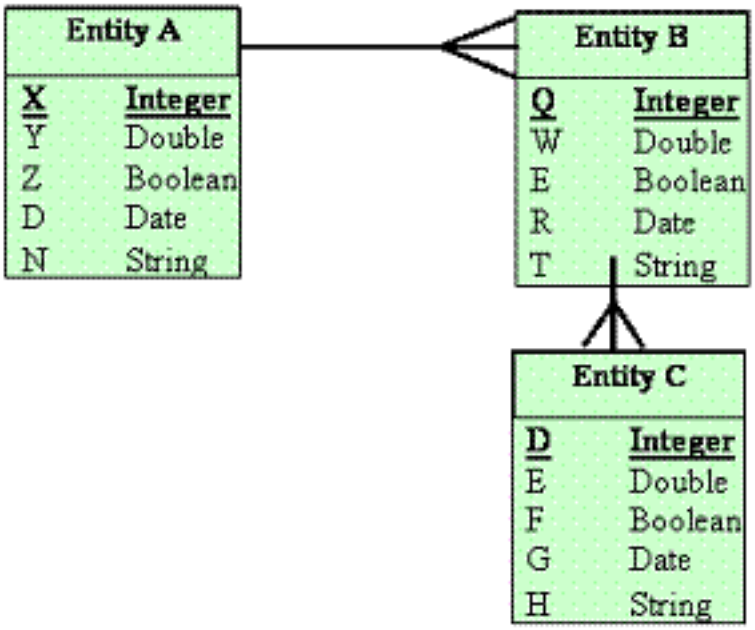
- BP1404: For DoD Programs requiring a **data model**, the **NATO** Generic Hub v.5 model (**LC2IEDM**) is an example of a successful **COI**-developed model.

References

- R1051: DoD Meta Data Registry for XSLTsamples.[http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal]
- R1052: XSLT 1.0 (Extensible Stylesheet Language Transformations), for transforming XML documents [http://www.w3.org/TR/xslt]
- R1053: XSLT 2.0 (W3C Working Draft, 5 November 2004) [http://www.w3.org/TR/xslt20]
- R1055: XSL (Extensible Stylesheet Language) 1.0, presentation rules to transform a document. [See http://www.w3.org/Style/XSL and http://www.w3.org/TR/xsl]
- R1058: CSS (Cascading Style Sheets) versions 1 (CSS1) and 2 (CSS2) [See http://www.w3.org/Style/CSS, http://www.w3.org/TR/REC-CSS1, http://www.w3.org/TR/REC-CSS2]

P1003: Data Modeling

Modeling is an essential step in understanding the data that will comprise a system. Before implementing a system, it is important to understand the basic **data elements** and the relationships of the elements. The end products of **data modeling** can be **XML schemas**, **RDBMS** schema definitions or the data portion of objects.



The following guidance applies to the data model used to describe the data tier.

Guidance

- G1141: Use standard **data models** developed by **Communities of Interest** (**COI**) as the basis of program or project data models.
- G1144: Develop two-level database models: one level captures the **conceptual** or logical aspects, and the other level captures the **physical** aspects.
- G1147: Use **domain analysis** to define the constraints on input data validation.
- G1148: **Normalize** data models.

Best Practices

- BP1394: Identify, publish and validate data objects exposed to the enterprise early in the data engineering process and update in a spiral fashion as system development proceeds.
- BP1397: For new systems, identify and develop use cases or reuse existing use cases as appropriate as early in the data engineering process as possible to support **data model** development.
- BP1398: Develop Interaction models as appropriate.
- BP1400: Programs will use authoritative **metadata** established by the Joint Mission Threads (JMTs) when available.

- BP1145: Use vendor-neutral **conceptual/logical models**.
- BP1396: Develop high-level conceptual data models for new systems prior to Milestone A based on the business process context in which the system will be used.

P1006: ASD(NII) Checklist

The purpose of the Net-Centric Checklist is to assist in the development of programs need in the net-centric environment as part of a **service-oriented architecture** (**SOA**) in the **Global Information Grid** (**GIG**). An SOA is a design style for building flexible, adaptable distributed-computing environments for the Department of Defense (DoD). Service-oriented design is fundamentally about sharing and reusing functionality across diverse applications. There are four sections in the Checklist: Data, Services, IA/Security, and Transport.

This perspective, including the following table, describes how the NESI Guidance relates to the **ASD(NII)** Net-Centric Checklist Data **tenets**. The first four columns of the table refer to the ASD(NII) Net-Centric Checklist; the final two columns contain the NESI approach and comments, respectively.

Section	Data Tenet Name	Text	Rationale	Approach	Comment
I. B. 01	Make data visible	Does the system provide discovery **metadata**, in accordance with the DoD Discovery Metadata Standard (DDMS), for all data posted to shared spaces?	Rationale Users and applications will migrate from maintaining private data (e.g., data kept within system specific storage) to making data available in community- and Enterprise-shared spaces (e.g., servers and services available on the Internet). Data will migrate from being maintained in private data stores alone, to being made available in community and Enterprise shared spaces.	Answered if **DoD Metadata Registry** Used. Also G1125	Included in DDMS Guidance
I. B. 02	Make data visible	Describe how the system is making its data assets visible to consumers.	Rationale Question will determine whether a consumer needs to know about a data asset and establish a point-to-point connection, or whether the	Answered if DoD Metadata Registry Used	Data assets are made available via registered services in the DoD Metadata Registry

			data asset be discovered.		
I. B. 02	Make data visible	Is all of the data that can and should be shared externally beyond the programmatic bounds of your system visible (i.e., advertised) to all potential consumers of the data?	Rationale Question will identify if the application is making use of Web services to expose its data.	Requires evaluator interaction	This is too subjective and cannot be readily evaluated.
I. B. 03	Make data visible	Describe how consumers are able to locate the data assets available from your system.	Rationale Question will determine whether a consumer needs to know about a data asset and establish a point-to-point connection, or whether the data asset be discovered.	Answered if DoD Metadata Registry Used. Also G1392	Data assets are locatable via registered services in the DoD Metadata Registry
I. B. 04	Make data visible	Describe how the system is making use of Web service standards (e.g., **SOAP** [Simple Object Access Protocol], **WSDL** [Web Services Description Language], **UDDI** [Universal Description, Discovery and Integration]) to make its data assets visible.	Rationale: Question will elicit whether the program is taking advantage of some of the open standards for Web services. (Also referenced in Net-Centric Operations and Warfare Reference Model)	Answered if DoD Metadata Registry Used. Also G1125	Implementation details

I. B. 05	Make data visible	Describe any subscribe/ notify mechanisms for the visible data assets available within the program that alert users and other applications when data has been created or updated.	Rationale Question will elicit whether a consumer can be notified when data assets change.	Answered if DoD Metadata Registry Used	Subscription and Notification methods provided in the DoD Metadata Registry registration requirements
I. B. 06	Make data visible	Describe where potential consumers can go to become aware of the data assets being made visible by your program.	Rationale: Question should elicit how the programs data is being advertised to potential consumers.	Answered if DoD Metadata Registry Used	Data assets are made available via registered services in the DoD Metadata Registry
I. B. 07	Make data visible	Describe how the program provides dynamic, flexible, and threat-tailorable solutions for exchanging data assets between different security domains (i.e., cross-domain) with flexibility to accommodate new operational needs with minimal impact on system and mission performance.	Rationale: DoD 8500 series, DCID 6/3	Answered as part of NESI Part 5 Security	Security - Data assets are made available via registered services in the DoD Metadata Registry. Security constraints should be contained therein

I. B. 08	Make data visible	Describe how data posted to shared spaces is controlled and managed by the applicable security policies, or regulations and how these IA controls are enforced. [Ref RCD 4.1, policy management, 4.3.2 , Information Access Management, 4.5 Access Control]	Rationale: Question will elicit details of design of information security characteristics of system data	Answered as part of NESI Part 5 Security	Security
I. C. 01	Make data accessible	Are there any limitations for the client appliance (e.g., workstation, desktop, laptop, PDA [personal digital assistant]) to access your data assets?	Rationale: Question will elicit whether the program is client neutral and supports standard presentation protocols.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. This information should be provided therein.
I. C. 01. D	Make data accessible	Describe for each visible data asset what the data consumer needs to access the data (e.g., an application client, a Web portal, access to a Web service, access to a shared data storage area, an **XML** (eXtensible Markup Language)	Rationale: Question will elicit whether the program is client neutral and supports standard presentation protocols.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. This information should be provided therein.

		schema/ parser, etc.).			
I. C. 01. F	Make data accessible	Is all of the data that can and should be shared externally beyond the programmatic bounds of your program accessible to all potential consumers of the data with sufficient access permissions and without any additional programming effort?	Rationale: Question will elicit whether the program is client neutral and supports standard presentation protocols.	Requires evaluator interaction	This is too subjective and cannot be readily evaluated.
I. C. 03	Make data accessible	Describe the programs architecture and the data separation from the presentation and business logic.	Rationale Question will elicit whether the program is an n-tier architecture where the data has been isolated from the business logic.	Requires evaluator interaction	Implementation details
I. C. 04	Make data accessible	Describe the security mechanisms used to restrict access to specific, visible data assets. How will the associated **metadata** labels be used to support these security mechanisms? (ref. RCD 4.1, IA Policy Management, 4.3.2 Information Access	Rationale Question will elicit whether appropriate security has been placed on data assets.	Answered as part of NESI Part 5 Security	Security

		Management, 4.5 Access Control)			
I. C. 05	Make data accessible	What mechanisms are planned/ implemented to protect the data in transit to the consumer? This would include protection from modification of the data, protection from unauthorized eavesdropping, or protection from data becoming lost in transit. [ref RCD 3.1 Confidentiality, 4.1, IA Policy Management, 4.6.1, EIAU Management]	Rationale: Question will elicit information on what confidentiality, integrity, and availability mechanisms beyond Inline Network Encryptor (INE) functions are in the system design.	Answered as part of NESI Part 5 Security	Security
I. C. 06	Make data accessible	What mechanisms are planned/ implemented to protect the data at rest within a consumer client? This would include protection from modification of the data, protection from unauthorized disclosure, or protection from data becoming corrupted or otherwise unavailable for mission use. [ref RCD 3.1 Confidentiality, 4.1, IA Policy	Rationale: Question will elicit information on what confidentiality, integrity, and availability mechanisms are envisioned where the end-user will be processing GIG data.	Answered as part of NESI Part 5 Security	Security

		Management, 4.6.1, EIAU Management]			
I. C. 07	Make data accessible	What mechanisms are planned/ implemented to protect the data at rest within the service providers systems? This would include protection from modification of the data, protection from unauthorized eavesdropping, or protection from data becoming corrupted or otherwise unavailable for mission use. [ref RCD 3.1 Confidentiality, 4.1, IA Policy Management, 4.6.1, EIAU Management]	Rationale: Question will elicit information on what confidentiality , integrity, and availability mechanisms are envisioned where the end-user will be processing **GIG** data.	Answered as part of NESI Part 5 Security	Security
I. C. 08	Make data accessible	Describe how the visible data assets are made available to other users outside the Community of Interest with a need for the data.	Rationale Question should help the assessor determine how easily the data is accessible.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. Any consumer who has access to the DoD Metadata Registry will have access to these data assets.
I. C. 09	Make data accessible	Describe the common design patterns employed in the program that aid in the	Rationale Question will elicit whether the program is making use of design patterns to simplify and	Requires evaluator interaction	Content management systems acting as the data catalog for unstructured collections

		accessibility of data assets.	standardize how data assets are accessed.		or structured data archives/ warehouses as data catalogs service generated data
I. C. 10. 00	Make data accessible	Describe the use within the program of the following design patterns:	Rationale Question will elicit more detailed discussion than the previous question. However, the program Requires evaluator interaction will not necessarily employ all of these patterns.	Implementation details	
I. C. 10. 01	Make data accessible	Request-Response	Rationale Question will elicit more detailed discussion than the previous question. However, the program will not necessarily employ all of these patterns.	Requires evaluator interaction	Implementation details
I. C. 10. 02	Make data accessible	Publish-Subscribe	Rationale Question will elicit more detailed discussion than the previous question. However, the program will not necessarily employ all of these patterns.	Requires evaluator interaction	Implementation details
I. C. 10. 03	Make data accessible	Transactional or Read-Only	Rationale Question will elicit more detailed discussion than the previous question. However, the program will not necessarily	Requires evaluator interaction	Implementation details

			employ all of these patterns.		
I. C. 10. 04	Make data accessible	Synchronous or Asynchronous	Rationale Question will elicit more detailed discussion than the previous question. However, the program will not necessarily employ all of these patterns.	Requires evaluator interaction	Implementation details
I. C. 10. 05	Make data accessible	Model-View-Controller	Rationale Question will elicit more detailed discussion than the previous question. However, the program will not necessarily employ all of these patterns.	Requires evaluator interaction	Manual
I. C. 11	Make data accessible	Describe how the program provides assurance that there is timely and reliable access to data assets anytime, anywhere for authorized users/entities. Availability is a core IA function that is critical to ensuring successful mission execution.	Rationale: DoD 8500 series, DCID 6/3. Integrity is a core information assurance (IA) function, and is necessary to provide confidence in data received.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. This information should be provided therein.
I. C. 12	Make data accessible	Describe how access control and IA policy enforcement will be used to ensure that only	Rationale: Question will elicit information on how access control will be implemented in the context	Answered as part of NESI Part 5 Security	Security

		authorized users/entities can access restricted data. (ref. RCD 4.2.2 Authorization/ Privilege Management, 4.3.2 Information Access Management, 4.5 Access Control)	of GIG wide access control policies and identity management.		
I. D. 01. D	Make data understandable	Describe how the program tags data with discovery **metadata**.	Rationale Metadata tagging enables users to discover the data for retrieval. The assessor should assess whether sufficient use of metadata is being made.	Answered if DoD Metadata Registry Used	Data assets in the DoD Metadata Registry should be tagged with discovery metadata as per **DDMS**. Automated tagging is best. There can be variability in the granularity of the data asset tagged but data catalogs should allow discovery metadata registration per DDMS and search per DDMS criteria
I. D. 01-	Make data understandable	Is all of the data that can and should be shared externally beyond the programmatic bounds of your program sufficiently documented and understandable that any potential consumer can comprehend the structural	Rationale: Question will indicate how registered metadata are being used for access control decisions on system data assets.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. The information provided therein should be adequate.

		and semantic meaning to determine if they can reliably use the metadata to make access control decisions on sensitive data? (ref. RCD 4.3.1 Information Labeling Management, 4.5 Access Control)			
I. D.02	Make data understandable	Is all of the data that can and should be shared externally beyond the programmatic bounds of your program sufficiently documented and understandable that any potential consumer can comprehend the structural and semantic meaning to determine how they may use it appropriately?	Rationale Metadata tagging enables users to discover the data for retrieval. The assessor should assess whether sufficient use of metadata is being made.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. This information should be provided therein.
I. D. 03	Make data understandable	Explain how the program is making use of the DoD Metadata Registry and Clearinghouse.	Rationale Question will elicit indications of whether discovery metadata is being generated that is compliant with the DoD Discovery Metadata Specification.	Answered if DoD Metadata Registry Used	Data assets are accessible via registered services in the DoD Metadata Registry. This information should be provided therein.
I. D. 04	Make data understandable	Has the DoD Metadata Registry	Rationale Question will elicit whether	Answered if DoD Metadata Registry Used	Included in DoD Metadata

		been used whenever possible?	the program is making use of existing, registered **data elements** from the Registry.		Registry requirements.
I. D. 04	Make data understandable	Have newly defined XML elements been registered with the Registry?	Rationale Question will elicit whether the program is making use of existing, registered data elements from the Registry.	Answered if DoD Metadata Registry Used	Included in DoD Metadata Registry requirements.
I. D. 04. D	Make data understandable	Describe the source of all XML elements.	Rationale Question will elicit whether the program is making use of existing, registered data elements from the Registry.	Answered if DoD Metadata Registry Used	Included in DoD Metadata Registry requirements.
I. D. 05	Make data understandable	Describe any data schemas or standards being applied in the program.	Rationale Question will elicit whether the program is using **XML Schemas**, DTDs [Document Type Definition], or something similar to describe its data assets.	Answered if DoD Metadata Registry Used	Included in DoD Metadata Registry requirements.
I. D. 06	Make data understandable	Describe any automated mechanisms that are available for data mediation/ translation (e.g., **XSL** [eXtensible Stylesheet Language], **XSD** [XML Schema Definition]).	Rationale Question will elicit any data translation capabilities that are available.	Answered if DoD Metadata Registry Used	Included in DoD Metadata Registry requirements.

I. D. 07	Make data understandable	Describe any automated mechanism that enforce translation of security markings from one policy domain to another. (ref. RCD 4.1 IA Policy Management)	Rationale: Question will elicit any capability to move data from one policy domain (e.g., U.S. Only) to another (e.g., **NATO**)	Answered as part of NESI Part 5 Security	Security
I. E. 01	Make data trustable	Can all potential consumers of all of the data available from your program determine the data pedigree (i.e., derivation and quality), security level, and access control level of your data?	Rationale: Question will elicit how a consumer can determine data asset quality.	Answered if DoD Metadata Registry Used. Further criteria to be established when this section needs to be connected to the NESI Security section	Included in DoD Metadata Registry requirements. Trust here is a function of access to data asset pedigree and identified autorative sources per DDMS. Our approach should be consistent with this.
I. E. 02	Make data trustable	Describe for each visible data asset in the program whether the program is the authoritative data source.	Rationale: Question will elicit whether any data assets are secondary sources.	Answered if DoD Metadata Registry Used	Included in DoD Metadata Registry requirements.
I. E. 03	Make data trustable	Describe what measures the program takes to ensure the integrity of the data (for internally used data, externally used data, and data that simply transits the program).	Rationale: Question will elicit whether data assets are protected against man-in-the-middle types of IA attacks.	Answered as part of NESI Part 5 Security	Security

I. E. 04	Make data trustable	Describe what measures the program takes to ensure that the program data is only provided to consumers via authorized sources. [ref RCD 3.2 Integrity, 4.4 Authentication]	Rationale: Question will elicit whether data assets are protected against man-in-the-middle types of IA attacks.	Answered as part of NESI Part 5 Security	Security
I. F. 01. D	Make data interoperable	Describe any programming changes that would need to be made to the program if a new consumer of a visible data asset were identified.	Rationale: Question will elicit whether new consumers can be added with no additional cost/effort or whether a new point-to-point interface needs to be established.	Requires evaluator interaction	Vague
I. F. 01. F	Make data interoperable	Does all of the data that can and should be shared externally beyond the programmatic bounds of your program have sufficient metadata descriptions and automated support to enable for mediation and translation of the data between interfaces?	Rationale: Question will elicit whether new consumers can be added with no additional cost/effort or whether a new point-to-point interface needs to be established.	Answered if DoD Metadata Registry Used	Fits into DDMS and DoD Metadata Registry Req's. Currently the MDR can store **XSL** to support mediation but much work is needed in this area
I. F. 02	Make data interoperable	Identify the published net-centric interoperability standards (e.g., DDMS)	Rationale: Question will help identify programs that have	Answered if DoD Metadata Registry Used	The approach should reference NR-KPPs and KIPs

		to which the program adheres. (ref. RCD 3.4 Availability)	thought through customer service and planned for accommodating changing consumer needs.		
I. F. 03	Make data interoperable	Describe the process a consumer would follow to a) request changes in the format (syntax or semantic) of the visible data asset; b) report a problem with a data asset; or c) request additional data from the data provider.	Rationale: Question will help to identify programs that have thought through customer service and planned for accommodating changing consumer needs.	Answered if DoD Metadata Registry Used	Vague
I. G. 01. D	Provide Data Management	Describe the effort associated with the program to define, develop, and maintain an **ontology** (i.e., schemas, thesauruses, vocabularies, key word lists, and taxonomies) that best reflects the community understanding of the visible data assets.	Rationale: Question will elicit the data survivability capability of the program and the consumers experience as a result.	Requires evaluator interaction and **COI** participation	Fits into Ontology requirement
I. G. 01. F	Provide Data Management	Is there sufficient management of all of the data available through your program to adequately maintain	Rationale: Question will elicit the data survivability capability of the program and the consumers experience as a result.	Requires evaluator interaction	Vague

		and improve your data assets within a changing environment?			
I. G. 02	Provide Data Management	Describe your processes for ensuring the usefulness and timely availability of all data assets associated with your program.	Rationale: Question will elicit the data survivability capability of the program and the consumers experience as a result.	Requires evaluator interaction	Vague
I. G. 03	Provide Data Management	Describe the various data survivability scenarios considered in your program.	Rationale: Question will elicit the data survivability capability of the program and the consumers experience as a result.	Requires evaluator interaction	Vague
I. H. 01	Be Responsive to User Needs	Are perspectives of users, whether data consumers or data producers, incorporated into data approaches via continual feedback to ensure satisfaction?	Rationale: This question helps determine if the program is putting in place appropriate mechanisms to enable responsiveness to user data and application needs.	Requires evaluator interaction	Vague
I. H. 02	Be Responsive to User Needs	What tools, services, processes, and resources is the program providing to facilitate user feedback and program responsiveness with respect to data needs?	Rationale: This question helps determine if the program is putting in place appropriate mechanisms to enable responsiveness to user data and application needs.	Requires evaluator interaction	Vague

I. H. 03	Be Responsive to User Needs	What metrics are being used to determine responsiveness to user data needs?	Rationale: This question helps determine the program's ability to measure its responsiveness to user data and application needs.	Requires evaluator interaction	Vague
I. H. 04	Be Responsive to User Needs	What is the degree of collaboration with respect to data that is enabled and is occurring among the user community (ies) and the program developers?	Rationale: This question helps assess the actual degree of visibility into ongoing user needs and the responsiveness and quality of interaction with respect to user data and application needs.	Answered if DoD Metadata Registry Used	Fits into the **COI** requirement
I. H. 05	Be Responsive to User Needs	What are measured/ assessed trends over time with respect to the programs responsiveness to user data needs and degree of satisfaction towards meeting those needs?	Rationale: This question helps determine the degree of program improvement in being responsive to user data and application needs over time.	Requires evaluator interaction	Vague
I. H. 06	Be Responsive to User Needs	What are the programs plans to enhance responsiveness to user data needs?	Rationale: This question helps determine potential for improving future responsiveness to user data and applications needs.	Requires evaluator interaction	Vague
I. I. 07	Ensure authorized users obtain reliable secure information	Describe the protection mechanisms for program data to ensure that undetected	Rationale: This question helps determine capability to perform in the face of	Answered as part of NESI Part 5 Security	Security

		compromises are contained and do not allow an adversary to access restricted or sensitive program data while still maintaining visibility to authorized users? [ref RCD 4.1 Confidentiality (attribute to be added to address this issue)]	adversarial disruption.		
I. I. 08	Ensure authorized users obtain reliable secure information	Describe the techniques that inhibit an adversary who has compromised a client or server from accessing all sensitive program data and services within the enterprise. [ref RCD 4.1 Confidentiality (attribute to be added to address this issue)]	Rationale: This question elicits the design techniques used to manage controlled sharing of sensitive data	Answered as part of NESI Part 5 Security	Security

Guidance

- G1382: Be associated with one or more **Communities of Interest** (**COIs**).
- G1383: Use a **registered namespace** in the XML Gallery in the **DoD Metadata Registry**.
- G1384: Review **XML Information Resources** in the **DoD Metadata Registry**, using those which can be reused.
- G1385: Identify **XML Information Resources** for registration in the XML Gallery of the **DoD Metadata Registry**.
- G1386: Review predefined commonly used **data elements** in the **Data Element Gallery** of the **DoD Metadata Registry**, using those in the **relational database** technology which can be reused in the Program.

- G1387: Identify **data elements** created during Program development for registering in the **Data Element Gallery** of the **DoD MetaData Registry**.
- G1388: Use predefined commonly used database tables in the **DoD Metadata Registry**.
- G1389: Publish database tables which are of common interest by registering them in the **Reference Data Set** Gallery of the **DoD Metadata Registry**.
- G1390: Standardize on the terminology published by relevant **COIs** listed in the **Taxonomy Gallery** of the **DoD Metadata Registry**.
- G1392: Adhere to a common mechanism of service location.

References

- R1020: Net-Centric Checklist, July 30, 2004, Version 2.1.4

P1049: Metadata

Services and **data** to be mediated should always be formally defined, and typically this is done with some form of computer readable **metadata**.

NESI currently requires metadata, defined primarily as **XML Schema** and **Web Services Description Language** (**WSDL**) documents, be registered in the **DoD Metadata Registry**. NESI further specifies rules system developers must follow in developing XML Schema, including the requirement to search the registry for existing schemas that can be reused, aligning new schemas as closely as possible to existing similar schemas, reviewing schemas with the DoD XML Namespace Manager, and looking for other relevant Government and industry schemas that could be leveraged. The purpose is to avoid unnecessary duplication of effort and improve the success of future interoperability through common definitions.

Challenges with Centralized Management of Metadata

The NCES Data Strategy team, including the maintainers of the DoD Metadata Registry, strives to create a common data model, per **Community of Interest** (**COI**); but recognizing the difficulty in accomplishing that goal the team promotes the use of "mediation" from one schema to another. NCES currently implements mediation simply through the use of eXtensible Style Language Transformations (**XSLT**) to transform **XML documents** from one schema to another.

This focus on centrally managed data models is not viable as a long term solution to mediation since it requires substantial effort to define accurate transformations, and the underlying "business objects" almost always lose information in the process. The vision of a non-redundant object model is considered by most experts as unachievable due to social and communications barriers among the hundreds of organizations working as part of or with the Federal Government and the DoD in particular.

Accepting the fact that use of the DoD Metadata Registry is a requirement gives rise to posing the question should there be a new **FORCEnet** COI "**namespace**," or should the FORCEnet activities simply try to find suitable existing namespaces in which to register their metadata. Clearly, some FORCEnet applications will be able to leverage some of the existing schemas. But are there a significant number of new schemas to be registered, and if so can they be aligned to existing COI namespaces or will there be unacceptable barriers to introducing the changes required.

Moreover, the technologies for application and system development continue to improve to allow more rapid turnaround of new software capabilities, and in fact software developers are finding less of a need to work at the XML document level at all. **Model Driven Architecture** (**MDA**) technology, for example, is becoming mainstream, and **interfaces** are being developed visually, with the schemas automatically generated according to the graphical model. The creation of interfaces and schemas is becoming more of a dynamic activity, and the projected ad hoc interoperability of **loosely coupled** components, enforced by the FORCEnet vision, will mean bureaucratic processes such as those introduced by the DoD Metadata Registry may introduce significant risk.

Advancing Mediation with Semantic Descriptions

Striving to minimize the number of schema variations by leveraging common schemas across applications is laudable and should be encouraged. However, more advanced solutions to mediation are critical to the interoperability problem where common schemas do not exist. This may require a more dynamic process for registering metadata, without restrictions. An argument can be made for a FORCEnet COI in this regard.

As promoted by the NCES Data Strategy team, XSLT is the common practice for mediation. However, XSLT only solves a single point-to-point integration, and it is limited in its ability to support semantic validation. The **Business Process Execution Language** (**BPEL**) is an emerging specification (likely to become a standard) for defining specific interactions among services using documents defined through schema. It can use XSLT and other technologies to perform transformation of data elements, and semantics are implicit through their use. However, each BPEL definition is limited even further to a single **use-case** for the data.

In order to reduce the work and the errors associated with mediation, we need to take the concept to the next logical step. Documents and services should include metadata that encodes their semantic intent. Technologies are emerging, such as the **Web Ontology Language** (**OWL**) [http://w3.org/], that assist in defining the semantic relationships and constraints in schemas.

These definitions can be used to automate the transformations between applications and services, to validate the transformations, and to support much more intelligent human-computer interaction. For example, a PEO C4I and Space sponsored program developed the Service Mediation Description specification for the DISA Net-Centric Capabilities Pilot. This metadata document automatically generated user interfaces (input forms, data result tables, and map overlays) from semantically-described **Web services** and schemas, using a document format—derived from BPEL and other Web standards.

Best Practices

- BP1408: Use a **semantic** description language such as **Web Ontology Language** (**OWL**) or **Resource Definition Framework** (**RDF**) to represent an **Ontology**.

- BP1409: Register **Web services** using **Web Services Description Language** (**WSDL**) and **Universal Description, Discovery, and Integration** (**UDDI**).

References

- R1046: "The Semantic Web," Michael C, Daconata, Leo J. Obrst, Kevin T. Smith; Wiley Publishing Inc., 2003
- R1047: http://members.optusnet.com.au/~webindexing/Webbook2Ed/glossary.htm
- R1048: http://www.w3.org/TR/2004/REC-owl-features-20040210/
- R1049: http://www.w3.org/TR/2002/NOTE-wscl10-20020314/
- R1050: Wikipedia

P1065: Application Security

In the post-9/11 period, security has taken top priority in the nation's agenda. The terrorist has made America painfully aware of the consequences of inadequate security. As a result, billions of dollars along with numerous resources have been allocated to homeland security. It is more critical than ever to establish security guidelines for new and evolving Military applications.

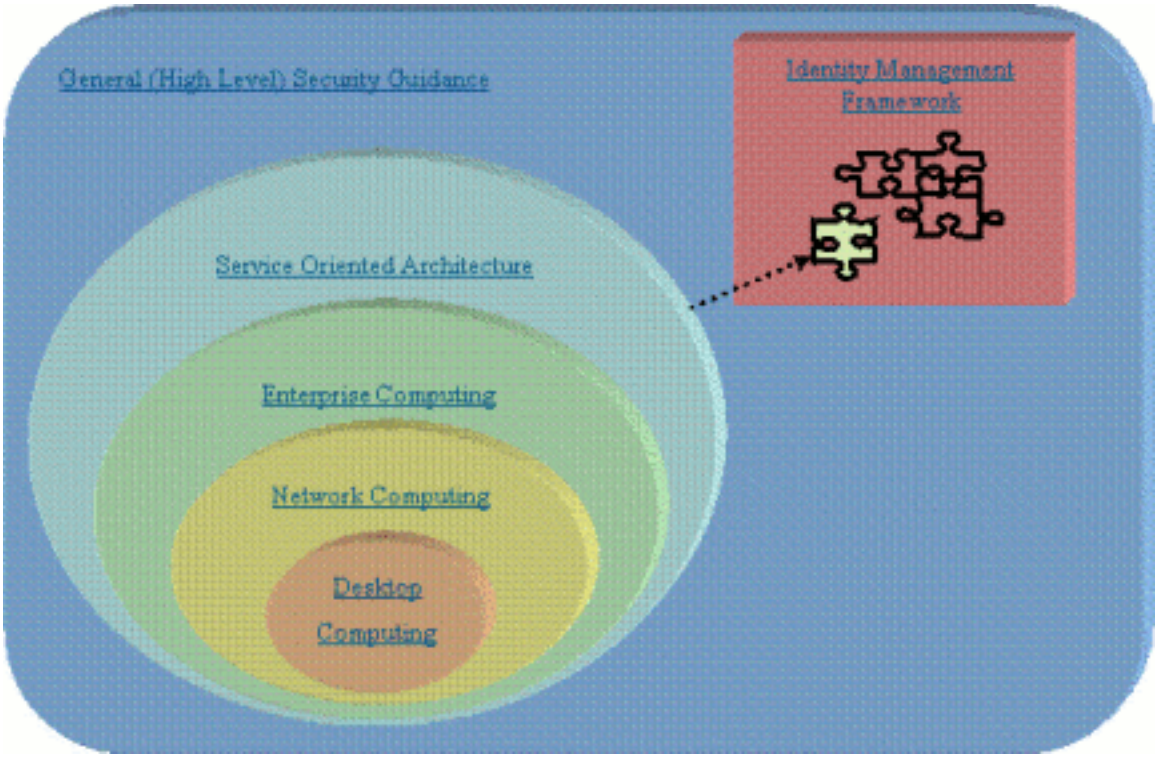
In general, there are two security aspects to consider for any application. The obvious one is the application itself; the other is security of the application deployment platform. NESI guidance focuses on the former as it would be a monumental (if not impossible) task to cover security for the various operating systems, application servers, database servers, etc., in use today.

Security is an enormous topic and one that is pervasive throughout all application models. Even though it would be convenient to have a single document that covers all security concerns, it simply is not possible. Security is an evolving process that should evolve with the application lifecycle. The approach of this document is first to cover general security guidance that will be applicable to all application types. After covering the general security guidance, this document will cover guidance that is specific to an application type. The coverage will be one of increasing application scale, starting with desktop applications and finishing with a look at how future net-centric application will integrate and interoperate with the DoD Identity Management Framework.

NESI application security guidance is applicable to applications at any stage of the development lifecycle. However, even if a software application adheres to all recommended guidance, there are no guarantees that the application will be secure. At best security is a moving target and an evolving process. In fact, a cottage industry of software applications grew out of the fact that software can not be trusted. As grim as it sounds, it does not mean that secure software is unachievable. Software can be designed and developed in such a way that it would be virtually impossible for attackers using current day resources. Following and applying NESI-recommended guidelines can be a good first step toward securing an application. Perfirming software compliance reviews throughout the lifecycle of a software application helps to insure software integrity.

The following diagram represents how secruity implementation at all levels supports application security in a net-centric, interoperable implementation environment:

- Desktop Computing
- Network Computing
- Enterprise Computing
- **Service-Oriented Architecture** [NESI SOA guidance is under development]
- General Application Security

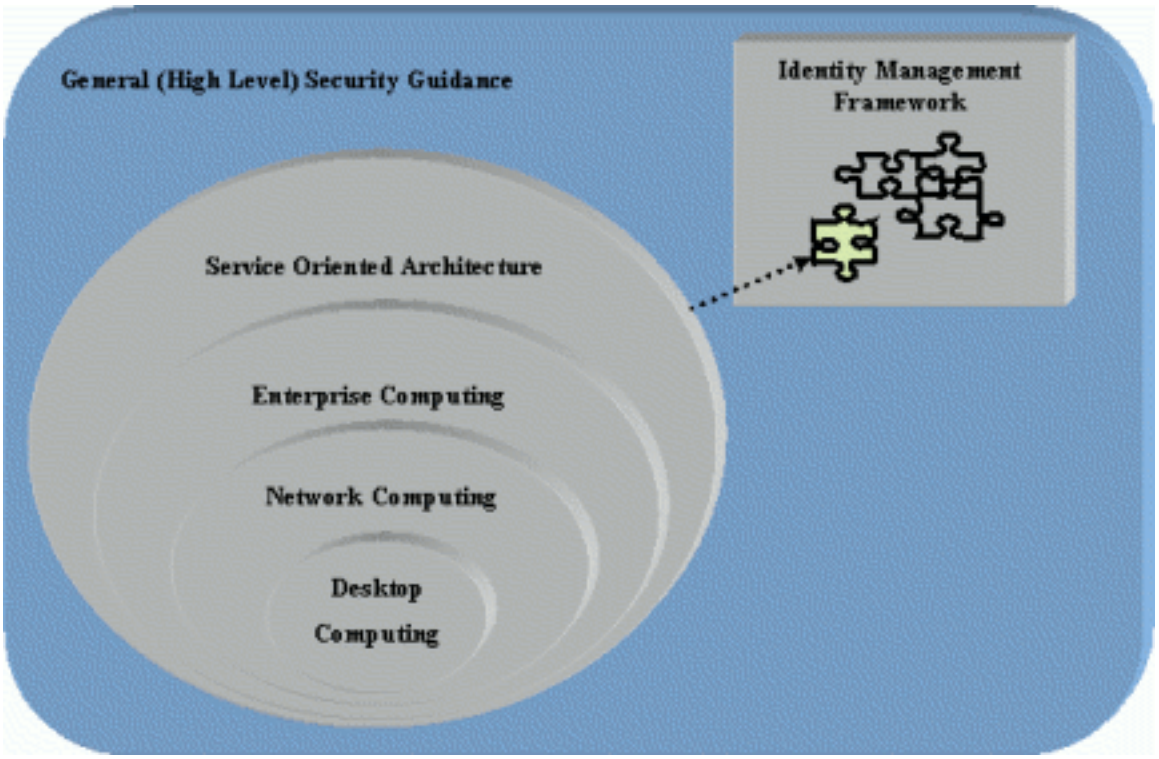

General (High Level) Security Guidance
Identity Management Framework
Service Oriented Architecture
Enterprise Computing
Network Computing
Desktop Computing

P1029: General Application Security

This perspective addresses high level guidance relevant to all application types and includes critical and common security infrastructure components. Related perspectives address application-specific guidance in terms of the Desktop/Network/Enterprise/**Service-Oriented Architecture** model illustrated in the diagram included in this perspective.

> ***Note:*** *A NESI Service-Oriented Architecture Perspective with related Guidance and Best Practices is under development.*

Some of the guidance in this perspective may not appear to be directly related to security; however, this guidance is important in ensuring the quality of code to prevent attackers from taking advantage of coding mistakes. Keep in mind there are no silver bullets with software security; scrutinize and test all aspects of an application to ensure the user and the application are protected.



Security **infrastructures** are fundamental building blocks that are common for all applications. The technologies in the Detailed Perspective list below have evolved into industry standards. Although no technology can be considered 100% secure, these technologies can provide a layer of protection that contribute to the overall security of the application.

Detailed Perspectives

- Public Key Infrastructure (PKI) and PK Enable Applications
- Key Management
- Encryption Services
- Certificate Processing
- Security Assertion Markup Language (SAML)

Guidance

- G1300: Secure all **endpoints**.
- G1301: Practice layered security.
- G1302: Validate all inputs.
- G1304: Unit test all code.
- G1305: Ensure the separation of **encrypted** and unencrypted information.
- G1306: **Identify** and **authenticate** users of the application.
- G1307: Provide a security policy file.

P1061: Public Key Infrastructure (PKI) and PK Enable Applications

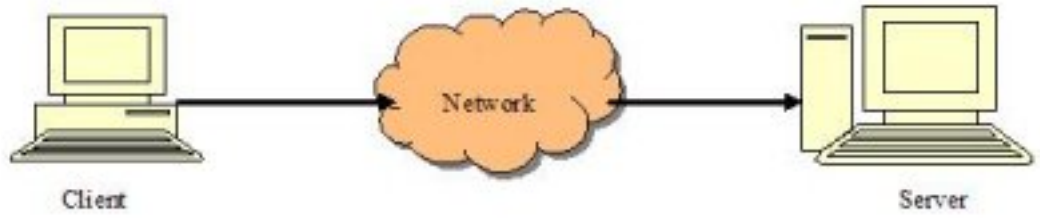
More and more secure **client/server** applications are appearing on the market. Applications today are relying heavily on **Digital Signature** technology to certify messages received were indeed sent by the sender. Both of these technologies use **Public Key encryption**, which is currently the only feasible way of implementing security over an insecure network such as the NIPRNET. Public Key encryption ensures that any form of communication that many contain sensitive information (i.e., passwords, credit card numbers) is protected while in transit and provides assurance to the receiver that the message was really sent by the sender. In the case of Web-based technologies, this is accomplished with a server that implements **encryption** at the communications level. The de facto standard for communication based encryption is the **Secure Sockets Layer** (**SSL**) protocol. The **infrastructure** used to support communication based encryption is **PKI** which is composed of a number of cryptographic technologies but provides for two key services, data integrity and confidentiality. **Public Key** systems involve a **Certificate Authority** (**CA**) responsible for issuing a pair of **encryption** keys: one public and one private. PKI systems typically rely upon the ability of the system to protect the **private key** from all but the intended user. If the private key can be copied or made public, then the authenticity of the transactions with the associated public key cannot be trusted. A CA creates, signs, and issues **Public Key Certificates**. The CA also posts **certificate** information to the directory and maintains a certificate revocation list (**CRL**).

A CA creates Public Key Certificates by interacting directly with users in the case of software certificates or by interacting with the Real Time Automated Personnel Identification System (RAPIDS) workstation via the Issuance Portal for Common Access Cards (CACs). CAs receives Public Keys from users or the RAPIDS workstation, add information about the user's **identity**, and format all of it into a Public Key Certificate. The CAs then signs the certificate. Consequently, the user can prove he or she is part of the PKI because the CA has signed his or her certificate, and the CA can prove it is part of the PKI because the root CA has signed its certificate.

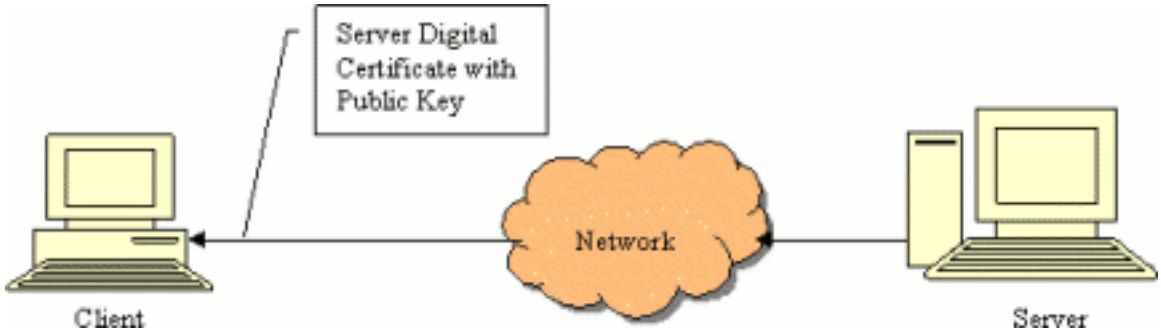
Digital Certificates are used to link a public key to an entity. The certificate contains information about the issuer of the certificate, the owner of the certificate, the Public Key contained in the certificate and a **digital signature**. Certificates authenticate the identity of owner because the digital signature is a message digest of all the information in the certificate. If the information was tampered with, the digital signature would be different and would not be able to be verified by the Certificate Authority.

To guarantee that data stays confidential and secure from attackers listening on the network in promiscuous mode (network sniffers), **Symmetric Encryption** (single key) is used to encrypt and decrypt the data. **Asymmetric Encryption** (public key- private key) is not used for all encryption because it is too expensive for high volume data. For **SSL**, Asymmetric Encryption is used initially to pass the **secret key** (often called the **session key**). Once the secret key has been established on both sides, all subsequent data communications can be performed using Symmetric Encryption. The entire SSL communications process is described as follows:

Note: *Step 1: Client Request Client sends the Server a "hello"* ***message****.*



Note: *Step 2: Server Response Server sends Client its certificate (including server's Public Key) as part of "hello" message.*



Note: *Step 3: Server requests Client certificate (this is an optional step).*

Note: *Step 4: Client validates Server certificate and replies with creation of session key and sends it encrypted using Server's Public Key.*

Note: *Step 5: Server decrypts data to obtain **Session Key**.*

Note: *Step 6: Client and Server communicate securely using **Symmetric Encryption** with the **Session Key**. **SSL** channel is now established.*

There are at least two options when an application needs to support **PKI**/**SSL**: use a module approved by **JITC** or develop the application abiding by the **DoD Class 3 Public Key Infrastructure Interface Specification**. The following guidances applies to **Public Key Enabled** applications wanting to operate within the DoD **PKI**.

Guidance

- G1308: Make applications handling unclassified medium value information in Moderately Protected Environments, unclassified high value information in Highly Protected Environments, and discretionary access control of classified information in Highly Protected Environments **Public Key Enabled** to interoperate with DoD **Class 3 PKI**.
- G1309: Make applications handling high value unclassified information in Minimally Protected environments **Public Key Enabled** to interoperate with DoD **Class 4 PKI**.
- G1310: Protect application cryptographic objects and functions from tampering.
- G1311: Use **LDAP**, **HTTP**, or **HTTPS** when applications communicate using DoD **PKI**.
- G1312: Make applications capable of being configured for use with DoD **PKI**.
- G1313: Provide documentation for application configuration and setup for use with DoD **PKI**.

P1041: Key Management

The key enabler in the **PKE** applications is **Asymmetric Encryption**, the use of **public** and **private keys**. It is used in exchanging session keys, and it is used to verify **Certificates** therefore, it is critical for applications to manage and protect the keys used in **PKI**. This includes the associated technologies used to store the keys and Certificates. The following list of guidance addresses key management issues.

Guidance

- G1314: Provide applications the ability to import and export keys (software certificates only).
- G1315: For applications, use key pairs and **Certificates** created for individuals using DoD **PKI** methods and procedures defined by the DoD Class 3 Public Key Infrastructure Interface Specification and the Personal Information Exchange Syntax Standard.
- G1316: Ensure that applications protect **private keys**.
- G1317: Ensure applications store **Certificates** for subscribers (the owner of the **Public Key** contained in the Certificate) when used in the context of signed and/or encrypted email.
- G1318: Develop applications such that they provide the capability to manage and store **trust points** (**Certificate Authority** Public Key **Certificates**).
- G1319: Ensure applications can recover data encrypted with legacy keys provided by the DoD **PKI** Key Recovery Manager (**KRM**).

P1020: Encryption Services

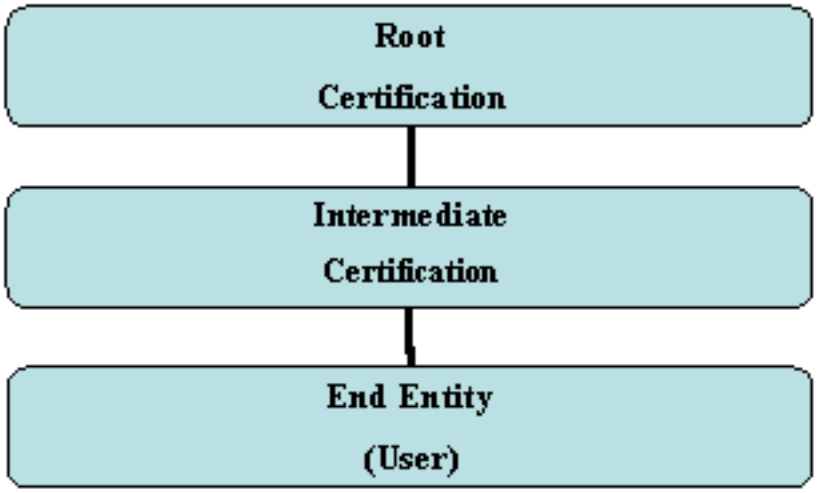
Successful implementation of **Public Key** enabled applications is predicated on the correct selection and use of security algorithms. This section provides guidance on the use of **encryption**, **digital signature**, and authentication services in a consistent manner to interoperate with DoD **PKI**.

Guidance

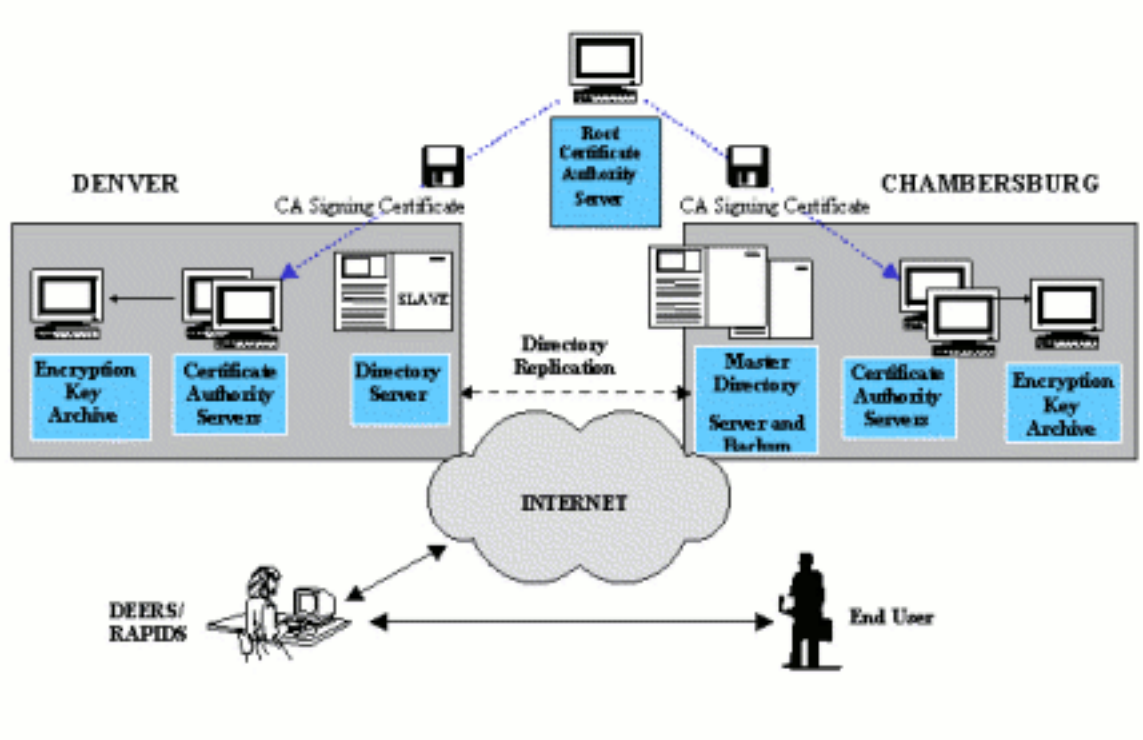
- G1320: Develop applications such that they use 128 bit **symmetric keys**, 1024 bit **asymmetric keys**.
- G1321: Enable applications to be capable of performing **Public Key** operations necessary to verify signatures on DoD **PKI** signed objects.
- G1322: Ensure that applications that interact with the DoD **PKI** using **SSL** (i.e., **HTTPS**) are capable of encrypting and decrypting data using the **Triple Data Encryption Algorithm** (**TDEA**).
- G1323: Generate random **symmetric encryption** keys when using symmetric encryption.
- G1324: Protect **symmetric keys** for the life of their use.
- G1325: Encrypt **symmetric keys** when not in use.
- G1326: Ensure applications are capable of producing Secure Hash Algorithm (**SHA**) digests of **messages** to support verification of DoD **PKI** signed objects.

P1009: Certificate Processing

The **DoD** implementation of the **Public Key Infrastructure** (**PKI**) is the framework and services that provide for the generation, distribution, control, tracking and destruction of **Public Key Certificates**. The purpose of a PKI is to manage keys and **Certificates** in a way whereby the DoD can maintain a trustworthy networking environment. Digital Certificates are issued by a DoD **Certificate Authority**. It is an electronic document that contains a user's **identity**, a pubic key, a validity period, and the issuing authority. It is digitally signed and the Certificate is chained hierarchically in a path that can be traced to the Root Certificate.



Certificates can be sent via email or more commonly retrieved from repositories (**Directory Server**). Applications must validate the Certificate by checking status of the Certificate. There are two forms of status checking, the legacy Certificate Revocation List (**CRL**) or **Online Certificate Status Protocol** (**OCSP**). The status check determines whether a Certificate is revoked. A Certificate can be revoked if the information in the Certificate may have changed (relocation, new email) or the Certificate has been compromised. The Certificate validation is a critical part of the PKI process; it is the application's responsibility to perform the status checks. The following guidance sets the guidelines for the Certificate processing.



Guidance

- G1327: Enable an application to request and obtain new **Certificates** for subscribers.
- G1328: Enable an application to retrieve **Certificates** for use, including relying party operations.
- G1330: Ensure applications are capable of checking the status of **Certificates** using a **Certificate Revocation List** (**CRL**) if not able to use the **Online Certificate Status Protocol** (**OCSP**).
- G1331: Ensure applications are able to check the status of a Certificate using the **Online Certificate Status Protocol** (**OCSP**).
- G1333: Only use a **Certificate** during the Certificate's validity range, as bounded by the Certificate's "Validity - Not Before" and "Validity - Not After" date fields.
- G1335: Make applications capable of being configured to operate only with PKI Certificate Authorities specifically approved by the application's owner/managing entity.
- G1338: Applications and **Certificates** need to be able to support multiple organizational units.

P1189: Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) is a vendor-neutral protocol specification for software applications and services to exchange security information in a distributed network environment. The SAML specification, maintained by the **OASIS** Security Services Technical Committee, defines schemas for how security assertions are structured and embedded within transport protocols.
SAML defines three types of assertions for an individual or machine:

Authentication	used for proving identity
Authorization	used for controlling access
Attributes	used to provide additional details to constrain the request

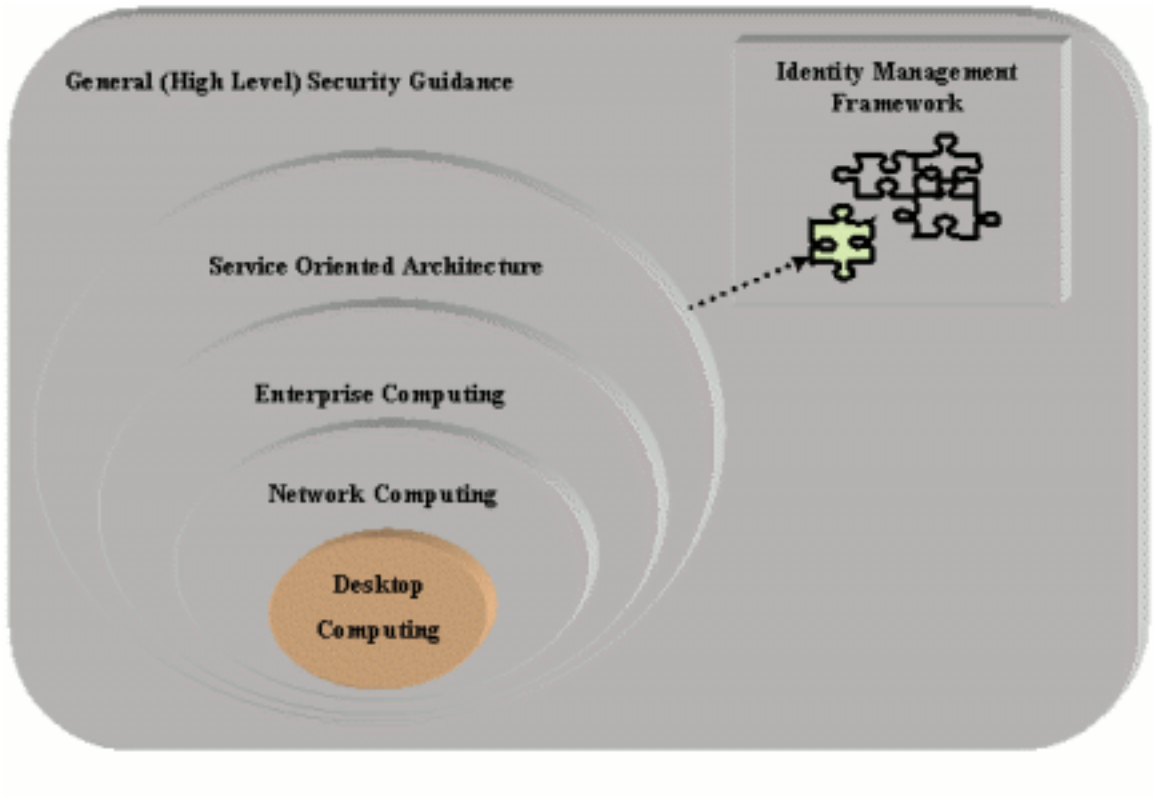
Email address, employee number, and rank are examples of attribute assertions.
SAML does not define any implementation of the services that authenticate or authorize users. Commercial vendors provide implementations in the form of authentication servers to authenticate and authorize users. Authentication servers respond to SAML requests and return SAML assertions that ensure the subject is logged in and authorized to access the resource.

Guidance

- G1379: Use **SAML** version 2.0 for representing security assertions.
- G1380: Use the **XACML** 2.0 standard for **SAML**-based rule engines.

P1018: Desktop Computing

Security is pervasive at all levels of computing. In the early days of computing, characterized by individual desktop computers with single users, security concerns were minimal compared to modern day systems. The user's main concerns were that the application did not crash and the data was safe. The Desktop Computing concept includes high level guidance that apply to applications on a generic level. NESI guidance deals with language and development issues such as protection of memory resources and protection of data (binary proprietary). This guidance also addresses application planning (i.e., a security policy plan) and application testing as it relates to security.



Desktop application security often does not get the attention that it should. First, most desktop applications are legacy applications that often did not consider security as part of the design. Second, most desktop applications are not network-based applications so security was not a primary concern. However, today's legacy applications quite often become tomorrow's net-centric **Web services**. Therefore, it is very important to evaluate and address security concerns of desktop applications not only during development but also in porting or migration efforts.

Detailed Perspectives

- API Security
- Java Security
- Application Resource Security

P1004: API Security

At the very fundamental level, applications are composed of calls to various **Application Programming Interfaces** (**APIs**) or **component** libraries. Develop APIs and component libraries with an ability to safeguard system resources and application reliability. It is important secure APIs and component libraries because these are often reused in multiple applications. A mistake in security could open up multiple applications to attacks. The guidance that follows provides some general API guidance that is independent of language or platform.

Guidance

- G1339: Practice defensive programming by checking all method arguments.
- G1340: Log all exceptional conditions.

P1038: Java Security

Java is an **Object Oriented Language**; applications benefit from the encapsulation features which offers protection for application data. Java was also designed and built with security in mind. Some of the security features include restricting direct access to memory (protecting data access privileges), array bounds checking (buffer overflow), and ability to install a security manager to protect system resources. Despite all the security features built into the Java language, it does not mean that Java **APIs** are immune to security problems. Take care in the design and implementation of APIs to prevent attacks. The following security guidance are targeted to Java-specific APIs.

Guidance

- G1341: Use a security manager support to restrict application access to privileged system resources.
- G1342: Restrict direct access to class internal variables to functions or methods of the class itself.
- G1343: Declare classes final to stop inheritance and prevent methods from being overridden.

P1005: Application Resource Security

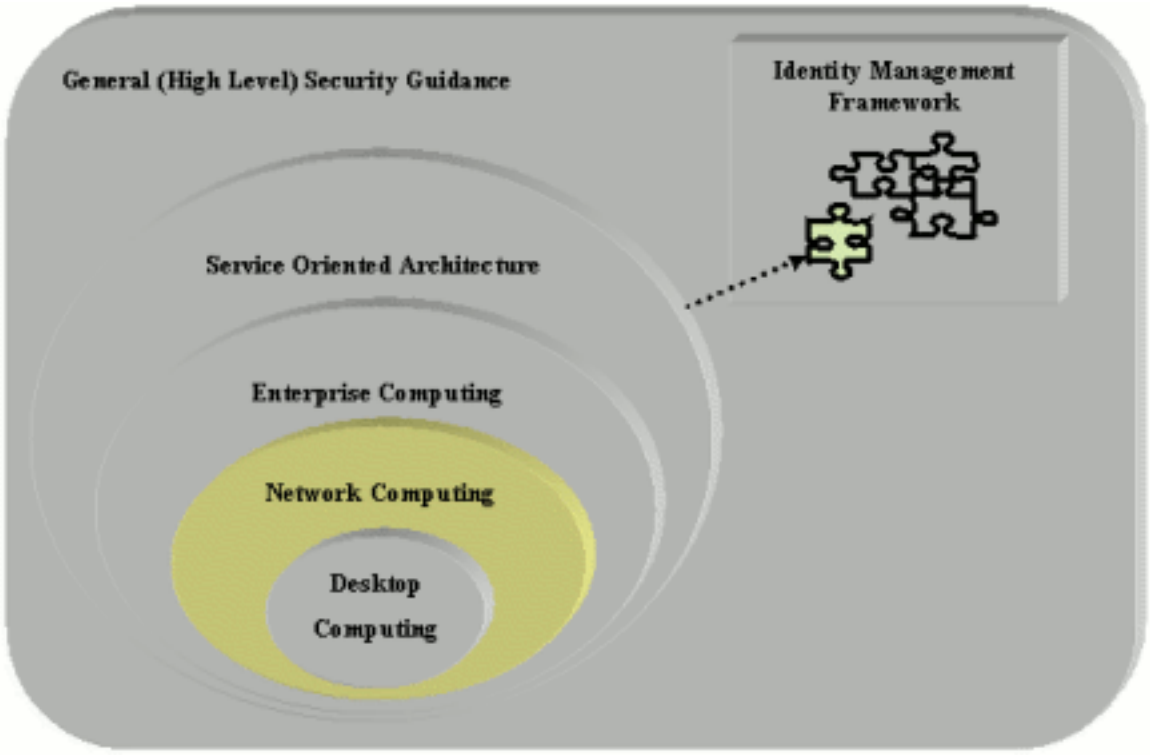
Applications use and store a large amount of data that often do not go into databases. For instance, an application often uses configuration files for application configuration, preferences files for personalization information (custom user experience) and resource files for internationalization support. Apply appropriate protection to sensitive resources to prevent attackers from tampering. Application bundles, properties files, configuration files when tampered could cause the user to execute inappropriate commands, expose sensitive data due to invalid configuration or cause the application to be inoperable. Therefore, it is of utmost importance to take appropriate measures to protect these resources.

Guidance

- G1344: Encrypt sensitive data stored in configuration or resource files.

P1053: Network Computing

As the migration from the desktop application model to a distributed application model (network) occurred, **Transmission Control Protocol/Internet Protocol** (**TCP/IP**) won the "protocol wars" and eventually dominated the local networked application space. The complexity of distributed architectures and an industry trend toward **Object Oriented Language** led to the advancement of **component**-based architectures. The need for component architectures was obvious because it was easier to divide a complex application into components and allow different teams of developers to work on individual components in parallel. Another added benefit was code reuse. A key security question was how to secure distributed components. In the early days, Applications typically created proprietary binary protocols for packet level communication on the **local area network** (**LAN**); therefore, intimate knowledge about the protocol and packet structure was needed to break into the system. However, this made it difficult to integrate systems because of the differences in network byte ordering of data. To solve the heterogeneous network problem and simplify system integration, a myriad of interface type network protocols such as **Remote Procedure Calls** (**RPC**), **Common Object Request Broker Architecture** (**CORBA**), and **Remote Method Invocation** (**RMI**) were invented (early incarnations of a **service-oriented architecture** or **SOA**). Each technology had its own merits and faults and none of these technologies dominated the market. The security concerns at this point were securing communications and limiting access to network data sources (database). The NESI Network Computing complex perspective encompasses the group of guidance that supports secure communications typically done through the use of **Secure Sockets Layer** (**SSL**) and **Public Key Infrastructure** (**PKI**) in a networked enterprise or SOA environment..



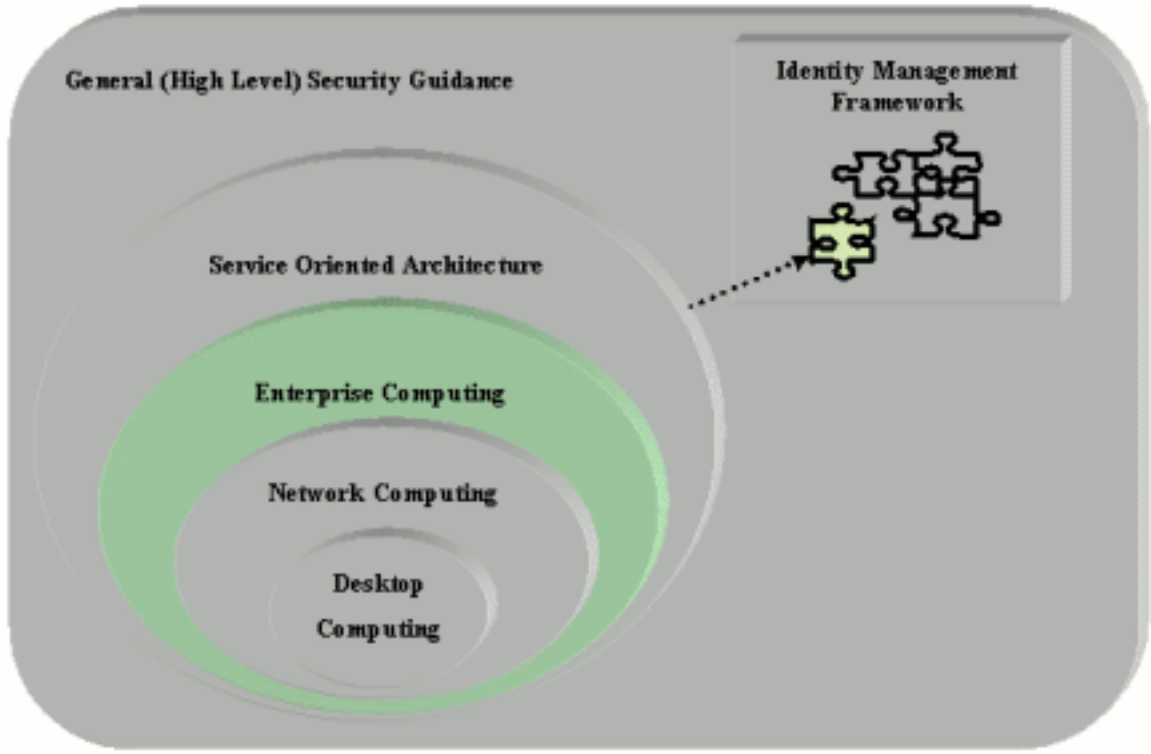
Detailed Perspectives

- Enterprise Computing
- Java Naming and Directory Interface (JNDI)
- Data Tier Security
- XML Web Services

P1021: Enterprise Computing

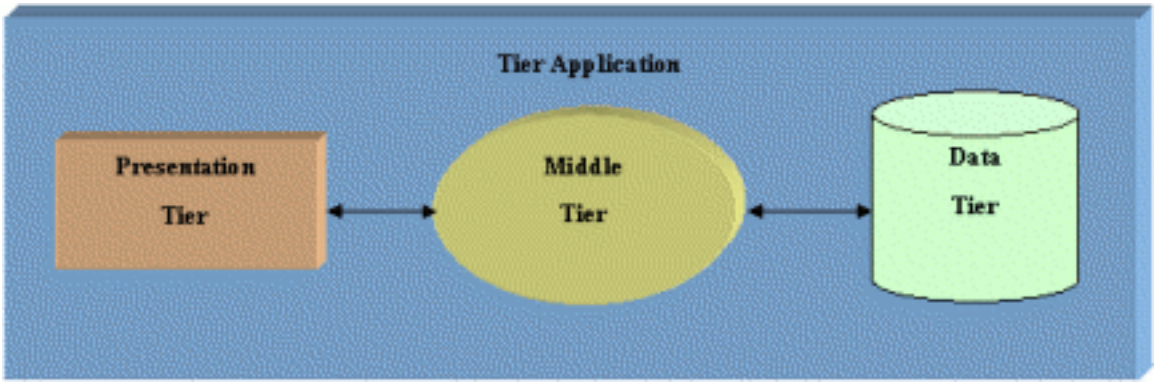
Enterprise computing existed long before the emergence of the **World Wide Web**. The Web simply facilitated extending the **Enterprise** to the World. The Web provided a ubiquitous protocol (**HTTP**) and interface for accessing network resources. Securing an enterprise application, however, provides a number of challenges. First, by virtue of being a **Web application**, it means the application must support multiple simultaneous users. Second, an enterprise Web application usually consists of a number of moving parts (**components**) on multiple computers. For instance, a Web application typically employs tier architecture (i.e., presentation, middle, and data) in which a complex group of servers and components work together to generate a response to the user. Addressing the security concerns in the same order, user management security requires guidance that assures the user's trust in the Web application and ensures protection of the customer data. **Public Key Infrastructure** (**PKI**) **Certificates** authenticate the Servers and Users through a **Certificate Authority**. **HTTPS** (**HTTP** over **SSL**) ensures encryption of communication data. Second, to address tier application architecture security concerns requires looking at component security in each of the architecture tiers. For the presentation tier, NESI guidance looks at security guidance in relations to user interaction (cross site scripting), form data processing and validating input. For middle tier security guidance addresses declarative security through deployment descriptors, **JNDI**, and programmatic security. Data tier security guidance involves securing user access to the **relational database management system** (**RDBMS**). There is also guidance on the **structured query language** (**SQL**) protocol that databases process and the API (i.e., **JDBC** or **ODBC**) that provides database-agnostic access to the data tier.

In general, component security within an enterprise presents less risk than components that are available outside the enterprise.



Addressing security concerns from the standpoint of an evolving software application, software requirements and software complexity will continue to grow. The complexities of today's enterprise software make it difficult to develop custom monolithic applications. Today's enterprise application must support multiple users using the application concurrently. It must be portable and interoperate with various standard and custom enterprise services through industry standard interfaces. To meet that demands, most enterprise application will rely on an architecture that is flexible, reusable, maintainable and interoperable. That application architecture model is the Tier Application architecture.

What is the Tier Application Architecture? Simply put, the Tiered Application Architecture takes an application and breaks it up into functional units, so call Tiers. A Tier is defined as a piece of software that provides part of the functionality for a complete application. The following diagram shows the general model of a three Tier application Architecture.



Three Tier Application Model

Even though an Enterprise Application can compose of N-Tiers, NESI uses a general three tier model to address the security concerns for the Enterprise application. The Presentation Tier is typically used to display the user interface and the application data. The Middle Tier provides the application logic and how data should be validated and processed. The Data Tier provides permanent store for the application data. The benefits of this model are interoperability, lower cost of maintenance, and interchangeability. This section will address the security guidance in accordance to the generalized three tier architecture. Starting from the Data Tier, to the Middle tier and finally to the Presentation Tier. The coverage of each tier may involve more than one applicable technology or platform which will have additional perspective and guidance specific to the topic.

Detailed Perspectives

- JNDI Security
- Data Tier Security
- RDBMS Security
- LDAP Security

P1039: JNDI Security

The **Java Naming and Directory Interface** (**JNDI**) is an **API** for directory services in a **Java EE** environment. It allows **clients** to discover and look up data and objects using a name. JNDI is portable and independent of the actual implementation. Additionally, it specifies a **service provider** interface (SPI) that allows plugging **directory service** implementations into the framework. The JNDI service implementations are hidden from the user and may make use of a **server**, a flat file, or a database. The choice is up to the JNDI provider.

Guidance

- G1071: Use vendor-neutral interface connections to the enterprise (e.g., **LDAP**, **JNDI**, **JMS**, databases).
- G1079: Isolate tailorable data values into the **deployment descriptors** for **Java EE** applications.
- G1239: Use design patterns (e.g., **facade**, **proxy**, or **adapter**) or property files to isolate vendor-specifics of vendor-dependent connections to the enterprise.

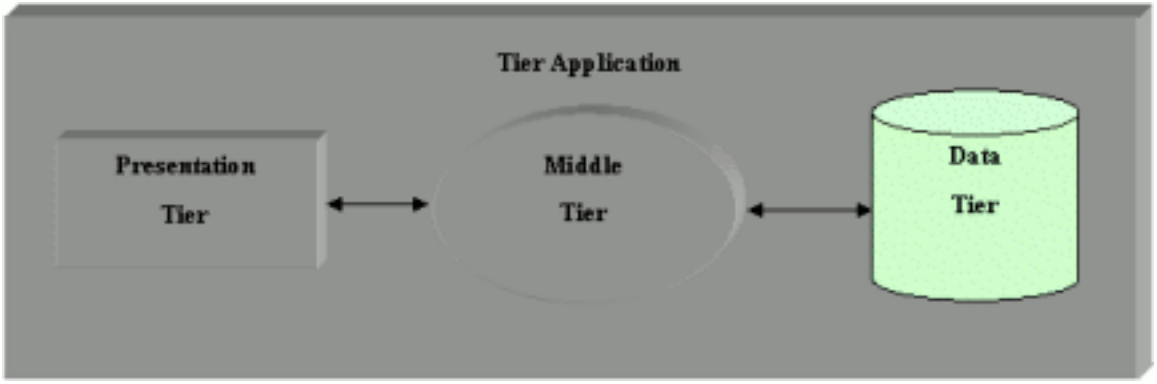
Best Practices

- BP1116: If using **Java**-based messaging (e.g., **JMS**), register destinations in **Java Naming and Directory Interface** (**JNDI**) so **message clients** can use JNDI to look up these destinations.

Examples

```
// Step 1
// Create a hashtable that contains the parameters
// used to initialize JNDI.
Hashtable contextParams = new Hashtable();
// Step 2
// Specify the context factory to use. The context
// factory is provided by the
// implementation.
contextParams.put( Context.INITIAL_CONTEXT_FACTORY, "com.jnidprovider.ContextFactory");
// Step 3
// The next parameter is the URL specifying the location
// of the JNDI provider's data store
contextParams.put( Context.PROVIDER_URL, "http://jndiprovider-database");
// Step 4
// Create the JNDI provider's context.
Context navyCurrentContext= new InitialContext ( contextParams );
// Step 5
// Look up the desired bean using its full name.
Object reference= navyCurrentContext.lookup ( "mil.us.navy.NavyBean" );
// Step 6
// Cast the located bean to the desired type.
MyBean navyBean= (NavyBean) PortableRemoteObject.narrow ( reference );
```

P1016: Data Tier



Tier Application Model

In general, applications use two mechanism for persistent storage of data: **Relational Database Management System** (**RDBMS**) and **Lightweight Directory Access Protocol** (**LDAP**) server. Other more primitive and/or custom forms of persistent store exists but are not included in this perspective. In practice, custom formats are not portable and therefore not recommended; aspects of forms such as properties files and **XML** files are covered in other ares of NESI guidance (i.e., Application Resource Security). The umbrella guidance G1381 exists to cover all custom formats and solutions.

Typically, applications are insulated from direct access to the database. Instead, industry standard abstract interfaces provide backend data store access. The benefit of this approach is that it decouples the application from database specific details and therefore allows interchangeable data store implementations. Security guidance for these standard **APIs** (**JDBC** for **RDBMS** and **JNDI** for **LDAP**) are in the following perspectives.

Detailed Perspectives

- RDBMS Security
- LDAP Security

Guidance

- G1381: Encrypt all sensitive persistent data.

P1064: RDBMS Security

Relational Database Management Systems remain on top amidst emerging technologies such as **XML** and **Object-Oriented Database Management Systems**. The continued dominance of **relational databases** is unlikely to change in the near future. First, there is still a large amount of legacy data and legacy applications that rely on **RDBMS**. Second, RDBMS are continuing to evolve to integrate XML as a function of the database. RDBMS is a reliable and proven technology that will be here for the long run. This perspective provides guidance on how best to secure the database.

Guidance

- G1346: Audit database access.
- G1347: Secure remote connections to database.
- G1348: Log database **transactions**.
- G1349: Validate all input that will be part of any dynamically generated **SQL**.
- G1350: Implement a strong password policy for **RDBMS**.
- G1351: Enhance database security by using multiple user accounts with constraints.
- G1352: Use database clustering and redundant array of independent disks (RAID) for high availability of data.

Best Practices

- BP1355: Do not design the database around the requirements of an application.
- BP1353: Use a data abstraction layer between the RDBMS and application for externally-visible applications to prevent the disclosure of sensitive data.

P1042: LDAP Security

The **Lightweight Directory Access Protocol** (**LDAP**) can be thought of as a datastore. It is an open Internet standard produced by the **Internet Engineering Task Force** (**IETF**). LDAP is, like X.500, both an information model and a protocol for querying and manipulating it. The LDAP overall data and namespace model is essentially that of X.500. The major difference is that the LDAP protocol itself is designed to run directly over the **TCP/IP** stack, and it lacks some of the more esoteric DAP protocol functions. LDAP can store text, photos, **URLs**, pointers to whatever, binary data, and Public Key **Certificates**.

Guidance

- G1377: Use **LDAP** 3.0 or later to perform all connections to LDAP repositories.
- G1378: Encrypt communication with **LDAP** repositories.

P1085: XML Web Service Security

An XML Web Service is a way to describe a software application that exposes its interfaces as a set of services that produce and consume **SOAP** formatted **XML** messages. This service-oriented architecture (**SOA**) describes its capabilities and requirements in an XML-formatted **Web Services Description Language** (**WSDL**) file. A user can consume this WSDL file to learn about the **Web service** interfaces available within an SOA. A provider may publish its WSDL file to a **UDDI** registry so a user can dynamically discover and utilize the Web service.

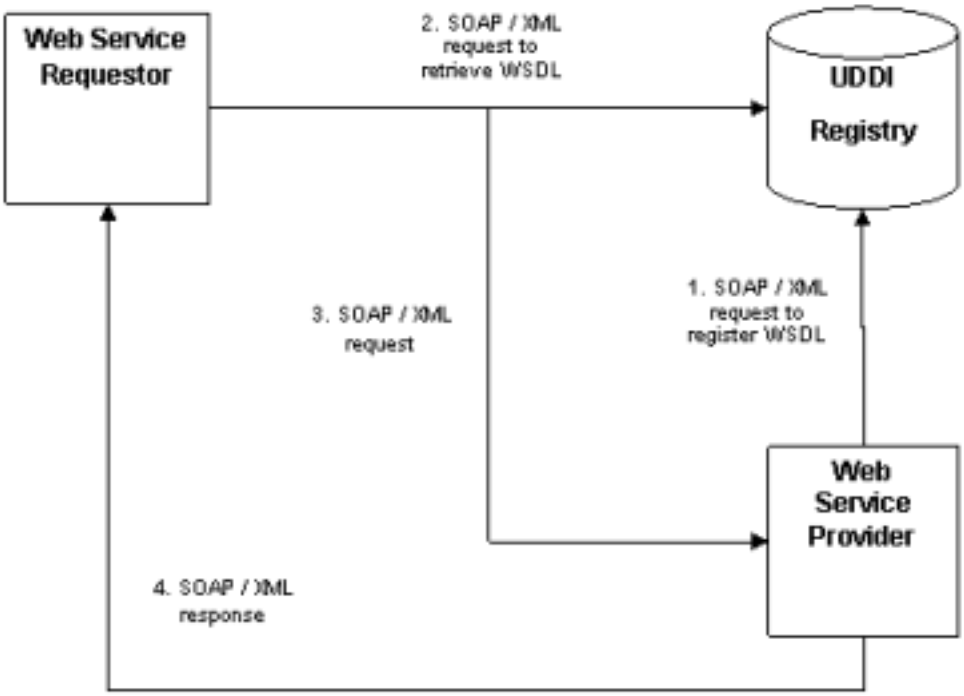


Figure 1

The drawing above depicts a typical implementation of a service-oriented architecture using XML Web Services. Several security challenges arise from this type of scenario including the following.

- **Authentication** (ensure that the sender of the **message** is genuine)
  - A hacker may try to spoof the **identity** of a Web service to gain access to a service.
  - A hacker may tamper with the WSDL file of a Web service provider in order to spoof an endpoint.
- **Integrity** (ensure that a message cannot be changed without detection by an unauthorized third party during transmission)
  - A hacker may intercept a message to or from a Web service provider and change its contents.
- Confidentiality (ensure that a message cannot be read by an unauthorized third party during transmission)
  - A hacker may intercept a message to or from a Web service provider and try to read the contents to obtain private information.

The XML Web services industry addresses these threats at the message level by incorporating existing technologies for challenging authentication, protecting integrity and ensuring confidentiality.

This message level security is based on the requirement that incoming **SOAP** formatted XML messages prove a set of claims made about the sender. These claims are cryptographically endorsed by an issuing authority and placed into the sender's message as security tokens. An X.509 certificate is just one example of a security token. The message is then encrypted and sent to the Web service provider who compares the claims of the incoming message with its security policy. If the claims are valid, the provider processes the message and sends a response.

The following defines the list of specifications in the XML Web Services space:

- WS-Security describes how to attach tokens, **digital signatures** and encrypted elements to a SOAP message. Tokens can be binary like X.509 or XML-based like **SAML**
    - XML Encryption
    - XML Signature
- WS-Trust describes how a message proves a set of claims (name, key, permission, etc.) and explains how to communicate with a token service to obtain a token
- WS-Policy describes how a Web service indicates its security requirements (required security tokens, supported encryption algorithms, etc.)
    - WS-SecurityPolicy
    - WS-PolicyAssertions
    - WS-PolicyAttachment

Guidance

- G1356: Use the **Simple Object Access Protocol** (**SOAP**) standard for all **Web services**.
- G1357: Do not rely on transport level security like **SSL** or **TLS**.
- G1359: Bind **SOAP Web service** security policy assertions to the service by expressing them in the associated **WSDL** file.
- G1361: Place the service provider **canonicalization** method inside the Web Services Description Language (**WSDL**) file.
- G1362: Use very intensive input validation (using a **schema**).
- G1363: Do not use clear text passwords.
- G1364: Hash all passwords using the combination of a timestamp, a **nonce** and the password for each **message** transmission.
- G1365: Specify a timeout value for all security tokens.
- G1366: Digitally sign all **messages**.
- G1367: Digitally sign **message** fragments that must not change during transport.
- G1368: Digitally sign any part of a **message** that is not **encrypted**.
- G1369: Digitally sign all requests made to a security token service.

- G1370: Digitally sign all **WSDL** files.
- G1371: Use the **Digital Signature Standard** for creating **Digital Signatures**.
- G1372: Use an X.509 **Certificate** to pass a **Public Key**.
- G1373: **Encrypt** all **messages** that cross an **IA** boundary.
- G1374: Individually **encrypt** sensitive **message** fragments intended for different intermediaries.
- G1376: Do not **encrypt** key elements that are needed for correct **SOAP** processing.

Best Practices

- BP1360: Use the **XML** Infoset standard to serialize messages.
- BP1375: Use Asymmetric Encryption.

References

- R1008: Web Services Security Specification, March 2004, (http://www.oasis-open.org/specs/index.php)
- R1009: An Introduction to the Web Services Architecture and Its Specifications, October 2004, (http://msdn.microsoft.com/webservices)
- R1010: XML (http://en.wikipedia.org/wiki/Xml)

P1113: Programming Languages

This Complex Perspectiive contains a collection of Detailed Perspectives which provide programming language guidance. The purpose of the following Perspectives is to provide language-specific guidance with the purpose of improving interoperability and net-centricity.

Detailed Perspectives

- C++
- VHDL

References

- R1150: "C++ Coding Standards, 101 Rules, Guidelines and Best Practices," Herb Sutter and Andrei Alexandrescu, Addison-Wesley, 2004. ISBN: 0-321-11358-6

P1090: C++

The development of software is a complex and difficult process that covers a wide range of activities starting at the earliest phases of requirements analysis all the way through the release of the software. In the **DoD**, many formal processes, documents and reviews need to occur before software is ready for release as a product. This complexity has increased as the accepted software development processes has evolved to embrace Object-Oriented techniques and incremental development.

A number of individuals, institutions, companies and products have attempted to solve software development issues and have produced a number of very useful papers, dissertations and books. It is not the intent of this NESI perspective to re-state written material or to endorse any particular institution, corporation or product. This perspective highlights those practices relating to the use of the C++ language which have demonstrated an ability to increase interoperability and enable net-centricity. In particular, one goal of this perspective is to identify guidance and best practices which facilitate interoperability of C++ code in order to promote reuse.

This perspective includes three sub-perspectives; much of the content is modeled after coding standards Herb Sutter and Andrei Alexandrescu put forth in the referenced text.

Detailed Perspectives

- C++ Header Files
- C++ Operator Overloading
- C++ Namesapces and Modules

P1114: C++ Operator Overloading

C++ allows for overloading of operators in order to change their implementation depending on the type of arguments provided. This can improve code clarity and serve as a short hand for developers. However, developers must be careful to not change the expected behavior or semantics of an operator in a way that provides unexpected behavior to developers using the code. Code which has clearly understood behavior has a better chance of being reusable.

Guidance

- G1775: Do not overload the logical **`AND`** operator.
- G1776: Do not overload the logical **`OR`** operator.
- G1777: Do not overload the **`comma`** operator.

Best Practices

- BP1780: Only overload arithmetic operators for objects that are arithmetic in nature.

P1115: C++ Namespaces and Modules

Namespaces and modules are abstract containers for related items. Often, software developers use both to isolate related items in order to promote reuse. Namespaces provide a context within which to define identifiers (i.e., classes, constants, variables, and functions). One advantage of namespaces is that they allow multiple identifiers with the same name to be used in the same code without name collisions.

Guidance

- G1778: Place all **`#include`** statements before all namespace **`using`** statements.
- G1779: Explicitly namespace-qualify all names in header files.

Best Practices

- BP1781: Allocate and de-allocate all module objects within the module that contains the objects.
- BP1782: Do not propagate exceptions across module boundaries.
- BP1783: Use portable types in a module's interface.

P1089: C++ Header Files

A header file in C++ describes the interface of the related implementation file. Header files serve as a communication mechanism to describe interfaces including data-types, **namespaces**, required resources, as well as serving as a source of reference documentation. The compiler uses header files during compilation, and humans use header files during software development. To promote reuse, header files need to be self-describing and developed such that compilation is straight forward and consistent from one compile to another.

Guidance

- G1773: Use `#include` guards for all headers.
- G1774: Make header files self-sufficient.
- G1779: Explicitly namespace-qualify all names in header files.

P1088: VHDL

The development of hardware described by software is a complex and difficult process that covers a wide range of activities: starting at the earliest phases of requirements analysis all the way through the fabrication of a functioning digital circuit. One language developed for describing digital circuits is **Very High Speed Integrated Circuit** (**VHSIC**) Hardware Description Language (**VHDL**).

In the **DoD**, there are many formal processes, documents and reviews which need to be done in order for the software code to be approved to be developed into a physical circuit. This complexity has been made more complicated in nature as modern chip designs have become increasingly large and intricate. There have been many articles and books written on these issues. It is not the intent of this perspective to re-state written material. It is the intent of this perspective to highlight those practices which have been demonstrated to increase interoperability and reuse of VHDL code.

Detailed Perspectives

- VHDL Coding and Design
- VHDL Synchronous Design
- VHDL Synthesizable Design
- VHDL Testbench

References

- R1113: "Circuit Design with VHDL" by Volnei A. Pedroni, MIT Press, 2004. ISBN: 0-262-16224-5
- R1114: "VHDL Coding Styles and Methodologies" (2nd Edition) by Ben Cohen, Kluwer Academic Publishers, 1999. ISBN: 0-7923-8474-1
- R1115: "JTRS JPEO Software Standards (Version 1.0)," SPAWAR Systems Center San Diego, 2006.

P1091: VHDL Coding and Design

There are coding and design decisions that are made during the lifecycle of a program or project which can have significant impact on interoperability and net-centricity. Many of these decisions directly relate to cohesiveness and coupling. Modifications to a project's code often create additional obstacles and decreases efficiency. The purpose of this perspective is to provide guidance and best practices to minimize these problems

Guidance

- G1717: Use constants instead of hard-coded numbers for characteristics that may change throughout the lifetime of the model.

Best Practices

- BP1720: Do not use commonly predefined VHDL identifier names for other identifiers.
- BP1721: Define a VHDL package for closely related VHDL items that support an application function.
- BP1722: Employ VHDL components for commonly used VHDL described circuits.

P1092: VHDL Synchronous Design

The engineers of digital integrated circuits (ICs) are very careful to make sure their designs are correct, for it is imperative that hardware designs are correct before being fabricated into physical circuits. However, digital circuits are not easily testable and real tests cannot be done on them until the circuit design has been finalized and physically produced. This is one of the reasons why the majority of today's digital designs are based on a synchronous design to improve the probability that the final produced chip will work by simplifying the process and using reliable techniques.

Guidance

- G1718: Design circuits to be synchronous.

P1093: VHDL Synthesizable Design

To be able physically to implement hardware described by software, the design must be synthesizable. Synthesis is a process where an abstract form of described circuit behavior (e.g., **VHDL** code) is mapped to an implementation in terms of logic gates (**`AND`**, **`OR`**, **`NOT`**, etc.). Logic synthesis is an essential part of digital electronic design automation and is often the step following code compilation and simulation.

Best Practices

- BP1723: Do not use guarded signals.

P1094: VHDL Testbench

A **VHDL testbench** is a **VHDL** component used to verify that a developing circuit design is functioning as planned. The testbench generates the stimulus to drive the unit under test under a variety of test conditions, verifies that it meets specifications, and reports all errors and warnings in a concise human readable format. The testbench is used during the simulation phase of digital electronic design automation.

Guidance

- G1719: Automate testbench error checking in VHDL development.

Guidance and Best Practice Details

G1001

Statement:

Define public **interfaces** in a formal standard.

Rationale:

It is important to use a common language to define the interfaces so producers and consumers can work independently and together.

There are many standards for defining interfaces (**UML**, **WSDL**, and **CORBA**). Use a documented standard that is widely accepted by industry.

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1001.1]

Do **UML** documents exist that describe the shared interfaces?

Procedure:

Ask for the design documents to be provided during the review process.

Example:

None

2) Test: [G1001.2]

Are there **WSDL** files that document the interface to Web services?

Procedure:

Look for the existence of **`.WSDL`** files.

Example:

None

3) Test: [G1001.3]

Are there **IDL** files that document the interfaces to **CORBA** services?

Procedure:

Look for the existence of **`.idl`** files.

Example:

None

G1002

Statement:

Separate public **interfaces** from implementation.

Rationale:

This guidance encourages clean separation between **interface** and implementation details for all types of application development. This allows components and systems to be **loosely coupled**. The flexibility allows groups of developers to work independently and in parallel to the contract defined by the interface.

Another benefit of hiding implementation details is that it allows the implementation to change without affecting users of the interface. This means the interface can support dynamic and pluggable implementation.

Justifies:

G1217

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1002.1]

`C++`: Check to make sure interfaces are defined as pure virtual functions.

Procedure:

Make sure **`C++`** classes are defined in header files. Classes that represent external interfaces should contain only pure virtual functions. Make sure the class does not declare non-constant data members. Also, make sure it does not define default implementation. An interface should provide no default behavior.

Example:

None

2) Test: [G1002.2]

`C`: Check to make sure functions are declared in a header file using prototypes.

Procedure:

Make sure each library function has a prototype declaration in the header file.

Example:

None

G1003

Statement:

Separate the contents of application libraries that are to be shared from libraries that are to be used internally.

Rationale:

The public libraries that are intended to be shared with outside consumers need to remain fairly static in order to facilitate independent development by the **consumer** and the **producer** of the libraries' functionality. The consumer and the producer should mutually agree to changes in libraries.

All library content should not have external dependencies that are not related to supporting the interface.

There must be clear separation between domain-specific and shared libraries. Libraries that will be used in joint or multiple projects should not have domain-specific code.

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1003.1]

Do the publicly shared libraries have any private or undocumented functionality?

Procedure:

Check each library against the publicly defined header and make sure that all objects or methods are public.

Example:

None

2) Test: [G1003.2]

Does the library contain extraneous interfaces or code that is not required?

Procedure:

Use coverage tool/Junit to make sure there is no extraneous code.

Example:

None

3) Test: [G1003.3]

Do the publicly shared libraries have any private or undocumented functionality?

Procedure:

Check to make sure that one library use of another library does not cross domain-specific boundaries. For instance, a common library of utilities should not have dependencies on another library that supports a specific such as UHF satellites. However, the reverse is okay.

Example:

None

G1004

Statement:

Make public **interfaces** backward-compatible within the constraints of a published **deprecation** policy.

Rationale:

The public interface is basically a contract between the **producer** of the functionality defined in an interface and the **consumer** of the functionality. This and related guidance statements are intended to ensure that this contract remains intact and that the consumer of the functionality is not broken during the update cycle of the interface.

Justifies:

G1019 G1208 G1214 G1018

Referenced By:

Versioning XML Schemas
Public Interface Design
Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1004.1]

Does the public interface (interfaces that are used externally, outside the project's domain) contain versioning information?

Procedure:

Check to make sure the interface/class has versioning information.

Example:

None

2) Test: [G1004.2]

Does the document structure contain a document that indicates the shelf life of deprecated interfaces?

Procedure:

Note: This is a mandatory document.

Check for project documents that have information on the life of deprecated interfaces.

Example:

None

G1005

Statement:

Separate infrastructure capabilities from **mission** functions.

Rationale:

Applications should not try to reinvent the wheel by creating custom **enterprise services** such as messaging, directory services, logging, etc. Application development should use standardized **APIs** to access common enterprise services. For instance, in Java, use **JMS** to access a messaging system.

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1005.1]

Does the application re-create common and available enterprise services?

Procedure:

Check the application code for code that recreates functionality of an enterprise service.

Example:

None

2) Test: [G1005.2]

Does the application code access enterprise services in a vendor-specific way?

Procedure:

Check for code that accesses a vendor-specific API instead of utilizing an industry-standard API.

Example:

None

G1007

Statement:

Ensure that applications use open, standardized, **vendor**-neutral **API**(s).

Rationale:

Using standardized, open APIs will enable the code to be more portable. It will also prevent vendor lock-in. "Standardized" means industry consensus. "Open" means available to everyone.

Justifies:

G1071

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1007.2]

Does the application utilize vendor-specific **API**s?

Procedure:

Check the application to make sure it is not using vendor-specific APIs. For instance, see if the application accesses the database using a proprietary interface from Oracle instead of the standard calls.

Example:

None

2) Test: [G1007.1]

Does the application create customized/proprietary solutions where standardized **API**s exists?

Procedure:

Check the application for code that has proprietary solutions where standardized APIs exists. For instance, does the application write its own messaging system, bypassing utilizing the API.

Example:

None

G1008

Statement:

Isolate platform-specific **interfaces** and **vendor** dependencies.

Rationale:

Insulating platform-specific code using standard abstractions or custom classes will keep all non-portable code in one place and prevent proliferation of non-portable code throughout the application.

Justifies:

G1073 G1118

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1008.1]

Does the application contain any platform-specific code that has not been abstracted?

Procedure:

Check code that is non-portable; for instance, the code does not use back slashes (Windows) or forward slashes (UNIX) in literal strings to create a path.

Example:

```
String path = "\tmp";
```

2) Test: [G1008.2]

Is platform-specific code isolated into a single class or file?

Procedure:

Search the files for platform-specific code.

Example:

None

G1010

Statement:

Use **open-standard** logging frameworks.

Rationale:

Standardizing on one logging **API** means the code will be more portable between developers, and developers no longer need to learn multiple logging frameworks.

Justifies:

G1210 G1209

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1010.1]

See sublevel guidance: G1209, G1210.

Procedure:

Example:

G1011

Statement:

Make components independently deployable.

Rationale:

Independently deployable components do not have any dependencies on other components. This is often unattainable because components are often aggregations of lower-level components. Exceptions to this rule can occur if the relationships between components are one or more of the following:

- well-defined and well thought out
- carefully managed
- externally configurable

Referenced By:

Implement a Component-Based Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1011.1]

Is the component dependent on other components?

Procedure:

Check for dependencies.

Example:

None

G1012

Statement:

Use a set of services to expose Component functionality.

Rationale:

By exposing discrete units of functionality as **services**, business and data integrity remain intact. A service receives a request, processes it, and returns the result to the requester as a single operation.

Referenced By:

Implement a Component-Based Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1012.1]

Are there **WAR** files that contain the component?

Procedure:

Check for the occurrence of **`.war`** files.

Example:

None.

2) Test: [G1012.2]

Are there **WSDL** files that define the services?

Procedure:

Check for the occurrence of **`.wsdl`** files.

Example:

None.

G1014

Statement:

Access databases through **open standard** interfaces.

Rationale:

The use of non-standard interfaces can cause portability issues. Standards-based database interfaces promote database independence. For example, **ODBC** is a standard database interface for referencing databases with C/C++ and .NET, while Java Database Connection (**JDBC**) is a standard **API** for accessing databases with Java.

Referenced By:

Decouple from Applications

Acquisition Phase:

Decouple from ApplicationsDevelopment

Evaluation Criteria:

1) Test: [G1014.1]

Are standard interfaces used to access databases?

Procedure:

Check that standards-based interfaces are used to access databases; for example, ODBC for C,C++, or .NET languages, or JDBC for Java.

Example:

G1018

Statement:

Assign identifiers to all public interfaces.

Rationale:

Assigning versions is necessary when determining compatibility between the **interface** and its **consumer**. Versioning public interfaces allows all parties to track the evolution of the interface for backward compatibility. This can help consumers plan for integration and migration. It is important to have the version information in the shared public interface code because it identifies the actual interface to which consumers of the interface will be coding. Another benefit is that it allows tools to generate the documentation automatically so it does not need to be in two places.

Derived From:

G1004

Justifies:

G1753

Referenced By:

Public Interface Design
Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1018.1]

Does the shared public interface code contain versioning information?

Procedure:

Inspect public interfaces or their supporting documentation for version identifiers.

Example:

G1019

Statement:

Deprecate public interfaces in accordance with a published deprecation policy.

Rationale:

By deprecating instead of removing interfaces, development teams can plan for software migration and continue to run the software with existing (but deprecated) interfaces.

Derived From:

G1004

Referenced By:

Versioning XML Schemas
Public Interface Design
Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1019.1]

Are public interfaces appropriately deprecated?

Procedure:

Check the project documentation for deprecation policy.

Check that interfaces are properly marked and removed according to the deprecation policy.

Example:

None

G1021

Statement:

Create fully insulated classes.

Rationale:

Data members should not be public.

Do not expose implementation details of a class. For instance, information such as the use of a link list or hashtablein a class should not be exposed (i.e., made public).

Making implementation details public creates interdependencies between the class and its users, subjecting the users to changes in implementation. Therefore, access should only occur via public interface methods. This makes the implementation more robust, because all data can be validated when assigned new values or the changes can be logged.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1021.1]

Do instance variables have public access or are they more accessible than necessary?

Procedure:

Check that the instance variable in classes does not have public access unless it is static and final.

Example:

None

2) Test: [G1021.2]

Does the class provide direct access to internal data via pass by reference?

Procedure:

Check to make sure that the methods that access the internal state do not return a reference to the internal data.

Example:

None

G1022

Statement:

Insulate public **interfaces** from compile-time dependencies.

Rationale:

There are three distinct advantages to separating interface from implementation:

- Multiple interested parties (**COIs**) can develop the interface and publish it to the user community ahead of any specific implementation. This allows groups to work independently and in parallel.
- It prevents multiple copies of the defining interface. Duplicating the code for the interface in each implementation (library, jar, and assembly) makes it difficult to maintain, especially as the interface evolves.
- It insulates developers from the constant changes in implementation.

Referenced By:

Public Interface Design
Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1022.1]

Is the packaging or deployment of the public interface self-contained and isolated to only the public interface(s)?

Procedure:

Check to make sure that the jar, library, assembly, and WSDL only contain the agreed-upon public interface (interfaces being shared externally).

Example:

None

2) Test: [G1022.2]

Does the container (jars, libraries, assemblies, WSDL) contain files other than the interface?

Procedure:

Check to make sure the library does not include or rely upon any other files such as resource files, properties files, configuration files, other libraries, XML files, and so on that would force the repackaging of the public interface.

Example:

None

3) Test: [G1022.3]

Are there any outside influences that could affect the packaging of the public interface?

Procedure:

Check the public interface for dependence on resource files, properties files, configuration files, XML files, and other libraries or packages.

Example:

None

G1027

Statement:

Internally document all source code developed with DoD funding.

Rationale:

Well-documented source code is easier to maintain and enhance over time. It is hard enough to get documentation about software and to keep it up to date. If the documentation is not internal to the source code, the chances that the software is current and up-to-date decreases. In recent years, the trend has been to generate external documentation about the software by processing the source code and comments (e.g., Javadoc).

In addition to documenting the functionality of the source code, it is important to capture the configuration control information (e.g., CVS).

Justifies:

G1751

Referenced By:

Standard Interface Documentation

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1027.1]

Do all the source code files have a header that includes a statement protecting government rights to the source code and the right to change the source code?

Procedure:

Scan each file and make sure the header includes a statement that protects the government's right to use, modify, and share the information with other government departments and agencies.

Example:

None

2) Test: [G1027.2]

Do all the source code files have a header that includes configuration information?

Procedure:

Scan each file and make sure the header also includes configuration management information such as author, date created, and a history of modifications and versions.

Example:

None

3) Test: [G1027.3]

Do all the source code files have internal documentation for attributes, methods that a computer process?

Procedure:

Scan the source files and make sure they are internally documented with tags such as Javadoc or XML tags.

Example:

None

G1030

Statement:

Use a standard GUI **component** library.

Rationale:

A predefined component library helps control cost and configuration. Licensing issues can be resolved before development begins, and component costs are minimized by avoiding library overlap.

Now that component architecture is standard, it is possible to put together applications using a variety of components from multiple vendors. These components are bundled in third-party toolkits that vastly extend the range of options available in standard Windows or Java GUI toolkits. These toolkits are in common use and possess a wide variety of pre-built components. Almost all support common **look-and-feel** (e.g., Windows or Java).

Referenced By:

Thick Clients

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1030.1]

Does the user interface code use any other toolkits besides a Standard GUI Toolkit?

Procedure:

Check to make sure the thick-client code is developed using the Swing/AWT library in Java, and the standard, included Windows Toolkit In .NET.

Example:

None

G1032

Statement:

Validate all input fields.

Rationale:

Detect errors as close to point-of-data-entry as possible. This greatly enhances the end-user experience and reduces frustration. This can be done by reducing the number of freeform text fields and using selection mechanisms such as radio buttons, option boxes, pull down lists, maps, calendars, clocks, slider bars, and other numeric validation entries.

Referenced By:

Presentation Tier
Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1032.1]

Do the GUI screens use non-freeform text entry fields?

Procedure:

Scan the GUI code looking for the use of non-freeform text data entry mechanisms.

Example:

None.

G1035

Statement:

Follow W3C standards for code which will generate a Web page display.

Rationale:

Code cannot be browser-independent if it uses vendor-specific add on features. Vendor-specific add-on features reduce the portability and **interoperability** of the code. Vendor-specific **API**(s) can cause vendor lock-in and in many cases can also cause version lock-in. Following the W3C standards avoids these problems.

Referenced By:

Browser-Based Clients

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1035.1]

Does the code adhere strictly to the W3C standards?

Procedure:

Check to make sure there is no vendor-specific code.

Example:

None

G1043

Statement:

Separate formatting from data through the use of **style sheets** instead of hard coded **HTML** attributes.

Rationale:

Formatting information will be located in one location instead of scattered throughout each individual Web page of a Web site. This makes a Web site more maintainable.

Referenced By:

Style Sheets
Browser-Based Clients

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1043.1]

Are any formatting attributes used in any of the HTML tags?

Procedure:

Search all web pages and make sure there are no formatting attributes such as align, color, font, or size in any tags.

Example:

None

G1044

Statement:

Comply with Federal accessibility standards contained in Section 508 of the Rehabilitation Act of 1973 (as amended) when developing software user interfaces.

Rationale:

Applicable software must comply with Federal standards to enable better application use for those with disabilities.

Referenced By:

Designing User Interfaces for Accessibility

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1044.1]

Do all Web document **HTML**, **JSP**, **ASP**, and **CSS** follow the Disability Act guidelines?

Procedure:

Check to make sure all Web documents follow the guidelines.

Use available validation tools to validate Section 508 accessibility and WAI accessibility. Go to http://www.contentquality.com/Default.asp to validate the page.

Example:

None

G1045

Statement:

Define **XML** format information separately in **XSL**.

Rationale:

XML documents should be free of any presentation information and should only contain data. Separating presentation data from content allows multiple presentations for the same content data.

Referenced By:

Defining XML Schemas
XML Rendering

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1045.1]

Check for presentation information in XML documents?

Procedure:

Does the XML document contain only data?

If the XML document is not an document, does it contain presentation information?

Example:

None

G1049

Statement:

Do not use **ActiveX** controls.

Rationale:

Browser incompatibility poses serious security risk, because it does not run inside a sandbox. ActiveX controls are like **applets**, except they are not restricted by a sandbox and can access client machine resources such as the hard disk directly. This makes them very dangerous.

Referenced By:

Active Server Pages (ASP)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1049.1]

Does the ASP use any ActiveX controls?

Procedure:

Check for Active X controls inside Web pages.

Example:

None

G1050

Statement:

In **ASP**, isolate the presentation tier from the middle tier using **COM** objects.

Rationale:

This is the best way to isolate the presentation tier from the middle tier in ASP.

Derived From:

G1058

Referenced By:

Active Server Pages (ASP)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1050.1]

Is all the middle tier code isolated from the presentation tier in ASP via COM?

Procedure:

Verify that ASP files do not contain middle-tier code. Instead, this code should be in COM objects referenced from the ASP.

Example:

None

G1052

Statement:

Use the code-behind feature in ASP.NET to separate presentation code from the business logic.

Rationale:

Separating presentation code from business logic allows the developers and content designers to work independently. It also makes the code more maintainable because changes in the design elements or business elements do not affect each other.

Referenced By:

Active Server Pages for .NET (ASP.NET)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1052.1]

Is there code in ASP pages?

Procedure:

Check to make sure that ASP files have the code-behind attribute in the first line instead of embedded C# code in the ASP.

Example:

None

G1053

Statement:

Do not embed HTML code in any code-behind code used by aspx pages.

Rationale:

Intermixing VB or C# or C++ with presentation code (HTML) makes the code unnecessarily difficult to maintain by both the developer and designer. This is similar in concept to Java's not embedding HTML code in **servlets**.

Derived From:

G1058

Referenced By:

Active Server Pages for .NET (ASP.NET)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1053.1]

Check for HTML code in code-behind code.

Procedure:

Check the code-behind file (**`.aspx.vb`** for example) for any HTML tags.

Example:

None

G1055

Statement:

Use a fully qualified, registered **namespace** with identity information for all custom controls.

Rationale:

.Net allows users to create a custom control from a Web page. This allows the custom Web page to be reusable just like a GUI control. This feature is great; however, users must fully qualify their controls to prevent namespace collisions.

Referenced By:

Active Server Pages for .NET (ASP.NET)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1055.1]

Does the **ASP** register its identity?

Procedure:

Check the **`.aspx`** file and make sure there is a statement to register the custom control.

Example:

None

G1056

Statement:

Specify a versioning policy for **.NET** assemblies.

Rationale:

Versioning assemblies and configuring dependent assemblies allow the **Common Language Runtime** (**CLR**) to load the proper assemblies at runtime for an application. This insulates the application from system configuration changes.

Referenced By:

Active Server Pages for .NET (ASP.NET)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1056.1]

Does the application assembly have versioning information?

Procedure:

Check the application assembly manifest for versioning information.

Use the .NET configuration tool to check for versioning policy and versioning information.

Example:

None

G1058

Statement:

Use the Model, View, Controller (MVC) pattern to decouple presentation code from other tiers.

Rationale:

Separating data-layer code from presentation-layer code provides the ability to base multiple views on the same model. This is especially important in the enterprise model because often, the user interface varies with the device (browser, mobile phone, thick client, etc.).

Isolating different layers allows changes to occur in each layer without impacting other layers. For instance, if the data layer (model) decides to switch databases, the changes are isolated to the data layer and do not affect the view layer or controller layer.

Lastly, because MVC architecture enforces separation between presentation, processing, and data layer, this allows functionality to be loosely coupled and therefore more suited for reuse.

Justifies:

G1053 G1050

Referenced By:

Active Server Pages (ASP)
Active Server Pages for .NET (ASP.NET)
Java Server Pages (JSP)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1058.1]

Does the application use a Model 2 (MVC) pattern?

Procedure:

Check to see if all requests are being mapped to a single controller servlet.

Check that all page rendering are being done by a and not a .

Example:

None

2) Test: [G1058.2]

Does the application enforce clear separation between data layer (model), presentation layer (view), and middle/business layer (controller)?

Procedure:

Check to make sure the application presentation is not accessing the database directly.

Check to make sure the application data layer (model) is not implementing business logic (store procedures).

Check to make sure the middle/business layer (controller) does not contain presentation code. For example, make sure servlets do not generate HTML.

Make sure access to the database is isolated to Data Access Object instead of proliferated throughout the middle layer.

Example:

None

G1060

Statement:

Encapsulate Java code that is used in **JSP**(s) in tag libraries.

Rationale:

Separating code from presentation allows developers and designers to work independently. It makes the code reusable and more maintainable because it is defined in a tag library.

Referenced By:

Java Server Pages (JSP)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1060.1]

Do the JSP pages use tag libraries?

Procedure:

Look through the JSP pages for embedded Java source code.

Example:

None

G1071

Statement:

Use vendor-neutral interface connections to the enterprise (e.g., **LDAP**, **JNDI**, **JMS**, databases).

Rationale:

Increase **portability** and maintainability. Many of the newer connection mechanisms are vendor-neutral. Use these instead of isolation design patterns or vendor-specific connection mechanisms.

Derived From:

G1007

Justifies:

G1239

Referenced By:

JNDI Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1071.1]

Is the connection mechanism vendor-neutral?

Procedure:

Examine the source code for vendor-specific imports or includes. Use only standard APIs.

Example:

None

G1073

Statement:

Isolate vendor extensions to enterprise-services standard interfaces.

Rationale:

Vendor extensions are convenient but help create "vendor lock" and reduce vendor neutrality and migration. It is best to avoid these extensions altogether. If that is not possible, then isolate them in an **adapter** or a wrapper-like construct.

Derived From:

G1008

Referenced By:

Publish and Insulate Public Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1073.1]

Are vendor extensions to enterprise services used?

Procedure:

Make sure that no vendor-specific code is included or imported except as part of an adapter or wrapper.

Example:

None

G1078

Statement:

Document the use of non-**Java EE**-defined **deployment descriptors**.

Rationale:

Deployment descriptors that are not defined by the J2EE specification are not portable between **application servers**. For example, BEA WebLogic has a vendor-specific deployment descriptor called **`weblogic-ejb-jar.xml`** and JBoss has a vendor specific deployment descriptor called **`jboss-jar.xml`** .

Referenced By:

Java EE Environment

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1078.1]

Are all the XML files that are not part of the Java EE specification identified in a delivered document?

Procedure:

Search all XML documents in the META-INF and WEB-INF directories and identify any XML files that are not defined by Java EE. These files should be in a README or other delivered file that describes their purpose:

Example:

Web application	**`WEB-INF/web.xml`**
EJB JAR	**`META-INF/ejb-jar.xml`**
J2EE Connector	**`META-INF/ra.xml`**
Client application	**`META-INF/application-client.xml`**
Enterprise application	**`META-INF/application.xml`**

G1079

Statement:

Isolate tailorable data values into the **deployment descriptors** for **Java EE** applications.

Rationale:

Do not hard-code tailorable data into source files. The standard location for tailorable data for Java EE applications is in deployment descriptors. Developers should not "reinvent the wheel" by creating a non-standard mechanism for retrieving configurable data. Make tailorable data accessible through application contexts provided by the application **container** (**Java EE application server**).

Referenced By:

Java EE Environment
JNDI Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1079.1]

Is tailorable data configured using deployment descriptors?

Procedure:

Check the deployment descriptor for instances of tailorable data.

Example:

Name-value pairs such as **environment variables** configured using resource-env-ref elements.

JNDI locations configured using resource-ref elments.

G1080

Statement:

Adhere to the Web Services-Interoperability Organization (**WS-I**) Basic Profile specification for **Web Service** environments.

Rationale:

Most of the **COTS** Web service products have already met this requirement. This is intended to cause a rejection of the non-standard Web server.

The WS-I Basic Profile specification is available from the Web Services Interoperability Organization Web site: WS-I Org Basic Profile;additional information is available via the Microsoft Developer Network (MSDN): Microsoft Basic Profile.

Referenced By:

WS-I Compliance

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1080.1]

Is the Web service product WS-I Basic Profile specification compliant?

Procedure:

Identify the Web service product being used, and verify through a literature search that it is WS-I Basic Profile specification compliant.

Example:

None

G1082

Statement:

Use the document-literal style for all data transferred using **SOAP** where the document uses the **World Wide Web Consortium** (**W3C**) **Document Object Model** (**DOM**).

Rationale:

The document-literal style requires defining the input and output parameters to a Web service as documents that follow the W3C Document Object Model (DOM). The DOM acts as a contract between the **producer** and the **consumer** of the Web service that is formal, well-defined, and rigorous. Validating the DOM against an **XML** Schema Definition (**XSD**) can help resolve discrepancies in the interface.

Referenced By:

WS-I Compliance
SOAP

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1082.1]

Does the **WSDL** define input, output, or returned parameters as Documents that follow the **W3C** Document Object Model (**DOM**)?

Procedure:

Review all WSDL files used to describe a Web service, and make sure they only pass documents. Document types should be **`xsd:anyType`**.

Example:

None

G1083

Statement:

Do not pass **Web Services-Interoperability Organization** (**WS-I**) **Document Object Model** (**DOM**) documents as strings.

Rationale:

Because of the relative simplicity of converting an **XML** document to a string, it is easy to pass an entire document as a string rather than as an XML document. This can cause problems if the document contains tags that are similar to the tags used in the **SOAP**. Passing it as an XML document ensures that the document is treated as a single entity.

Referenced By:

WS-I Compliance

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1083.1]

Does the **WSDL** define input, output, or returned parameters as strings?

Procedure:

Review all the WSDL files used to describe a Web service and make sure that they only pass documents, not strings. Document types should be **`xsd:anyType`**.

Example:

None

G1084

Statement:

Validate documents transferred using **SOAP** against the **W3C XML** Standard by an **XML Schema Definition** (**XSD**) defined by the **Community of Interest** (**COI**).

Rationale:

Numerous **COIs** are defining data specific to their needs. Many are capturing the data exchange requirements through **XML schemas**. **COI** information service definitions identify the appropriate schema. **SOAP** Web service implementations per the **COI** should be faithful to these requirements. Use of **COI** schemas will minimize the risk to interoperability.

For example, the Joint Air and Missile Defense (JAMD) **COI** is working in accordance with the DoD Network Centric Data Strategy.

Referenced By:

SOAP
WSDL

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1084.1]

Has the Program adopted **COI** (Community of Interest) data schemas?

Procedure:

Check the DoD Metadata Registry for the **COI** schemas to compare to program **WSDL** references. Check code for validation processing.

Example:

None

G1085

Statement:

Establish a **registered namespace** in the **XML Gallery** in the **DoD Metadata Registry** for all DoD Programs.

Rationale:

A **registered namespace** permits unique identification and categorization of a Program which avoids name collisions and conflicts. The DoD Net-Centric Data Strategy requires storing data products in shared spaces to provide access to all authorized users and tagging these data products with **metadata** to enable discovery of data by authorized users. The use of a unique **registered namespace** provides an absolute identifier to products associated with a particular product and is an **XSD** schema requirement.

Referenced By:

Using XML Namespaces
WSDL

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1085.1]

Does the Program have an assigned namespace DoD Metadata Registry?

Procedure:

Check the DoD Metadata Registry to determine whether program is associated with **COI**(s).

Example:

None

G1087

Statement:

Validate all **Web Services Definition Language** (**WSDL**) files that describe **Web services**.

Rationale:

Manually editing a **WSDL** file is error-prone, work-intensive, and hard to maintain. However, if the user wants to do it, there is no way to detect a manually edited file from one that was auto generated. The important thing is not how the **WSDL** file is generated but rather that the **WSDL** file is valid. It must be validated with a **WSDL** validator.

Note: Not all **WSDL** files that are generated and valid are necessarily interoperable.

Referenced By:

Insulation and Structure
Web Services
WSDL

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1087.1]

Can the **WSDL** file be validated?

Procedure:

Download a validation tool and test WSDL files.

Example:

Sample tools:

WS-I Organization:	http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools
Eclipse:	http://dev.eclipse.org/viewcvs/indextech.cgi/wsvt-home/main.html?rev=1.20
XMethods:	http://xmethods.net/ve2/Tools.po
Pocket Soap:	http://pocketsoap.com/wsdl/

G1088

Statement:

Use isolation design patterns such as **facade**, **proxy**, or **adapter** to isolate the application from the connection and manipulation of **SOAP** messages.

Rationale:

Insulating Web-services (network)-specific code using standard abstractions such as a proxy object or an adapter will insulate the application from changes in Web service code and make the code easier to maintain, because it is centrally located.

Referenced By:

Insulation and Structure
Web Services
SOAP

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1088.1]

Are Web service calls inside of the application code?

Procedure:

Check for proliferation of Web service calls inside an application.

Example:

None

2) Test: [G1088.3]

Are SOAP-client calls inside the application code?

Procedure:

Check to see if SOAP-client code is proliferated inside the application code?

Example:

None

3) Test: [G1088.2]

Are Web service calls isolated in a single adapter or proxy object?

Procedure:

Check to see if all Web service calls are isolated to a single adapter or proxy object.

Example:

None

G1090

Statement:

Do not hard-code a **Web service's endpoint**.

Rationale:

This causes unnecessary dependencies between the client code and the Web service that it uses.

Sometimes hard-coding may be unavoidable. For example, many tools provided by Web service vendors hard-code the Web service's URL in the generated client-side helper classes.

Referenced By:

Web Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1090.1]

Are there any hard-coded URLs in the client-side code?

Procedure:

Parse the client code looking for hard-coded URLs.

Example:

The Java code samples below illustrate how this might be done. The first sample shows parameters that are hard-coded; the second sample shows how parameters and Web service endpoints are insulated.

1. Hard-coded parameters:

```
// Sample code that has hard-coded parameters
// before applying insulation
public static void main
  ( String[] args
  ) throws Exception
{ //The SOAP endpoint
  String sSoapEndpoint
     = "http://live.capescience.com:80"
       + "/ccx/AirportWeather";
   AirportWeatherClient myProxy = null;
  try
  { myProxy
     = AirportWeatherClientFactory.create
        ( sSoapEndpoint);
   System.out.println
     ("Location: "
      + myProxy.getLocation(args[0])
     );
   //rest of code removed for brevity
  } // End try
```

```
  Catch ( Exception exception )
  { System.out.println("Error: " + exception);
  } // End catch
};//end of main program
```

2. Insulated parameters and Web service endpoints

a. Property file - this code shows the property file itself:

c. Client sample code:

```
import java.io.*;
import java.rmi.*;
import java.util.*;
import AirportWeatherClient; // auto-generated SOAP
                             // client from IDE */
public class WeatherProxy
  implements airportWeatherProxy
{
 //
 //code removed for brevity
 //
 public WeatherProxy
  ( String propFileStr )
 { try
   { getEndPoint(propFileStr);
   } // End try
   catch(Exception e)
   { // Handle exception here
   } // End catch
   connect2SOAP();
 }// End constructor
 /* public api's */
 public String getLocation()
 { return location;
 } // End getLocation
   . . . // Other public API's removed for brevity
 private void getEndPoint
  ( String propsFile )
  throws Exception
 { if ( propsFile == null || propsFile.length() == 0 )
   { throw new Exception
       ( "SOAP EndPoint parameter not defined");
   } // End if
   props = new Properties();
   try
   { InputStream is = new FileInputStream(propsFile);
     props.load(is);
     is.close();
    } // End try
    catch ( Exception exception )
    { throw new Exception
        ( "can't read props file " + propsFile);
    } // End catch
    Enumeration enum = props.propertyNames();
    while ( enum.hasMoreElements() )
    { String endPointString = null;
      String propName = enum.nextElement().toString();
      if ( propName.equals ( endPointString ) )
     { soapEndpoint = props.getProperty( propName );
       break;
     } // end if
    } // End while
  }//end getEndPoint
 private void connect2SOAP()
 { try
   { myProxy
       = AirportWeatherClientFactory.create
           ( soapEndpoint );
     . . . //code removed for brevity
```

```
    } // End try
    catch ( Exception exception )
    { System.out.println
        ( "Error connecting to SOAP server: "
           + exception
        );
    } // End catch
 } // End connect2SOAP
  private Properties props = null;
  private String propsFile = null;
  private AirportWeatherClient myProxy = null;
  private String soapEndpoint = null;
  private String location = null;
}//end WeatherProxy
public class Weather
{ private static WeatherProxy myWeatherProxy = null;
  public static void main
    ( String[] args
    ) throws Exception
  { try
    { myWeatherProxy = new WeatherProxy ( args[0] );
    } // End try
    Catch ( Exception exception )
    { throw new Exception
        ( "can't connect to SOAP server");
    } // End catch
    System.out.println
      ( "Location: "
        + myWeatherProxy.getLocation()
      );
    . . . //code deleted for brevity
  }//end main
}//end Weather
```

G1093

Statement:

Ensure **Web services** handle **SOAP** exceptions and faults.

Rationale:

SOAP exceptions result when there are connectivity problems or violations in the SOAP protocol between the client and the server.

Justifies:

G1095

Referenced By:

SOAP
Error Handling

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1093.1]

Does the Web application client have exception handlers for **`SOAPException`**s.

Procedure:

Check to see that the Web application client has an exception block specifically for **`SOAPException`**.

Example:

None

2) Test: [G1093.2]

Does the Web application client test the SOAP response for a fault?

Procedure:

Verify the Web application client handles a true value returned from the **`response.generatedFault`**.

Example:

None

G1094

Statement:

Catch all exceptions for application code exposed as a **Web service**.

Rationale:

Any exception can reveal system internals and thus compromise security. Also, internal exceptions are not user friendly.

Referenced By:

Error Handling

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1094.2]

Does each exposed Web method catch all possible runtime exceptions and re-throw a declared application runtime exception?

Procedure:

Verify that each exposed Web method has an exception block that catches all possible exceptions and then re-throws them as a declared application exceptions.

Example:

None

2) Test: [G1094.1]

Does each exposed Web method catch all possible exceptions and re-throw a declared application exception?

Procedure:

Verify that each exposed Web method has an exception block that catches all possible exceptions and then re-throws them as a declared application exceptions.

Example:

None

G1095

Statement:

Use **W3C** fault codes for all **SOAP** faults.

Rationale:

Having predefined and accepted fault codes allows consumers to handle SOAP faults appropriately without prior knowledge of custom fault codes.

Derived From:

G1093

Referenced By:

SOAP
Error Handling

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1095.1]

Does the Web application throw fault codes from the accepted list of fault codes?

Procedure:

Verify that each fault code thrown by the Web application is from the accepted list of SOAP fault codes defined by the W3C.

Example:

None

G1101

Statement:

Use **Web services** to bridge **Java EE** and **.NET**.

Rationale:

The easiest and best way to bridge Java EE and .NET is to define a Web service.

There are other ways to bridge Java EE and .NET using **COTS** products. If used, these should follow the **ANSI** Abstract Syntax Notation One (ASN.1) standard (http://asn1.elibel.tm.fr/en/standards/index.htm#asn1).

ASN.1 is a formal notation for describing data transmitted by telecommunications protocols. It applies regardless of language implementation, physical representation of this data, application, and degree of complexity (http://asn1.elibel.tm.fr/en/introduction/index.htm).

Referenced By:

.NET Framework

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1101.1]

Are Java and .NET files in the project?

Procedure:

Look for files with the .java, .class, .obj, .cs, .cc, or .c extensions existing with the source code.

Example:

None

G1117

Statement:

Isolate topic and queue names by not hard-coding them in **client** code.

Rationale:

Since topics and queues are vendor-specific, maintain portability by isolating the hard-coded topics and queues from the rest of the application. To do this, use helper classes or property files.

Referenced By:

Message-Based Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1117.1]

Does the client code use hard-coded topics and queues in unisolated places in the application?

Procedure:

Verify that all occurrences of hard-coded topics and queues are in isolated locations within the source code.

Example:

None

G1118

Statement:

Localize **CORBA** vendor-specific source code into separate modules.

Rationale:

The general guidance is to minimize CORBA vendor-specific source code, while recognizing that vendor-specific features are necessary in certain circumstances. However, isolating vendor-specific code reduces maintenance effort.

Vendor capabilities tend to change more rapidly than CORBA-standard specifications. Experience shows that vendor updates frequently require modification to application source code, due to changing vendor interface conventions. These modifications impose vendor-version-specific constraints on the application, thereby complicating maintenance.

Example

Encapsulating CORBA ORB operations

The following examples show how to encapsulate binding operations for a C++ ORB, and naming service operations for a Java ORB.

C++ ORB binder template

The code below shows a sample template for binding to the C++ ORB. IONA's ORBIX was used in this example.

```
/* ====================================================
ServerBinder.h (Template)
this is a generic binder to ORBIX
==================================================== */
#ifndef _BINDER_H_
#define _BINDER_H_
#ifndef IOSTREAM_H
#define IOSTREAM_H
#include <iostream.h>
#endif
#ifndef STDLIB_H
#define STDLIB_H
#include <stdlib.h>
#endif
template <class SERVERNAME, class VARPTR>
class Binder
{ private:
    char* serverName;
  public:
    Binder(char* svName):serverName(svName){};
    ~Binder(){};
    int bind( VARPTR* p)
    { int attempts = 0, success = 0;
      int maxtries = 5, retval = 0;
      while ( ( attempts < maxtries )
             && (!success)
            )
      { ++attempts;
        cout << "Binding to server, attempt "
            << attempts
            << endl;
        try
        { (*p) = SERVERNAME::_bind();
```

```
          cout << "Bound to server"
              << endl;
          success = retval = 1;
        } // End try
        catch ( CORBA::SystemException &systemException )
        { cout << "SystemException, ServerBinder::bind"
              << endl
              << systemException;
          success = 1;
          retval = 0;
        } // End catch SystemException
         catch (...)
         { cout << "unknown Exception, ServerBinder::bind"
               << endl;
           success = 1;
           retval = 0;
         } // End catch all
      } //end while
      return retval;
    } //end bind
} //end Binder
#endif
```

Ada ORB binder template for C++

The code below shows a C++ template for binding to an Ada ORB. ORBexpress was used in this example.

```
/* ====================================================
ada_binder.h (Template)
this is a generic binder to ORBExpress
==================================================== */
#ifndef _ADA_BINDER_H_
#define _ADA_BINDER_H_
#ifndef IOSTREAM_H
#define IOSTREAM_H
#include <iostream.h>
#endif
#ifndef STDLIB_H
#define STDLIB_H
#include <stdlib.h>
#endif
template <class SERVERNAME, class VARPTR >
class Ada_Binder
{ private:
    char* adaIorString;
  public:
     Ada_Binder
       ( char* iorString)
       : adaIorString ( iorString )
     {};
     ~Ada_Binder(){};
     int bindToAda( VARPTR* p)
     { int attempts = 0, success = 0;
       int maxtries = 5, retval = 0;
       while ( ( attempts < maxtries)
              && (!success)
             )
       { ++attempts;
         cout << "Binding to server, attempt "
             << attempts
             << endl;
         try
         { cout <<"adaIorString:"
               << endl
               << adaIorString
               << endl;
               (*p) = SERVERNAME::_bind(adaIorString);
//can't use string_to_object in this version
//it kills the ada IOR
//            CORBA::Object_ptr myptr
              CORBA::Orbix.string_to_object
                ( adaIorString );
```

```
//            (*p) = SERVERNAME::_narrow(myptr);
            cout << "Bound to server" << endl;
            success = retval = 1;
         } // End try
         catch (CORBA::SystemException& systemException)
         { cout << "SystemException, "
               << "AdaServerBinder::bind"
               << endl
               << systemException;
           success = 1;
           retval = 0;
         } // End SystemException
         catch (...)
         { cout << "Unknown Exception, "
               << "AdaServerBinder::bind"
               << endl;
           success = 1;
           retval = 0;
         } // End catch all
      } // end while
      return retval;
    } // end bind
} // end ADA_Binder
#endif
```

Example

Naming service operations for a Java ORB

Java helper class

This example is a helper class, JavaNamingHelper.java, that encapsulates CORBA naming service operations for all services to use. We used Java JDK 1.4 ORB to create this example.

```
import java.util.*;
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import org.omg.CORBA_2_3.ORB.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContext.*;
import org.omg.CosNaming.NamingContextPackage.*;
import CBRNSensors.JSLSCAD.*;
public class JavaNamingHelper
{ static NamingContext nameSvc = null;
  static org.omg.CORBA.Object objref = null;
  static JSLSCADSensor myCBRNSensor = null;
  static org.omg.CORBA.Object myobj = null;
  public JavaNamingHelper()
  {
  }
  private static void showNamingContext
    ( org.omg.CORBA.ORB myorb )
  {
  public static NamingContext getNamingSvc
    ( org.omg.CORBA.ORB lclorb,
      String nameSvcName
    )
  { NamingContext lclNameSvc = null;
    try
    { org.omg.CORBA.Object nameSvcObj
        = lclorb.resolve_initial_references
            ( "NameService" );
       // . . . other business logic removed
       //       for brevity
    } // End try
    catch(org.omg.CORBA.COMM_FAILURE cf)
    { . . . // error code goes here
    } // End cstch
    catch ( org.omg.CORBA.ORBPackage.InvalidName invalidName)
```

```
    { . . . // error code goes here
    } // End catch
    catch ( SystemException systemException )
    { . . .// error code goes here
    }
  } // End getNamingSvc
  public static org.omg.CORBA.Object getObjFromNameSvc
    ( org.omg.CORBA.ORB myorb,
      String targetSensorName
    )
  { . . . // business logic goes here
  } //end getObjFromNameSvc
  public static int setObj2NameSvc
    ( org.omg.CORBA.ORB myorb,
      BasesSensor mySensor,
      String targetSensorName
    )
  {. . . // business logic goes here
  }//end setObj2NameSvc
}; //end class JavaNamingHelper
```

Java server implementation

The code below is a sample Java server implementation that uses the naming service helper class.

```
import java.io.*;
import java.util.*;
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import org.omg.CORBA_2_3.ORB.*;
import org.omg.PortableServer.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContext.*;
import org.omg.CosNaming.NamingContextPackage.*;
class MyServer
{ public static Properties props;
  public static ORB myorb = null;
  public static NamingContext nameSvc = null;
  public static RootSensor mySensor = null;
  public static String propertyFilePath = null;
  public static final String MY_SENSOR_NAME = "MYSENSOR";
  static public void main(String[] args)
  { // handle arguments
    System.out.println(" CORBA Server starting...\n");
    try
    { // Initialize the ORB.
      myorb = ORB.init(args, props);
      //instantiate servant and create ref
      POA rootPOA
        = POAHelper.narrow(myorb.resolve_initial_references
          ( "RootPOA" );
     . . . // rest of initialization code goes here
    } // End try
    catch ( org.omg.CORBA.ORBPackage.InvalidName invalidName )
    { . . . //error code goes here
    } // End invalidName
    // other exception types to catch go here
    catch ( SystemException systemException)
    {  System.err.println ( systemException );
    } // End systemException
    // naming service hookup
    JavaNamingHelper.setObj2NameSvc
      ( myorb,mySensor,
        MY_SENSOR_NAME
      );
    try
    { System.out.println(" Ready to service requests\n");
      myorb.run();
    } // End try
    catch(SystemException systemException)
    { System.err.println ( systemException );
    } // End catch systemException
```

```
  } // End static block
} // End MyServer
```

Java client implementation

The code below is a sample client implementation that uses the naming service helper class.

Derived From:

G1008

Justifies:

G1202

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1118.2]

Are any non-CORBA compliant CORBA:: objects declared or defined in the module?

Procedure:

Review the code for a service that can be used to obtain configuration.

Example:

None

2) Test: [G1118.1]

Does the module contain vendor names anywhere in code text?

Procedure:

Review the code looking for a service that can be used to obtain configuration.

Example:

None

G1119

Statement:

Isolate user-modifiable configuration parameters from the **CORBA** application source code.

Rationale:

Configuration parameters control the behavior of the CORBA **ORB** service environment and client/service processes during startup, execution, and termination. This parameterization allows execution-time control modification without having to rebuild, reinstall, or redeploy.

Configuration defines the state of the client-and-service environment throughout the lifetime of the processes involved. This relates to considerations such as the allocation of threading and resources, **POA** policies, the instantiation of servants and their invocations, failure and security behavior, connection management, quality of service prioritization, and so forth. The point is that CORBA provides an extremely complex but flexible environment for distributed computing interaction. Consequently, the designer requires flexible guidance to handle this option-rich environment.

Configuration processes and their related parameters fall into two categories. The first involves configuration matters, which are defined to be perpetually static by the system architecture. The second involves matters that are intended to be modifiable by users.

The first category, immutable configuration settings, relates to fundamental underlying assumptions that are foundational for the implementation. These are matters for which no user modification is ever intended as it would lead to unspecified behavior. Consider the example of a service implementation that is programmed to be single threaded. In this case, multi-threading controls are irrelevant and multiple instantiation would lead to dangerous confusion. For immutable configuration parameters, localized and well-commented implementation in the application source code is appropriate.

For user-modifiable configuration settings, there are two further by-design divisions. The first involves configuration settings that are intended to be accessible by distributed processes. The second involves host-specific settings which relate to resources locally available, for which remote access is not desired. These are discussed in the related sublevel guidance

Justifies:

G1204 G1205

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1119.1]

See G1204.

Procedure:

Example:

2) Test: [G1119.2]

See G1205 .

Procedure:

Example:

G1121

Statement:

Do not modify **CORBA** Interface Definition Language (**IDL**) compiler auto-generated stubs and skeletons.

Rationale:

The purpose of the IDL auto-generated stub and skeleton files is to provide a source code facility/mechanism for the developer in a specific language to use the IDL-described object interface in that specific language. The internal content of these files changes with the application's IDL modification, with IDL compiler-environment configuration settings, and with vendor-product compiler and **ORB** upgrades. By design, these files are not intended to be modified by the application developer. Developer modification of any auto-generated stub or skeleton file will typically lead to very severe maintenance hazards and failed application rebuild results.

The stub files describe the language source-code interface from the client side. Their use involves including the client stub header in the application's call invocation code.

The skeleton files describe the language source code interface from the service implementation side. Their use involves including the skeleton header in the application's operator implementation code. Their use also requires developer modification of a renamed clone of the auto-generated skeleton body file. These techniques are described in every ORB vendor's programming reference manuals.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1121.1]

Is any application code contained in the auto-generated code?

Procedure:

Inspect the auto-generated file creation/modification dates to verify that no tampering occurred after the IDL compilation step in the build process.

Example:

The following examples are all based upon a single CORBA IDL interface.

MyIdlInterface.idl

```
interface MyIdlInterface
{
  readonly attribute string version;
  void stop();
  void start();
  string error();
}; // End MyIdlInterface
```

ORBExpress compiler

The ORBExpress IDL compiler generates these files:

- **`myIdlInterface.h`** - Client-side stub header
- **`myIdlInterface.cxx`** - Client-side stub implementation
- **`MyIdlInterface_s.h`** - Abstract servant header
- **`MyIdlInterface_s.cxx`** - Abstract servant implementation
- **`MyIdlInterface_impl.h`** - Server implementation header
- **`MyIdlInterface_impl.cxx`** - Server implementation implementation

> ***Note:*** *The only files that should be edited are* ***`MyIdlInterface_impl.h`*** *and* ***`MyIdlInterface_impl.cxx`*** *. The IDL compiler checks for the existence of the implementation (i.e. _impl) files and will not overwrite them.*

MyIdlInterface_impl.cxx

```
// Generated for interface MyIdlInterface
// in myIdlInterface.idl
#include "MyIdlInterface_impl.h"
MyIdlInterface_impl::MyIdlInterface_impl
  ( PortableServer::POA* oe_poa,
    const char* oe_object_id
  ) : POA_MyIdlInterface
        ( oe_object_id,
          oe_poa
        )
{ . . . // TO DO: add implementation code here
} // emd constructor
MyIdlInterface_impl::MyIdlInterface_impl
  ( const MyIdlInterface_impl& obj )
  : POA_MyIdlInterface(obj)
{ . . . // TO DO: add implementation code here
} // End constructor
MyIdlInterface_impl::~MyIdlInterface_impl()
{ . . . // TO DO: add implementation code here
} // End destructor
CORBA::Char* MyIdlInterface_impl::version
  ( CORBA::Environment& _env )
{ return CORBA::string_dup(_version);
} // End version
void MyIdlInterface_impl::stop
   ( CORBA::Environment& _env )
{ . . . // TO DO: add implementation code here
} // End stop
void MyIdlInterface_impl::start
   ( CORBA::Environment& _env )
{ . . . // TO DO: add implementation code here
} // End start
CORBA::Char* MyIdlInterface_impl::error
  ( CORBA::Environment& _env )
{ CORBA::Char* result;
  . . . // TO DO: add implementation code here
  return result;
} // End error
```

Java JDK compiler

The Java JDK IDL compiler generates these files:

- **`MyIdlInterface.java`**

- **MyIdlInterfaceHelper.java**
- **MyIdlInterfaceHolder.java**
- **MyIdlInterfaceOperations.java**
- **MyIdlInterfacePOA.java**
- **_MyIdlInterfaceStub.java**

MyIdlInterfacePOA.java

```
/**
 * MyIdlInterfacePOA.java .
 * Generated by the IDL-to-Java compiler
 * (portable), version "3.1"
 * from myIdlInterface.idl
 */
public abstract class MyIdlInterfacePOA
  extends org.omg.PortableServer.Servant
  implements MyIdlInterfaceOperations,
             org.omg.CORBA.portable.InvokeHandler
{ . . . // rest of the auto-generated code removed for brevity
} // End MyIdlInterfacePOA
```

MyIdlInterfaceImpl.java

```
package myIdlImpl;
import org.omg.CORBA.*;
import org.omg.CORBA.ORB.*;
import org.omg.CORBA_2_3.ORB.*;
import org.omg.PortableServer.*;
public class  MyIdlInterfaceImpl
  extends  MyIdlInterfacePOA
{
  private String strVersion;
  private String errString;
  public String version ()
  { . . . // implementation code goes here
    return strVersion;
  } // End version
  public void stop ()
  { . . . // implementation code goes here
  } // End stop
  public void start ()
  { . . . // implementation code goes here
  } // End start
  public String error ()
  {. . . // implementation code goes here
   return errString;
  } // End error
} // End MyIdlInterfaceImpl
```

G1123

Statement:

Use the Fat Operation Technique in **IDL** operator invocation.

Rationale:

This reduces the CORBA messaging overhead. The performance cost of network CORBA messaging is determined by two factors: latency and marshaling rate. Call latency is the minimum cost of sending any message at all. The marshaling rate is determined by the sizes of sending and receiving parameters and of return values.

In the situation of a large number of objects involving objects that hold a small amount of stat, the call latency cost far exceeds the marshalling costs. Taking advantage of this reality, the "Fat Operation Technique" involves constructing structure objects which hold an aggregation of related attributes, and using the resulting structures in operation invocation parameters and returns. This amounts to transferring a larger amount of information with each network transaction.

For more information, see "Advanced CORBA Programming with C++" by Henning & Vinoski, 1999 Addison Wesley, Chapter 22.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1123.1]

Does the IDL contain function calls which have structure objects that are passed as parameters or returned from operators?

Procedure:

Inspect the IDL file and manually check for parameters or returns using objects defined as structures, and verify that they are passed from methods also declared in the IDL.

Example:

None

G1125

Statement:

Use the **Department of Defense Metadata Specification** (**DDMS**) for standardized tags and taxonomies.

Rationale:

These standardized tags or Metacards will be developed, maintained, and placed under configuration as appropriate and will comply with the **DDMS** and **COI** guidance. These include specifications defining the tagging for security classification and dissemination control. See the DoD Discovery Metadata Specification Web site (http://metadata.dod.mil/mdr/irs/DDMS/) for the current **DDMS** standards.

Referenced By:

Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1125.1]

Has the Program documented the profile used for published data assets in accordance with guidance?

Procedure:

Check the DoD Metadata Registry to determine whether the program is associated with **COI**(s).

Example:

None

G1127

Statement:

Use a **UDDI** specification that supports publishing discovery services.

Rationale:

UDDI provides a registration for services, and the **OASIS** UDDI 2.0 specification has become a standard method for publishing discovery services.

Referenced By:

Universal Description, Discovery, and Integration (UDDI)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1127.1]

Are the Web services registered in a **UDDI** registry?

Procedure:

Verify the registration in the UDDI registry.

Example:

None

2) Test: [G1127.2]

Is the registry **UDDI** 2.0 or higher?

Procedure:

Determine if the particular UDDI registry is UDDI Version 2.0 or higher.

Example:

None

G1131

Statement:

Use industry standard Universal Description, Discovery, and Integration (**UDDI**) **APIs** for all UDDI inquiries.

Rationale:

There is a standard **API** that uses **SOAP** messages to communicate with the UDDI registry. To increase compatibility and portability, use this API exclusively.

Referenced By:

Universal Description, Discovery, and Integration (UDDI)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1131.1]

Are all the interfaces to the UDDI registry made using the UDDI standard API?

Procedure:

The standard API for UDDI is SOAP based. Requests and responses are passed using documents. Test the traffic flow between the client and the UDDI registry for messages that are defined in the UDDI specification. Use standard libraries to send and receive the messages (e.g., JUDDI for Java).

Checking for the use of packages like JUDDI does not require the application to be running.

Example:

The following is an example as provided in the UDDI API reference:
http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm#_Toc25137712 .

find_binding

The find_binding API call returns a bindingDetail message that contains zero or more binding Template structures matching the criteria specified in the argument list.
Syntax

Syntax

Arguments

serviceKey	This uuid_key is used to specify a particular instance of a businessService element in the registered data. Only bindings in the specific businessService data identified by the serviceKey passed will be searched.

maxRows	This optional integer value allows the requesting program to limit the number of results returned.
findQualifiers	This optional collection of findQualifier elements can be used to alter the default behavior of search functionality. See the findQualifiers appendix for more information.
tModelBag	This is a list of tModel uuid_key values that represents the technical fingerprint of a bindingTemplate structure contained within the businessService specified by the serviceKey value. Only bindingTemplates that contain all of the tModel keys specified will be returned (logical AND). The order of the keys in the tModel bag is not relevant.

Returns

This API call returns a bindingDetail message upon success. In the event that no matches were located for the specified criteria, the bindingDetail structure returned will be empty (i.e., it contains no bindingTemplate data.) This signifies a zero match result. If no arguments are passed, a zero-match result set will be returned.
In the event of an overly large number of matches (as determined by each Operator Site), or if the number of matches exceeds the value of the maxRows attribute, the Operator site will truncate the result set. If this occurs, the response message will contain the truncated attribute with the value "true".

Caveats

If any error occurs in processing this API call, a dispositionReport element will be returned to the caller within a SOAP Fault. The following error number information will be relevant:

E_invalidKeyPassed	This signifies that the uuid_key value passed did not match with any known serviceKey or tModelKey values. The error structure will signify which condition occurred first, and the invalid key will be indicated clearly in text.
E_unsupported	This signifies that one of the findQualifier values passed was invalid. The invalid qualifier will be indicated clearly in text.

G1132

Statement:

Implement the data tier using readily available **COTS RDBMS** products that implement the **SQL** standard and provide a rich set of generic capabilities such as row-level locking, **stored procedures**, **triggers**, and a high-level language **API** interface.

Rationale:

COTS RDBMSs are mature technical products, the capabilities of which are being continually expanded to adapt to and accommodate new technologies. Moreover, there is a large technical community able to develop and maintain data systems based on these products. It is likely that a COTS RDBMS will provide all of the data tier capabilities required by the developer.

Referenced By:

Database Implementations

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1132.1]

Is the proposed COTS RDBMS product a readily available and supportable COTS product that implements the SQL standard?

Procedure:

Verify that the COTS RDBMS product is widely in use in the DoD environment (e.g., Oracle, SQL Server, or DB2), has a large support community, and is likely to be supported for the lifecycle of the project.

Example:

None

G1141

Statement:

Use standard **data models** developed by **Communities of Interest** (**COI**) as the basis of program or project data models.

Rationale:

Standard **data models** are under development in many areas of the DoD and will be stored in and made available from DoD **metadata** repositories. The use of these models or portions thereof supports interoperability among applications. The **C2IEDM** data model, used in the **Command and Control** area, is an example of one of these standard data model development efforts.

Justifies:

BP1404

Referenced By:

Database Development
Data Modeling
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1141.1]

Have standard **data models** been considered for use in the system?

Procedure:

Determine whether standard DoD **data models** exist for the technical areas accommodated in the system requirements. Verify that data model the developed for the application accommodates the use of these data models.

Example:

None

2) Test: [G1141.2]

If the system is a command-and-control application, has preference been given to the use of the Command & Control Information Exchange Data Model (**C2IEDM**) rather than locally defined values?

Procedure:

Examine the system **data model** and verify that the **C2IEDM** data model has been incorporated.

Example:

None

G1144

Statement:

Develop two-level database models: one level captures the **conceptual** or logical aspects, and the other level captures the **physical** aspects.

Rationale:

There are a number of modeling tools available that support entity-relationship diagram (ERD) development. Developers can use these tools to create conceptual/logical models that are independent of the **DBMS** in which the system is implemented and to develop the physical models that are translated directly into data definition language (DDL), the **SQL** code used to create the database. Using a conceptual/logical model permits implementation or reuse of a complex ERD on multiple **DBMS** products.

Referenced By:

Database Development
Data Modeling

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1144.1]

Have separate **conceptual**/logical and **physical** models been developed?

Procedure:

Verify the presence of a conceptual/logicalmodel0 and a physical model.

Example:

None

G1146

Statement:

Include information in the **data model** necessary to generate a **data dictionary**.

Rationale:

A **data dictionary** is an integral part of every system including databases. A description of each data item and the units in which the contents are measured are essential. **Data modeling** tools provide a mechanism for storing information necessary to produce a **data dictionary**.

Referenced By:

RDBMS Internals
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1146.1]

Does the data model include description information?

Procedure:

Examine the physical data model.

Example:

None

G1147

Statement:

Use **domain analysis** to define the constraints on input data validation.

Rationale:

Domain analysis is an integral part of any data system including databases. Domains describe the set or range of values that are acceptable for a specific data item. These include, at a minimum the following:

- Data type
- Precision
- Minimum
- Maximum
- Length

These values are used to validate the data.

In the database, the range checking is done via check constraints on the data item. These **check constraints** are generated from the **physical data model** as part of the DDL.

Referenced By:

Database Development
Data Modeling
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1147.1]

Does the data model include include constraints derived from domain analysis?

Procedure:

Examine the physical data model.

Example:

None

G1148

Statement:

Normalize data models.

Rationale:

Normalization is a central **tenet** of **relational database** theory. It is also part of **OOA**.

A database should usually be normalized to at least third normal form. Although there are seven normal forms, normalization beyond third normal form is rarely considered in practical database design.

Objects developed in the absence of data normalization are prone to unnecessary complexity required to keep multiply copies of data.

Referenced By:

Database Development
Data Modeling
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1148.1]

Is the database design in third normal form?

Procedure:

Examine the conceptual/logical **data model**.

Example:

None

G1151

Statement:

Define declarative **foreign keys** for all relationships between tables to enforce **referential integrity**.

Rationale:

Foreign Key constraints enforce referential integrity. The principle of referential integrity requires that the foreign key values of a child table are either null or match exactly those of the **primary key** in the parent table.

Referenced By:

RDBMS Internals
Database Development

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1151.1]

Have foreign-key constraints been incorporated into the database?

Procedure:

Examine the database to determine whether foreign-key constraints have been included in the database creation scripts and created in the database.

Example:

None

G1153

Statement:

Support n-tier architectures for efficient and accurate maintenance operations.

Rationale:

Modern software design methodologies call for the implementation of an n-tiered architecture. For example, the separation of presentation, middle and data tiers with well defined interfaces between each provides scalability, efficient maintenance and simplify development.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1153.1]

Has the system been designed and developed using a multi-tier architecture?

Procedure:

Verify that the system design accommodates a multi-tier architecture.

Example:

None

G1154

Statement:

Use **stored procedures** for operations that are focused on the insertion and maintenance of data.

Rationale:

Current software design methodologies and architectures call for the implementation of an n-tiered architecture with business rules in the middle tier and data stored in a separate data tier. When multiple applications access a common database, however, the rules may be best located at the data-tier level. Otherwise, changes in one application would have to be coordinated across all applications.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1154.1]

Are database triggers used?

Procedure:

Check for stored procedures that are triggered on insertion, deletion, and update events.

Example:

```
CREATE TRIGGER PersonCheckAge
AFTER INSERT OR UPDATE OF age
ON Person
FOR EACH ROW
BEGIN
  IF (:new.age < 0) THEN
    RAISE_APPLICATION_ERROR
      ( -20000,
        'no negative age allowed'
      );
  END IF;
END;.
```

G1155

Statement:

Use **triggers** to enforce **referential** or **data integrity**, not to perform complex **business logic**.

Rationale:

Triggers are fired on events. Current software design methodologies and architectures call for the implementation of an n-tiered architecture with business rules in the middle tier and data stored in a separate data tier. Implementing business logic in triggers, as well as in the middle tier, violates this concept.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1155.1]

Has business logic been incorporated into database triggers?

Procedure:

Examine the database trigger code to determine whether business logic or calls to stored procedures incorporating business logic have been coded into them.

Example:

None

G1190

Statement:

Use a build tool.

Rationale:

A build tool allows for the encapsulation of building instructions into machine-readable files or sets of files. The instructions can be successfully and consistently repeated.

Justifies:

G1218 G1219 G1221 G1222 G1223 G1224 G1220 G1225

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1190.1]

Does the program or project use a build tool?

Procedure:

Identify which build tool the program or project is using.

Example:

G1202

Statement:

Use the **CORBA Portable Object Adapter** (**POA**) instead of the **Basic Object Adapter** (**BOA**).

Rationale:

The CORBA Basic Object Adapter (BOA) was the CORBA Version 1 specification for the client-server object capability. The BOA specification was found to be so incomplete that vendor-specific interpretations were required for operable implementation. In CORBA Version 2, the Portable Object Adapter (POA) was significantly more complete and flexible. In the current marketplace, POA implementations are standard and, in quality implementations, are not vendor-specific. Consequently, using POA eliminates one significant area of vendor-specific coding.

BOA	POA
Focuses on CORBA server implementations and not CORBA object implementations Naming convention issues on server side Tightly coupled to **ORB** implementation Non-standardized way to connect to ORB Four activation models for server processes	Services for lifecycle management Abstract layer between ORB and object Standard, portable interface for communicating with ORB runtime Two servant incarnation styles

Derived From:

G1118

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1202.1]

Does any CORBA application code reference the **`CORBA::BOA`** identifier.

Procedure:

Review the code for the use of the **`CORBA::BOA`** identifier.

Example:

BOA Coding Example

Client Side

The code below shows a C++ CORBA client BOA initialization for the ORBIX ORB. Other ORB vendors may have different initialization sequences.

```
int main
  ( int argc,
    char **argv
  )
{ MyServer_var MyVar;
  CORBA::ORB_ptr myOrbPtr
    = CORBA::ORB_init(argc, argv,"Orbix");
  try
  { // The default is the local host:
    MyVar = MyServer::_bind(":ServerName");
  } // End try
  catch ( CORBA::SystemException &sysEx )
  { cerr << "Unexpected system exception" << endl;
    cerr <<&sysEx;
    exit(1);
  } // End CORBA::SystemException
  catch(...)
  { // an error occurred while trying
    // to bind to the grid object.
    cerr << "Bind to object failed" << endl;
    cerr << "Unexpected exception " << endl;
    exit(1);
  } // End catch ...
} // End main
```

Server Side

Use the code below as a model. This example shows a C++ CORBA server BOA init for the ORBIX ORB. For BOA, other ORBS will have a different initialization sequence.

```
try
{ MyObject::myOrb_
    = CORBA::ORB_init(argc, argv, "Orbix");
  MyObject::myboa_
    = MyObject::myOrb_->BOA_init(argc, argv, "Orbix_BOA");
} // End try
catch ( CORBA::SystemException &sysEx )
{ //some exception handling code
} // End catch
try
{ NoeLoggerCfg::myboa_->impl_is_ready("MyServiceName",
  CORBA::ORB::INFINITE_TIMEOUT);
} // End try
catch ( CORBA::SystemException &sysEx )
{ //exception handling code
}
```

POA Coding Example

Client Side

This example shows a C++ CORBA client POA init for the ORBIX ORB. For BOA, other ORBS will have a different initialization sequence.

```
int main
  ( int argc,
    char **argv
  )
{ CORBA::ORB_var myOrb = CORBA::ORB_init(argc, argv);
  try
  { CORBA::Object_var obj
      = ... // however you get the object reference
    if(CORBA::is_nil (obj))
    { cerr << "Nil object reference" << endl;
      throw 0;
    } // End if
  } // End try
  catch ( CORBA::SystemException &sysEx )
  { cerr << "Unexpected system exception" << endl;
    cerr <<&sysEx;
    exit(1);
  } // End catch CORBA::SystemException
  catch ( ... )
  { cerr << "Unexpected system exception" << endl;
    exit(1);
  } // End catch ...
  myinterface::myobject_var myvar;
  try
  { myvar = myinterface::myobject::_narrow(obj);
  } // End try
  catch ( CORBA::SystemException &sysEx)
  { cerr << "Unexpected system exception" << endl;
    cerr <<&sysEx;
    exit(1);
  } // End catch CORBA::SystemException
} // End main
```

Server Side

Use the code below as a model. This example shows a C++ CORBA server POA init for the ORBIX ORB. For POA, other ORBS will have a different initialization sequence.

```
int main
  ( int argc,
    char *argv[ ]
  )
{ try
  { // initialize the ORB
    orb_var orb  = CORBA::ORB_init(argc, argv, "Orbix");
    // obtain an object reference for the root POA
    object_var obj
      = orb->resolve_initial_references ("RootPOA");
    POA_var poa = POA::_narrow(obj);
    // incarnate a servant
    My_Servant_Impl servant;
    // Implicitly register the servant with the root POA
    obj =  servant._this ();
    //start the POA listening for requests
    poa -> the_POAManager ()->activate ();
    //run the orb's event loop
    orb->run ();
  } // End try
  catch ( CORBA::SystemException &sysEx )
  { // some exception handling code
  } // End catch
} // End main
```

G1203

Statement:

Localize frequently used CORBA-specific code in modules that multiple applications can use.

Rationale:

In a family of applications, similar patterns of CORBA Object Request Broker (**ORB**) invocation sequences frequently arise. This is common in service object initialization, policy association, discovery, binding, and release handling. Implementing this functionality in a utility library paradigm localizes the code to reduce maintenance and facilitate extensibility, and assures consistency across the family of applications.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1203.1]

Do the standard object initialization CORBA invocations occur in more than one module?

Procedure:

The presence of "**`CORBA::ORB_var`**" or "**`CORBA::ORB_init`**" in C++ indicates ORB initialization. The presence of "**`CORBA::Object_var`**" in C++ indicates ORB access.

Example:

None

2) Test: [G1203.3]

Do the standard object policy association CORBA invocations occur in more than one module?

Procedure:

The presence of "**`CORBA::PolicyList`**" in C++ indicates policy presence.

Example:

None

3) Test: [G1203.4]

Do the standard object discovery CORBA invocations occur in more than one module?

Procedure:

The presence of "**`Resolve_NamingService()`**"in C++ indicates intended access to one of CORBA's discovery capabilities.

Example:

None

4) Test: [G1203.5]

Do the standard object binding and release CORBA invocations occur in more than one module?

Procedure:

The presence of "**`::_narrow(obj.in())`**" or "**`CORBA::is_nil(`**" in C++ indicates activity associated with obtaining and validating an object binding to a legitimate reference. The presence of "**`CORBA(release)(`**" in C++ indicates intended release of a CORBA-bound object reference.

Example:

None

5) Test: [G1203.2]

Do the standard object policy association CORBA invocations occur in more than one module?

Procedure:

The presence of "**`CORBA::PolicyList`**" in C++ indicates policy presence.

Example:

None

G1204

Statement:

Create configuration services to provide distributed user control of the appropriate configuration parameters.

Rationale:

For user-modifiable configuration settings that are intended to be accessible by distributed processes at runtime, the appropriate mechanism for implementation involves **CORBA** services. The first form is a network service to be invoked as a client by the target system application at initialization. This can support a consistent, network-wide distribution of startup parameters. The second form is a service implemented by the target application which allows communication to the application during execution (after startup). This allows **real-time** configuration changes for matters such as Portable Object Adapter (**POA**) instantiation threading policies to address load management.

Derived From:

G1119

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1204.1]

Is a service defined in the IDL to obtain the configuration parameters?

Procedure:

Review the code for a service that can be used to obtain configuration.

Example:

The following code is an example of a CORBA server that instantiates a configuration service. The service manages the individual configuration parameters for the servers on the ORB.

Ada Example

```
CORBA.ORB.IIOP_English;
pragma Elaborate_All(CORBA.ORB.IIOP_English);
with CORBA ;
with CORBA.BOA ;
with CORBA.ORB ;
with CORBA.Object ;
with Configuration.Impl ;
with Configuration.Helper ;
with Ada.Exceptions ;
with Ada.Text_IO ;
```

```
with my_CORBA ;
with Event_Ada_API ;
procedure Configuration_Server is
    -- required for OrbExpress
    First_Variable : CORBA.ORB.Life_Span ;
    -- declare the object instance
    Configuration_Object : Configuration.Ref ;
    --variables needed for ior writing
    No_Timeout : constant := 0.0;
    Config_Name : constant String
      := Configuration.Helper.Simple_Name ;
    Config_Host : Corba.String ;
    Config_Port : Corba.String ;
begin -- Configuration_Server
  -- create (and initialize) the object
  -- config file is read and the port needed
  -- is in there
  Configuration_Object
    := Configuration.Impl.Create(Config_Name) ;
  GET_HOSTNAME:
  begin
    Config_Host
      := Configuration.Get_String
         ( Self => Configuration_Object,
           Name => Corba.To_Corba_String
                     ( "Local_Host_Shortname" )
         );
  exception -- GET_HOSTNAME
    when others =>
      Ada.Text_IO.Put_Line
        ( "ERROR: Missing parameter"
         & "<Local_Host_Shortname> "
         & "in the config_parameters.txt file."
        );
  end GET_HOSTNAME;
  GET_CS_PORT:
  begin
    Config_Port
      := Configuration.Get_String
         ( Self => Configuration_Object,
           Name => Corba.To_Corba_String
                     ( "Config_Service_Port" )
         );
  Exception -- GET_CS_PORT
    when others =>
      Ada.Text_IO.Put_Line
        ( "ERROR: Missing parameter "
         & "<Config_Service_Port> "
         & "in the config_parameters.txt file."
        );
  end GET_CS_PORT;
  Ada.Text_IO.Put_Line
    ( "Host => "
        & Corba.To_Standard_String(Config_Host)
        & " Port => "
        & Corba.To_Standard_String(Config_Port)
    );
  --timeout 0 so we can write IOR out
  CORBA.BOA.Impl_Is_Ready
      ( Time_Out             => No_Timeout,
        Server_Instance_Name => Config_Name,
        Listen_On_Endpoints  =>
          "tcp://"
          & Corba.To_Standard_String(Config_Host)
          & ":"
          & Corba.To_Standard_String(Config_Port)
       );
      -- ----------------------------------------------
      -- HERE IS WHERE CODE FOR THE IOR TO BE
      -- USED ON THE C++ ORB
      -- ----------------------------------------------
  -- get the IOR and write it to disk
  my_CORBA.Write_IOR_To_File
    ( Server_Name => Config_Name,
      Server_Ref  =>
```

```
        CORBA.Object.Ref(Configuration_Object)
    );
  READY_BLOCK:
  begin
    -- notify subscribers of availability
    -- of configuration parameters via the
    -- event service
    Event_Ada_API.Send
      ( Channel_Name => "Config_Channel",
        Event        => "Configuration Service Ready."
      );
  Exception - READY_BLOCK
    when others =>
      Ada.Text_IO.Put_line
        ( "Configuration_Server : "
         & Exception sending ready signal."
        );
  end READY_BLOCK;
  Ada.Text_IO.Put_line
    ( "Configuration_Server : "
     & Configuration Service Ready."
    );
  CORBA.BOA.Impl_Is_Ready
    ( Time_Out             => CORBA.Infinite_Timeout,
      Server_Instance_Name => Config_Name
    ) ;
exception -- Configuration_Server
  when X_Other: others =>
    Ada.Text_IO.Put_line
      ( "Configuration_Server : "
       & Ada.Exceptions.Exception_Name(X_Other)
      );
end Configuration_Server ;
```

C++ Example

The following code snippets depict a C++ server that instantiates a version collection service for an About box. It uses the IORs from the servers on the Ada ORB via the IOR files, and invokes those objects to get version information. It uses the utility templates for binding. It exemplifies the approach described in Encapsulate CORBA ORB operations for C++.

Note: *This was done on the ORBIX C++ and Ada ORBs.*

```
#include <iostream.h>
#include <rw/cstring.h>
#ifndef _STDIO_H
#include <stdio.h>
#endif
#ifndef _STRING_H
#include <string.h>
#endif
#ifndef _STDLIB_H
#include <stdlib.h>
#endif
#ifndef _ASSERT_H
#include <assert.h>
#endif
// Include files for all the objects desired for
// collecting version information
//Ada configuration service
#ifndef configuration_hh
#include <configuration.hh>
#endif
// include files for other desired services;
// removed for brevity
// other support objects and utilities
#ifndef _CORBA_UTILS__
#include <corba_utils.h>
#endif
#ifndef __LOG_API_H__
```

```
#include <log_api.h>
#endif
#ifndef _VERSION_AGENT_GLOBALS_H_
#include "version_agent_globals.h"
#endif
const RWCString  Version_Agent_i::MSG_VERSION_NOT_FOUND_
  = "Version Info. not found for ";
const CORBA::ULong Version_Agent_i::MAXSERVERS_
  = 12;
Version_Agent_i:: Version_Agent_i(): theVersionInfoPtr_(0)
{ theVersionInfoPtr_
     = new versionInfoType(MAXSERVERS_);
  theVersionInfoPtr_->length(MAXSERVERS_);
} // End constructor
Version_Agent_i:: ~Version_Agent_i()
{ // Do nothing
} // End destructor
/*********************************************************
FUNCTION NAME: createVersions
PURPOSE: helper function that gets the version info
INPUT:
OUTPUT:
*********************************************************/
void Version_Agent_i::createVersions ()
{ char *iorString;
  int bBindOk = 0;
  int versionCnt = 0;
  versionInfoType* rl = theVersionInfoPtr_;
  CORBA::ULong MAXSERVERS Version_Agent_i::MAXSERVERS_;
  // server variables for all the objects desired
  // for collecting version information
  // most declarations removed for brevity
    EventServiceFactory_var es_var;
  // Ada configuration service
    Configuration_var cfg_var;
  // === load the versions of the individual components
  // Code for other services removed for brevity
  // This is an ADA service using the IOR string
  {  //****************** config service ***************
    logMsg
      ( "get config service version",
        Log_Api::DEBUG_1_MSG
      );
    RWCString errMsg
      ( Version_Agent_i::MSG_VERSION_NOT_FOUND_.data()
      );
    errMsg.append ( "Configuration Service" );
    // here we get the IOR from the ADA orb using
    // the helper methods
      iorString = getIorFile("Configuration");
    //template class to hide binding issues to the ADA ORB
    If ( iorString )
    { Ada_Binder < Configuration,
      Configuration_var > bo ( iorString );
      bBindOk = bo.bindToAda(&cfg_var) ;
      // get the version info and load it
      If ( bBindOk
          && !( CORBA::is_nil(cfg_var))
         )
     { try
       { char* str = cfg_var->version();
         if ( str )
         { (*theVersionInfoPtr_)[versionCnt]
             = CORBA::string_dup(str);
          delete str;
         } // End if
         else
         { (*theVersionInfoPtr_)[versionCnt]
             = CORBA::string_dup(errMsg.data());
         } // End else
       } // End try
       catch(...)
       { (*theVersionInfoPtr_)[versionCnt]
           = CORBA::string_dup(errMsg.data());
       } // End catch
```

```
        cfg_var->_closeChannel();
      } // End if
      else
      { (*theVersionInfoPtr_)[versionCnt]
           = CORBA::string_dup(errMsg.data());
      } // End else
      if(iorString)
      { free (iorString);
        iorString = NULL;
      } // End if
    } //endif iorstring
    else
    { (*theVersionInfoPtr_)[versionCnt]
        = CORBA::string_dup(errMsg.data());
    } // End else
    //leaving scope releases the corba object
  } //end cfg_svf
  bBindOk = 0;
  versionCnt++;
  assert(versionCnt <= MAXSERVERS);
} // End createVersions
/*********************************************************
FUNCTION NAME: start
PURPOSE:  handle startup specific stuff
INPUT:
OUTPUT:
*********************************************************/
void Version_Agent_i:: start
  ( CORBA::Environment &IT_env
  ) throw (CORBA::SystemException)
{ //get all the version info
  createVersions();
} // End start
/*********************************************************
FUNCTION NAME: stop
PURPOSE:  handle stop specific stuff
INPUT:
OUTPUT:
*********************************************************/
void Version_Agent_i:: stop
  ( CORBA::Environment &IT_env
  ) throw (CORBA::SystemException)
{ // Release info
  // Let CORBA time out the service
  logMsg ( "stop received" );
  VersionAgentGlobals::myboa->setNoHangup ( 0 );
  VersionAgentGlobals::myboa->deactivate_impl
    ( "Version_Agent" );
} //end version impl
```

G1205

Statement:

Use non-source code persistence to store all user-modifiable **CORBA** service configuration parameters.

Rationale:

For user-modifiable configuration settings that are host-specific and that are not intended to be accessible by distributed processes at runtime, the appropriate mechanism for implementation involves local persistent storage. The appropriate form of local storage depends on the local host architecture and may be file- or host-DBMS oriented. It is important that such parameters are not stored in source code that requires build processes for modification.

For **SOA** services, configuration parameters relating to invoked services should not be service-host-specific at the invoking client application.

Derived From:

G1119

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1205.1]

Are there any user-modifiable configuration parameters hard coded in the non-auto-generated files?

Procedure:

Inspect the code for constant strings or constants that contain configuration parameters.

Example:

None

G1208

Statement:

Add new functionality rather than redefining existing interfaces in a manner that brings incompatibility.

Rationale:

By not replacing old methods of objects, library functionality consumers can continue to operate and not be forced to upgrade.

Derived From:

G1004

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1208.1]

Are methods that are being replaced marked with deprecated tags?

Procedure:

Check revision history to make sure that methods are deprecated and not removed unless they have expired. "Expired" means that they have passed the expected shelf life, as defined by the project standards or other standards documentation.

Example:

None

2) Test: [G1208.2]

Do new methods being added contain information on methods they are replacing?

Procedure:

Check to make sure newly added methods contain information and rationale on the methods they are replacing.

Example:

None

G1209

Statement:

For Java, use **JDK** logging facilities.

Rationale:

Java has a built-in logging framework that is portable across platforms, projects, and installations.

Derived From:

G1010

Referenced By:

Java EE Environment

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1209.1]

Does the application use anything other than the specified logging frameworks?

Procedure:

Check for use of logging frameworks other than the JDK.

Example:

None

G1210

Statement:

For **.NET**, use Debug and Trace from the **`System.Diagnostics`** **namespace**.

Rationale:

.NET has a built-in logging framework that is portable across .NET projects and installations.

Derived From:

G1010

Referenced By:

.NET Framework

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1210.1]

Does the application use anything other than the specified logging frameworks?

Procedure:

Check for use of logging frameworks other than **`System.Diagnostics`**.

Example:

None

G1213

Statement:

Provide an architecture design document.

Rationale:

An architectural design document provides evaluators with a roadmap of the application. This helps evaluators verify that the application follows guidance such as using the Model View Controller model.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1213.1]

Do the project deliverables for evaluation include a document that contains the architectural design of the application?

Procedure:

See if an architectural design document exists.

Example:

None

G1214

Statement:

Provide a document with a plan for **deprecating** obsolete **interfaces**.

Rationale:

This information allows users to phase out deprecated interfaces. For instance, Sun plans to maintain backward compatibility for the **JDK** for seven years. This means developers can count on deprecated methods not being removed for seven years.

Derived From:

G1004

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1214.1]

Do the project deliverables for evaluation include a document that contains a plan for deprecating obsolete interfaces?

Procedure:

See if a document with a plan for deprecating obsolete interfaces exists.

Example:

None.

G1215

Statement:

Provide a coding standards document.

Rationale:

The standards ensure a consistent code base. A coding standards document defines rules to keep code readable and maintainable.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1215.1]

Do the project deliverables for evaluation include a coding standards document?

Procedure:

See if a coding standards document exists.

Example:

None

G1216

Statement:

Provide a software release plan document.

Rationale:

The release plan document ensures that there is a formal process for releasing the software. It includes a description of how to acquire the software from the software configuration management (SCM) repository and how to build, label, and release it.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1216.1]

Do the project deliverables for evaluation contain a release plan document?

Procedure:

See if a software release plan exists.

Example:

None

G1217

Statement:

Develop and use externally configurable components.

Rationale:

To be portable and to accommodate reuse, components must be configurable using external descriptors usually defined in **XML**. Examples of things that might need to be configured include the following:

- A data source for the component to obtain a Java Database Connection (**JDBC**)
- The location of a service with which the component must communicate
- The location of implementation classes that the component uses

Derived From:

G1002

Referenced By:

Implement a Component-Based Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1217.1]

Are deployment descriptors used?

Procedure:

Check for the existence of deployment descriptors in the appropriate directories. Usually the file is named **`web.xml`**.

Example:

None

G1218

Statement:

Use a build tool that supports operation in an automated mode.

Rationale:

During testing, human interaction can be a cause of error and unrepeatable results. Operating in automated mode can eliminate these errors.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1218.1]

Does the tool have a build all target?

Procedure:

Check the build scripts or descriptors of the build tool for the ability to build the entire project, system, or application.

Example:

None

G1219

Statement:

Use a build tool that checks out files from configuration control.

Rationale:

To make sure all the parts of the build are under configuration control, compare all files with the configuration baseline, and download the appropriate files.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1219.1]

Does the tool have a checkout target?

Procedure:

Check the build scripts or descriptors of the build tool for the ability to check out the entire project, system, or application.

Example:

None

G1220

Statement:

Use a build tool that **compiles** source code and dependencies that have been modified.

Rationale:

To limit the changes made between builds, only compile code that has been modified. If there are no intermediate files, then compile all files.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1220.1]

Does the tool have a compile target?

Procedure:

Check the build scripts or descriptors of the build tool for the ability to compile the entire project, system, or application.

Example:

None

2) Test: [G1220.2]

Do all the intermediate files (e.g., **`.obj`** or **`.class`**) have the same date and time stamps?

Procedure:

Scan the files for date and time stamps.

Example:

None

G1221

Statement:

Use a build tool that creates libraries or archives after all required compilations are completed.

Rationale:

Libraries should be able to be recreated independently of any executables and should always verify that any intermediate files are not stale.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1221.1]

Does the tool have a generate library target?

Procedure:

Check the build scripts or descriptors of the build tool for the ability to generate the composing libraries or archives.

Example:

None

G1222

Statement:

Use a build tool that creates executables.

Rationale:

An executable is dependent on many files, including source files, intermediate files, and libraries or archives. The building of the executable must support a control process that includes configuration management, compiling, and testing.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1222.1]

Does the tool have an executable target?

Procedure:

Check the build scripts or build tool descriptors for the ability to build the executables for the entire project, system, or application.

Example:

None

G1223

Statement:

Use a build tool that is capable of running unit tests.

Rationale:

All code should be able to be tested independently of creating intermediate files, libraries, or executables.

Tests should be unit tests as well as system-level tests.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1223.1]

Does the tool have a test target?

Procedure:

Check the build scripts or descriptors of the build tool for the ability to test the entire project, system, or application.

Example:

None

G1224

Statement:

Use a build tool that cleans out intermediate files that can be regenerated.

Rationale:

For security reasons, all files that comprise the build need to be under configuration control. Cleaning out all files is essential in ensuring that only approved code is incorporated into the build.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1224.1]

Does the tool have a clean target?

Procedure:

Check the build scripts or descriptors for the build tool for the ability to remove the entire project, system, or application files.

Example:

None

G1225

Statement:

Use a build tool that is independent of the **Integrated Development Environment**.

Rationale:

Some build tools are tightly coupled with an **Integrated Development Environment** (**IDE**) that causes vendor lock-in and license issues when the software is delivered to the Government.

Derived From:

G1190

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1225.2]

Is the build tool one of the recognized standards, such as ant?

Procedure:

Check for files named **`build.xml`**.

Example:

None

2) Test: [G1225.3]

Is the build tool one of the recognized standards, such as **`make`** or **`nmake`**?

Procedure:

Check for files with the name **`makefile`**.

Example:

None

3) Test: [G1225.1]

Does the build tool require a license?

Procedure:

Check for files with the name **`makefile`**.

Example:

None

G1236

Statement:

Do not hard-code the **endpoint** of a **Web service** vendor.

Rationale:

An endpoint is the URL or location of the **Web service** on the **Internet**. A major benefit of Web services is the ability to relocate a Web service to another location or dynamically discover and use a Web service using registry facilities. Some Web service vendors hard-code the URL of the Web service which causes maintenance and portability problems.

Referenced By:

Insulation and Structure

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1236.1]

Are there any hard-coded Web service vendor endpoints in the client code?

Procedure:

Parse the code and look for hard-coded endpoints. These endpoints look just like a normal HTTP Web address.

Example:

None

G1237

Statement:

Do not hard-code the configuration data of a **Web service** vendor.

Rationale:

Some vendors generate code that passes Web service vendor-specific configuration data during initialization or startup. This reduces the portability of the code and can cause maintenance problems later.

Referenced By:

Insulation and Structure

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1237.1]

Is there any Web service vendor-specific configuration data in the client code?

Procedure:

Parse the code and look for hard-coded configuration data that might be used to configure the vendor's Web service.

Example:

None

G1239

Statement:

Use design patterns (e.g., **facade**, **proxy**, or **adapter**) or property files to isolate vendor-specifics of vendor-dependent connections to the enterprise.

Rationale:

This isolation increases maintainability. Guidance G1071 asserts that vendor-neutral connection mechanisms should be used. When vendor-specific connection mechanisms are unavoidable, this guidance will apply.

Derived From:

G1071

Referenced By:

JNDI Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1239.1]

Is the connection mechanism vendor-dependent?

Procedure:

Examine the source code for vendor-specific imports or includes.

Make sure that all references to the vendor-specific connection mechanisms are isolated to a single class (like a helper) or set of methods that are used as part of an isolation design pattern such as facade, proxy, or adapter.

Also, look for hard-coded vendor-specific connection strings.

Example:

None

G1245

Statement:

Isolate the Web service portlet from platform dependencies using the Web Services for Remote Portlets (**WSRP**) Specification protocol.

Rationale:

The OASIS **WSRP** 1.0 Specification accounts for the fact that **producers** and **consumers** may be implemented on very different platforms, such as a Java EE-based Web service, a Web service implemented on the Microsoft .Net platform, or a **portlet** published directly by a **portal**.

Referenced By:

Web Portals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1245.1]

Does the Web service implement the WSRP Markup interface?

Procedure:

Look for the definition of the **`getMarkup`**, **`performBlockingInteraction`**, **`initCookie`** and **`releaseSessions`** methods as defined in the OASIS WSRP Markup API Specification.

Example:

```
public MarkupResponse getMarkup
   ( RegistrationContext registrationContext,
     PortletContext portletContext,
     RuntimeContext runtimeContext,
     UserContext userContext,
     MarkupParams markupParams
   ) throws java.lang.Exception
public void performBlockingInteraction
   ( RegistrationContext registrationContext,
     PortletContext portletContext,
     RuntimeContext runtimeContext,
     UserContext userContext,
     MarkupParams markupParams,
     InteractionParams interactionParams
   ) throws java.lang.Exception
public Extension[] initCookie
   ( RegistrationContext registrationContext
   )  throws java.lang.Exception
public Extension[] releaseSessions
   ( RegistrationContext registrationContext,
     java.lang.String[] sessionIDs
   ) throws java.lang.Exception
```

2) Test: [G1245.4]

Does the Web service implement the WSRP Registration interface?

Procedure:

Look for the occurrence of the **`getService`**, **`register`**, **`deregister`**, and **`modifyRegistration`** methods as defined in the OASIS WSRP Specification.

Example:

```
public static RegistrationService getService
   ( java.lang.String baseEndpoint
   ) throws java.lang.Exception
public RegistrationContext register
  ( java.lang.String consumerName,
    java.lang.String consumerAgent,
    boolean methodGetSupported,
    java.lang.String[] consumerModes,
    java.lang.String[] consumerWindowStates,
    java.lang.String[] consumerUserScopes,
    java.lang.String[] customUserProfileData,
    Property[] registrationProperties
  ) throws java.lang.Exception
public ReturnAny deregister
  ( java.lang.String registrationHandle,
    byte[] registrationState
  ) throws java.lang.Exception
public RegistrationState modifyRegistration
  ( RegistrationContext registrationContext,
    RegistrationData registrationData
  ) throws java.lang.Exception
```

3) Test: [G1245.2]

Does the Web service implement the WSRP Service Description interface?

Procedure:

Look for the occurrence of the **`getService`**, **`register`**, and **`getServiceDescription`** methods as defined in the OASIS WSRP Service Description API Specification.

Example:

```
public static ServiceDescriptionService getService
  ( java.lang.String baseEndpoint
  ) throws java.lang.ExceptionThrows:
jpublic ServiceDescription getServiceDescription
  ( RegistrationContext registrationContext,
    java.lang.String[] desiredLocales
  ) throws java.lang.Exception
```

4) Test: [G1245.3]

Does the Web service implement the WSRP Portlet Configuration interface?

Procedure:

Look for the occurrence of the **`getService`**, **`getPortletDescription`**, **`clonePortlet`**, **`destroyPortlets`**, **`setPortletProperties`**, **`getPortletProperties`** and **`getPortletPropertyDescription`** methods as defined in the OASIS WSRP Portlet Configuration API Specification.

Example:

```
public static PortletManagementService getService
  ( java.lang.String baseEndpoint
  ) throws java.lang.Exception
public PortletDescriptionResponse getPortletDescription
  ( RegistrationContext registrationContext,
    PortletContext portletContext,
    UserContext userContext,
    java.lang.String[] desiredLocales
  ) throws java.lang.Exception
public PortletContext clonePortlet
  ( RegistrationContext registrationContext,
    PortletContext portletContext,
    UserContext userContext
  ) throws java.lang.Exception
public DestroyPortletsResponse destroyPortlets
  ( RegistrationContext registrationContext,
    java.lang.String[] portletHandles
  ) throws java.lang.Exception
public PortletContext setPortletProperties
  ( RegistrationContext registrationContext,
    PortletContext portletContext,
    UserContext userContext,
    PropertyList propertyList
  ) throws java.lang.Exception
public PropertyList getPortletProperties
  ( RegistrationContext registrationContext,
    PortletContext portletContext,
    UserContext userContext,
    java.lang.String[] names
  ) throws java.lang.Exception
public PortletPropertyDescriptionResponse getPortletPropertyDescription
  ( RegistrationContext registrationContext,
    PortletContext portletContext,
    UserContext userContext,
    java.lang.String[] desiredLocales
  ) throws java.lang.ExceptionThrows
```

G1267

Statement:

Use industry standard HTML data entry fields on Web pages.

Rationale:

Macromedia Flash and Java Applets can also be used for data input but are not HTML standards and tend to decrease the maintainability of a Web site.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1267.1]

Do any Web pages have data entry fields?

Procedure:

Search all Web pages for the "applet" and "embed" tags. Load each page found in the search by loading and visually inspecting to see if Flash or Applets are used for data entry.

Example:

Correct Usage:

Person's Name:

Incorrect usage:

Applet	
Flash	

G1268

Statement:

Label all data entry fields.

Rationale:

A label provides the user with a brief description of the text to be entered. Labels are essential for a user to understand the data entry field.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1268.1]

Are all data entry fields labeled?

Procedure:

Search all Web pages for the word "form" and load each resulting Web page in a browser. Visually inspect each data entry field to make sure it has labels.

Example:

None

G1270

Statement:

Include scroll bars for text entry areas if the data buffer is greater than the viewable area.

Rationale:

Scroll bars provide a visual cue to the user that the text extends beyond the viewable area. Scroll bars will appear by default for an HTML text area.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1270.1]

Do any Web pages turn off scroll bars for text areas?

Procedure:

Search all Web pages and style sheets for the phrase "overflow:hidden" or a form thereof. This turns off scroll bars using styles, but only works in certain browsers. Make sure it is not used.

Example:

Correct Usage

Scroll bars should not be hidden.

Incorrect Usage

Inline style:

```
<html>
<body>
<form>
<textarea style="overflow:hidden"></textarea>
</form>
</body>
</html>
```

External style:

```
textarea.scroll {
   overflow:hidden;
}
```

G1271

Statement:

Provide instructions and **HTML** examples for all style sheets.

Rationale:

An instruction manual will enable developers to use the style sheet correctly and efficiently.

Referenced By:

Style Sheets
Browser-Based Clients

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1271.1]

Are instructions included for each style sheet provided?

Procedure:

Verify that a document is provided that contains instructions and example code for each style provided.

Example:

Correct usage:

```
Cascading style sheet:
.td-items {
   text-align:right;
}
```

Example of usage:

Incorrect usage:
No HTML example explaining style usage.

G1276

Statement:

Do not modify the contents of the Web browser's status bar.

Rationale:

Using the browser's status bar to display text unrelated to status affects interoperability because a user expects the status bar to provide status and nothing else.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1276.1]

Do any of the Web pages modify the browser status bar?

Procedure:

Search every Web page for the word "status" and visually inspect each of the search results to see if the status bar has been modified.

Example:

Correct usage:

```
Web pages contain no references to window.status
```

Incorrect usage:

```
window.status = 'text to display in status bar'
```

G1277

Statement:

Do not use tickers on a Web site.

Rationale:

Ticklers can irritate the user and use unnecessary bandwidth.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1277.1]

Do any Web pages contain scrolling text?

Procedure:

Most tickers are written using Applets or Flash. Search all Web pages for the "applet" and "embed" tags. Load each page found in the search and visually inspect to make sure no tickers exist.

Example:

Correct usage:

```
No applet or flash references contain tickers.
```

Incorrect usage:

Applet:
```
applet code="myticker.class" width="200" height="200"
```
Flash:
```
embed src="myticker.swf" width="200" height="200"
```

G1278

Statement:

Use the browser default setting for links.

Rationale:

Browsers underline links by default. Do not rely on "mouse over" to identify links. Using mouse over to designate links can confuse and slow down infrequent users because they are uncertain which links perform which functions.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1278.1]

Do any Web pages or style sheets modify the browser default settings for links?

Procedure:

Search all the Web pages and style sheets for "A:link," "A:visited" and "A:active." Inspect all search results and make sure none of them modify the "A:" items.

Example:

Correct usage:

```
Web pages and style sheets should have no reference to A:link, A:visited or A:active.
```

Incorrect usage:

```
A:link, A:visited, A:active {
   text-decoration:none;
}
```

G1283

Statement:

Use linked style sheets rather than embedded styles.

Rationale:

Only by referencing an external file will you be able to update the look of an entire Web site with a single change. Also, by pulling style definitions out of the pages, they (web pages) will be smaller and faster to download.

Referenced By:

Style Sheets
Browser-Based Clients

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1283.1]

Does a Web page use the LINK tag to include external style sheets instead of embedding styles?

Procedure:

View the source of the HTML page. The header tag (head) should contain links to external style sheet (.css) files. The header tag should not contain any style tags.

Example:

Correct usage:

External style:

```
<head>
  <link rel=stylesheet href="style.css" type="text/css" media=screen>
  <link rel=stylesheet href="basic.css" type="text/css" media=screen>
</head>
```

Incorrect usage:

Embedded style:

```
<head>
  <style type="text/css">
      td {
      background:#ff0;
      }
  </style>
</head>
```

G1284

Statement:

Use only one font for body text.

Rationale:

Users may not have a wide variety of fonts available in their browser, so it is best to use a single, common font. The general standard is to make body text sans serif since most people find sans serif fonts easier to read on monitors and **serif** fonts better for printed materials.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1284.1]

Does the HTML or style sheet refrain from using more than one font?

Procedure:

Search all Web pages and style sheets for the word "font." Make sure only one type of font is used for body text. May need to visually inspect Web pages to see if a defined font style is used within the body.

Example:

Correct usage:

Cascading style sheet:

```
body.main {
   font:sans-serif;
}
```

HTML:

Incorrect usage:

Several font styles are used within a body.

G1285

Statement:

Use **relative font sizes**.

Rationale:

Relative font sizes make Web sites more accessible and support meeting the requirements of Section 508 of the Rehabilitation Act of 1973. Relative font sizes allow for a low-vision user to enlarge the size of the text. Relative font sizes also support maintainability by not hard coding fixed **font sizes**.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1285.1]

Are any absolute font sizes utilized?

Procedure:

Search all Web pages and style sheets for the word "font." Inspect the results to make sure no fixed fonts are used (e.g., 12pt).

Example:

Correct Usage

Relative or no font sizes settings are used.
Cascading style sheets:

```
p {
   font-size:200%;
}
p {
   font-size:2em;
}
```

Incorrect Usage

Cascading style sheets:

```
p {
   font-size:12pt;
}
```

HTML (the font attribute should not be used at all within HTML code, only external style sheets):

G1286

Statement:

Provide text labels for all buttons.

Rationale:

Users need to understand the purpose of all buttons. In some cases an image on the button is not sufficient to convey meaning. Screen scrapers used by the visually impaired work better when text labels are available for buttons

In cases where icons serve as buttons in order to fit within a small display device (such as a personal digital assistant), providing an option to enable text labels (or providing alternate attributes in the case of Web-based interfaces) supports screen scrapers.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1286.1]

Do all buttons have associated text labels?

Procedure:

Inspect the user interface to verify text labels are available for all buttons.
Text labels may optionally be displayed:
- on or near the button
- as a tooltip when the user hovers over a button
- as part of a help system where a user clicks and identify tool and then clicks a button.

Button label text may not be enabled by default on all applications, especially systems with small resolution screens such as PDAs.

Example:

Correct usage:

Incorrect usage:

Using images only:

G1287

Statement:

Provide feedback when a transaction will require the user to wait.

Rationale:

Users may think that the application has stopped running or is malfunctioning.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1287.1]

Does the application provide feedback during long processes?

Procedure:

Run the application and observe any processes that take longer than 10 seconds to complete. Observe if any status indication is provided to alert the user of the status.

Example:

None

G1292

Statement:

Use text-based Web site navigation.

Rationale:

Text-based navigation works better than image-based navigation because it enables users to understand the link destinations. Users with text-only browsers and browsers with deactivated graphics can see only text-based navigation options.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1292.1]

Are there any instances where graphics are used for navigation?

Procedure:

Visually inspect all Web pages and make sure navigation elements are textual.

Example:

None

G1293

Statement:

Use descriptive labels for all clickable graphics.

Rationale:

Clickable images generally confuse users, especially images that contain only graphics. Some that contain both graphics and words are also confusing because users do not know if the images are clickable without using the mouse pointer.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1293.1]

Do Web pages contain clickable images?

Procedure:

Search all Web pages for image ("img") tags embedded inside link ("a") tags. Visually inspect each image found in the search and make sure there is an associated text description.

Example:

Correct Usage

```
Click myimage to go to www.mywebsite.com
<a href="www.mywebsite.com"><img src="myimage.gif"></a>
```

Incorrect Usage

G1294

Statement:

Provide a site map on all Web sites.

Rationale:

A site map shows explicit organization of the site. Inexperienced users do not readily form a mental model of the way that information is organized in a Web site, making it hard for them to recover from navigational errors.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1294.1]

Does the Web site have a site map?

Procedure:

Search all Web pages for anything with the name "sitemap," "site map" and "map." Visually inspect the search results to make sure a site map is included.

Example:

None

G1295

Statement:

Provide redundant text links for linked images and each active region of an image map.

Rationale:

Redundant text links for linked images let users navigate the Web site even if their browser cannot display images.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1295.1]

Do any Web pages contain image maps or linked images?

Procedure:

Search all Web pages for images and visually inspect to make sure redundant text links exist for all active regions on image maps and redundant text links exist for all linked images.

Example:

Correct Usage

Image map:
Redundant text links for image map:
Linked image:
Redundant text link for linked image:

Incorrect Usage

No redundant text links exist for linked images or image maps.

G1300

Statement:

Secure all **endpoints**.

Rationale:

Something is only as secure as its weakest link. Therefore, all access points in an application should be secured. An endpoint is defined as an entry or an exit point of an application. Any access point can be vulnerable to attacks. For instance, if an application file reads configuration settings from a properties file, that file can be corrupted or incorrectly configured. This can cause incorrect behavior in the application. Also if component, module or application provides remote access or is part of any inter-process communications, these areas are vulnerable to attacks. For instance, if the application provides an external socket interface, does it validate commands being sent by the client?

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1300.2]

Does the application handle invalid configuration, provide appropriate defaults, and protect sensitive data?

Procedure:

Check application processing of data files (configuration files, properties files, preferences, XML, etc.).

Example:

2) Test: [G1300.1]

Does the application properly handle security when dealing with externally accessible API(s) and external ports?

Procedure:

Verify sensitive data is protected, and verify all network base protocols validate commands and values.

Example:

G1301

Statement:

Practice layered security.

Rationale:

An application with layered security provides more protection against attacks. Combining multiple layers of security defenses can provide additional protection when one layer is broken.

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1301.1]

Do internal and external API(s) perform security checks?

Procedure:

Make sure layers of API(s) starting from externally accessible API(s) down through the layers of internally accessible API(s) provide sufficient security checks. For example, does each layer of the API perform data validation? If internal API is calling remote services, is the data sufficiently protected from snoopers (e.g., use of secure sockets)?

Example:

None

2) Test: [G1301.2]

Does the application handle security when processing data files?

Procedure:

Embed all application specific resources such as graphics, internal application configuration files such as internationalization properties/resources, XML files as part of a signed application deployment file (.jar, .exe, etc.).

Example:

None

G1302

Statement:

Validate all inputs.

Rationale:

Input validation should not be limited to the presentation tier; rather, all external APIs should validate inputs prior to use. This can prevent many attacks including SQL Injection, Cross-Site Scripting, Buffer Overflows, and Denial of Service.

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1302.2]

Does the application provide proper handling for null input?

Procedure:

Check application handling of null values.

Example:

None

2) Test: [G1302.1]

Does the application use prefix or postfix validation (asserts) to verify input parameters?

Procedure:

Check application range validation of externally accessible API(s).

Example:

None

G1304

Statement:

Unit test all code.

Rationale:

A high percentage of all security violations can be attributed to inadequate or non-existent unit testing. Hackers can take advantage of these.

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1304.1]

Does the project unit test the code base?

Procedure:

Use a coverage tool to determine how much of the project's code have been tested.

Check for use of a unit testing framework (JUnit for example).

Example:

None

G1305

Statement:

Ensure the separation of **encrypted** and unencrypted information.

Rationale:

Not separating encrypted and unencrypted information can cause the application to incur performance hits due to unnecessary encryption. It can also cause inconsistent application processing.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1305.1]

Does the data model separate sensitive data from other data?

Procedure:

Check **UML** or entity diagram to ensure that separate components or entities are used to defined sensitive data.

If annotation support is provided via **XML**, ensure that the data is properly labeled (XML attribute) with correct security attributes.

Example:

None

G1306

Statement:

Identify and **authenticate** users of the application.

Rationale:

This ensure there is some traceability and also provides the first in a multilayer security system.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1306.2]

Does the application authenticate with another service (**LDAP**, database or simple password)?

Procedure:

Inspect application code to ensure that the user is authenticated against an LDAP, database or simple password service.

Example:

None

2) Test: [G1306.1]

Does the application require user certificates?

Procedure:

Ensure the application is setup to require client side certificates. This can be done easily by using a machine without any DoD client certificates installed and attempting to access the application.

Example:

None

G1307

Statement:

Provide a security policy file.

Rationale:

Security should not be an afterthought after application design and implementation. A security policy file can go along way in ensuring that application security has been part of the design and implementation of the application. A security policy file can identify all the security measures that the application has laid out.

Referenced By:

General Application Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1307.1]

Does the project have Security Policy File?

Procedure:

Check for the existence of a Security Policy file.

Example:

None

G1308

Statement:

Make applications handling unclassified medium value information in Moderately Protected Environments, unclassified high value information in Highly Protected Environments, and discretionary access control of classified information in Highly Protected Environments **Public Key Enabled** to interoperate with DoD **Class 3 PKI**.

Rationale:

The guidance defines the application types required to support DoD class 3 PKI.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1308.1]

Is the application using a DoD class 3 PKI approved FIPS 140-1 cryptographic module?

Procedure:

Check the cryptographic module to see if it is FIPS 140-1 compliant.

Example:

None

2) Test: [G1308.2]

Procedure:

Example:

G1309

Statement:

Make applications handling high value unclassified information in Minimally Protected environments **Public Key Enabled** to interoperate with DoD **Class 4 PKI**.

Rationale:

The guidance defines the application types require to support DoD class 4 PKI.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1309.1]

Is the application using a DoD class 4 PKI approved FIPS 140-1 cryptographic module?

Procedure:

Check cryptographic module to see if it is FIPS 140-1 compliant.

Example:

G1310

Statement:

Protect application cryptographic objects and functions from tampering.

Rationale:

If cryptographic objects such as private keys, key store, and CA trusted certificates are not protected, the system is not secure.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1310.1]

Are cryptographic objects protected?

Procedure:

Check that key stores, private keys, and trust points are protected.

Check that an established procedure is followed for creating and documenting the creation of keys.

Check that an established procedure is followed for obtaining certificates.

Example:

Use High Security Level setting in Internet Explorer to ensure password protection is used. See https://infosec.navy.mil/PKI/certs.html for software certificate steps. See https://infosec.navy.mil/PKI/cac.html for CAC.

G1311

Statement:

Use **LDAP**, **HTTP**, or **HTTPS** when applications communicate using DoD **PKI**.

Rationale:

These are the DoD approved protocols and the only supported ones.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.3.2.1, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1311.1]

Does the application use only LDAP, HTTP, or HTTPS protocols to communicate when using DoD PKI?

Procedure:

Configure application to use HTTP.

Have application access the DoD PKI GDS Directory (DoD411.chamb.disa.mil) via HTTP.

Repeat access to GDS using HTTPS and LDAP protocols.

Example:

None

G1312

Statement:

Make applications capable of being configured for use with DoD **PKI**.

Rationale:

Applications must be able configurable to accept certificates, load key stores with private key, add trust points, etc.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.4, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1312.1]

Does the application provide external configuration files, properties files, and configuration applications?

Procedure:

Check to make sure the application is configurable to accept certificates, load key stores, and add trust points; this may involve inspecting user and administrator manuals.

Example:

None

G1313

Statement:

Provide documentation for application configuration and setup for use with DoD **PKI**.

Rationale:

If the application can not be configured or setup correctly, the application is insecure. Without detail documentation, personnel with little knowledge of security or PKI will have little chance of keeping the overall system secure.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.5, Version 1.0, 13 July 2000.*

Referenced By:

Public Key Infrastructure (PKI) and PK Enable Applications

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1313.1]

Is there documentation (such as Standard Operating Procedures [SOPs]) on how to configure and setup the application to interoperate within the DoD PKI?

Procedure:

Verify by inspection of the SOPs and by a demonstration that the application performs as documented when the configuration guidance is followed.

Example:

Most application manuals have detailed instructions in enabling PKI (either under the heading "enabling SSL" or "certificates").

G1314

Statement:

Provide applications the ability to import and export keys (software certificates only).

Rationale:

The whole PKI system is predicated on the use of public-private key pair. The ability to import and use private keys is critical to a functional PKI application.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.5, Version 1.0, 13 July 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1314.1]

Is the application able to import and export keys associated with standard certificates for individuals?

Procedure:

Have the application import and export at least one set of keys and certificates for each certificate type supported by the application. Demonstrate interoperability by performing representative subscriber and relying party operations with each certificate type and its related keys.

Note: *Verify the correctness of the exported file through analysis.*

Example:

Internet Explorer can import/export certificates using Tools > Internet Options. Click on Internet tab and then click on Certificates link. Import/Export options are located here.

UNIX-based Web server keys are exported by making a copy of the keys file and placing it in a safe location.

G1315

Statement:

For applications, use key pairs and **Certificates** created for individuals using DoD **PKI** methods and procedures defined by the DoD Class 3 Public Key Infrastructure Interface Specification and the Personal Information Exchange Syntax Standard.

Rationale:

DoD PKI supports these standards for importing keys and certificates. If the key or certificate is not created or issued by approved DoD Certificate architecture, it can not be trusted to interoperate within the DoD network.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.5, Version 1.0, 13 July 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1315.1]

Can the application import and export keys associated with standard certificates for individuals?

Procedure:

Verify by importing and exporting to DoD PKI key store.

Access the application using a DoD PKI Class 3 Certificate.

Example:

For servers, verify that the application requires client side authentication. Access the application server using a DoD PKI certificate.

G1316

Statement:

Ensure that applications protect **private keys**.

Rationale:

In order for the PKI system to stay secure, the private key must not be compromised. Protecting the private key helps prevent attackers from decrypting secured data communications.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.5, Version 1.0, 13 July 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1316.1]

Does the application use and store the private key securely?

Procedure:

Check for the following:

- all copies of the private key destroyed when private key operation is complete; for example, check that the private key does not stay in application memory permanently

- the private key is password protected with a strong password
- the keystore is password protected with a strong password

Example:

Attempt to view the contents of the private key using a document viewer program.

G1317

Statement:

Ensure applications store **Certificates** for subscribers (the owner of the **Public Key** contained in the Certificate) when used in the context of signed and/or encrypted email.

Rationale:

This will allow other parties to use the public key to encrypt messages sent to the application.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document. Section (4.5), Version 1.0, July 13, 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1317.1]

Is the public key available from the Directory Server application?

Procedure:

See if it is possible to extract the public key certificate from the Directory Server application.

Example:

None

G1318

Statement:

Develop applications such that they provide the capability to manage and store **trust points** (**Certificate Authority** Public Key **Certificates**).

Rationale:

This will ensure the certificate is valid and expedite verification of the certificate.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1318.1]

Is the Certificate Authority public key available from the application?

Procedure:

View the application's trust list to verify DoD PKI Class 3 CA certificates are present.

Example:

For Internet Explorer, view the DoD PKI Class 3 CA certificates by selecting **`Tools`**>**`Internet Options`**. Click on the **`Internet`** tab and then click on the **`Publishers`** button. Click on the **`Trusted Root Certification Authorities`** tab and scroll down to verify that the DoD PKI Class 3 CA certificates are present.

Web server Certificate Authority certificates can usually be viewed by the application's GUI. If a GUI is not offered, reference the application's manual concerning certificate management.

G1319

Statement:

Ensure applications can recover data encrypted with legacy keys provided by the DoD **PKI** Key Recovery Manager (**KRM**).

Rationale:

Applications may have the need to decrypt legacy information that the application originally encrypted.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Key Management

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1319.1]

Is the application able to recover legacy encrypted data?

Procedure:

Acquire the legacy key and demonstrate the ability
to decrypt data that is encoded by that key.

Example:

None

G1320

Statement:

Develop applications such that they use 128 bit **symmetric keys**, 1024 bit **asymmetric keys**.

Rationale:

Strong encryption helps to prevent unauthorized data decryption using modern day resources.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1320.1]

Are encryption levels adequately configured?

Procedure:

Check the server configuration and verify that the ciphers being used are 1024 and not 512.

Example:

Verified Web server ciphers under the SSL portion of the configuration pages of the administration server.

For Internet Explorer 5.0 and above, click the **`Help`** menu and then click the **`About Internet Explorer`** option. The About box will list the Cipher Strength.

2) Test: [G1320.2]

Is the application using domestic (U.S.) grade ciphers?

Procedure:

Verify that the application supports domestic (U.S.) grade ciphers.

Example:

None

G1321

Statement:

Enable applications to be capable of performing **Public Key** operations necessary to verify signatures on DoD **PKI** signed objects.

Rationale:

An application must verify the digital signature and check its validity against the current **Certificate Revocation List** (**CRL**) maintained by an on-line repository (e.g., **Online Status Check Responder** or **OSCR**).

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1321.1]

Does the application verify signed objects?

Procedure:

Check that the application validates signed objects against DoD root certificates.

Check that the signing certificate has not been revoked by checking against Certificate Revocation Lists or using the Online Certificate Status Protocol (OCSP).

Example:

Make a back-up copy of the certificate. For Windows based applications, stop the application and edit the signature of the certificate and save the certificate. Start the application back up. The application should fail to start as the signature check will fail.

For validity checking, confirm a validity check of the certificate was performed by viewing the application's audit log.

G1322

Statement:

Ensure that applications that interact with the DoD **PKI** using **SSL** (i.e., **HTTPS**) are capable of encrypting and decrypting data using the **Triple Data Encryption Algorithm** (**TDEA**).

Rationale:

Applications should use cryptographic modules approved under [FIPS 140], Level 1.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1322.1]

Does the application use TDEA for encrypting and decrypting data?

Procedure:

Inspect the application's configuration file to confirm that TDEA is used for encrypting and decrypting data.

Example:

Most server based applications have cipher related information stored under SSL, certificates, or security. Verify that the application is using TDEA.

G1323

Statement:

Generate random **symmetric encryption** keys when using symmetric encryption.

Rationale:

If the application can not generate random keys, then it is vulnerable to attacks if attackers can determine the algorithm for generating the random symmetric encryption keys.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1323.1]

Does the application generate random symmetric encryption keys?

Procedure:

Verify that the random seed is generated (e.g., by viewing the application's vendor documentation).

Example:

Most server based applications either user MOD_SSL or OPEN_SSL. These two toolkits properly use random seed generators.

Apache based servers may require the administrator to type random keystrokes on the keyboard. This process is generating the random seed.

G1324

Statement:

Protect **symmetric keys** for the life of their use.

Rationale:

Symmetric key encryption algorithms are based on trivially related keys for both encryption and decryption. The advantage of symmetric key encryption is that it is much less computationally intensive for encryption and decryption compared to asymmetric algorithms. The disadvantage is that the shared symmetric key must be kept secure during storage and transmission.

To prevent disclosure, new symmetric keys are often generated for each unique session and exchanged using another encryption algorithm. Store symmetric keys that are used long term carefully to prevent disclosure.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1324.1]

Are symmetric keys stored in unprotected locations?

Procedure:

Check for hard coded symmetric keys in source code or files with weak permissions.

Example:

Symmetric keys should be generated for each session and destroyed when the session is destroyed, never stored in a file with weak permissions or hard coded in source code.

G1325

Statement:

Encrypt **symmetric keys** when not in use.

Rationale:

Symmetric keys enable both sides of the conversation to have knowledge of the key for encryption. It can not be given out freely, which means if it is going to be stored for repeated use, it should be encrypted first before storage.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1325.1]

Does the application encrypt symmetric keys when not in use?

Procedure:

Check that the application encrypts symmetric keys during storage.

Example:

G1326

Statement:

Ensure applications are capable of producing Secure Hash Algorithm (**SHA**) digests of **messages** to support verification of DoD **PKI** signed objects.

Rationale:

Symmetric keys enable both sides of the conversation to have knowledge of the key for encryption. It can not be given out freely, which means if it is going to be stored for repeated use, it should be encrypted first before storage.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Encryption Services

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1326.1]

Does the application use SHA digest?

Procedure:

Visually validate that the SHA digest is used for symmetric keys.

Example:

Most application servers allow one to configure the hash to SHA1. Please note that the default for most applications is MD5.

G1327

Statement:

Enable an application to request and obtain new **Certificates** for subscribers.

Rationale:

If the application generates subscriber keys, the application shall demonstrate the ability to generate keys, request new certificates, and obtain new certificates through interaction with the DoD PKI. If the generated keys are for encryption applications, the application shall demonstrate its ability to provide keys to the DoD PKI KRM.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.3.2.2, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1327.1]

Can the application request and obtain new certificates for subscribers?

Procedure:

For application servers, verify that the application can successfully request a certificate via the appropriate certificate request page from a DoD PKI CA.

For application servers, verify that the application can successfully download an issued certificate from a DoD PKI CA.

Example:

Instructions in obtaining a DoD PKI certificate for a user are available at https://infosec.navy.mil/PKI/users.html.

Instructions for obtaining a DoD PKI certificate for web servers including Netscape, Lotus, and IIS is available at https://infosec.navy.mil/PKI/training.html.

G1328

Statement:

Enable an application to retrieve **Certificates** for use, including relying party operations.

Rationale:

The ability to retrieve certificates from DoD certificate repositories further ensures the authenticity of the certificate .

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.3.2.3, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1328.1]

Can the application retrieve **Certificates** from a DoD PKI certificate repository?

Procedure:

Verify that the application can communicate with a DoD PKI certificate repository such as GDS.

Example:

This test procedure is only required for applications that must send encrypted e-mail. For this scenario, assume that Outlook is used; instructions for using Outlook 2000 are available at https://infosec.navy.mil/PKI/Outlook_2000_0704.pdf

G1330

Statement:

Ensure applications are capable of checking the status of **Certificates** using a **Certificate Revocation List** (**CRL**) if not able to use the **Online Certificate Status Protocol** (**OCSP**).

Rationale:

Applications must verify the validity of the certificate prior to establishing trust with another entity. CRL is the legacy mechanism for validating certificates. Applications should favor OSCP for new development.

Applications operating in environments with network connectivity to a CRL distribution point should be able to obtain a current CRL. Applications should be able, without user intervention, to obtain a current CRL to check the status of a certificate that contains a CRL distribution point extension. Applications with network connectivity unable to find CRL distribution points automatically should be capable of being configured with a distribution point that the application then uses to obtain CRLs as needed.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.3.2.4.1, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1330.1]

Can the application perform Certificate status checking with a CRL?

Procedure:

Verify that the application can download a CRL successfully .

Example:

Visually inspect the application is configured to use CRLs for validity checking. This can be achieved by looking at the directory in which the application stores the CRLs.

G1331

Statement:

Ensure applications are able to check the status of a Certificate using the **Online Certificate Status Protocol** (**OCSP**).

Rationale:

Applications must verify the validity of the certificate prior to establishing trust with another entity. CRL is the legacy mechanism for validating certificates. Applications should favor **OCSP** for new development.

Applications may use an OSC responder to check the status of a particular certificate when the DoD has an operational responder. Applications shall prepare and transmit the request to the responder using HTTP in accordance with the DoD Class 3 PKI Infrastructure Interface Specification.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Section 4.3.2.4.2, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1331.1]

Can the application perform **Certificate** status checking with **OCSP**?

Procedure:

Verify that the application can performing OCSP queries to an **OSC** Responder successfully.

Example:

Visually inspect the application is configured to use OCSP for validity checking. This can be achieved by looking at the configuration file to see that the application is configured to use OCSP. One can also visually look at the application's log file to validate that the application is making OCSP queries.

G1333

Statement:

Only use a **Certificate** during the Certificate's validity range, as bounded by the Certificate's "Validity - Not Before" and "Validity - Not After" date fields.

Rationale:

Expired certificates should not be accepted except in cases where legacy data was archived.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1333.1]

Do the date and time of the use of the Certificate fall within the Certificate's validity period?

Procedure:

Visually inspect the certificate's validity dates. The certificate should be valid and not expired.

Example:

Each digital certificate has a lifetime. When viewing a certificate, the certificate will have a valid from date and a valid to date. The current date should fall within this range.

G1335

Statement:

Make applications capable of being configured to operate only with PKI Certificate Authorities specifically approved by the application's owner/managing entity.

Rationale:

Using approved PKI Certificate Authorities ensures certificate authenticity and ensures that the certificate is chained to the issuer.DoD trust points ensure certificates are chained to the issuer of the certificate and are authentic.

For example, DoD applications are configured to use DoD PKI Certificate Authorities only per the DoD Class 3 PKI - Public Key-Enabled Application Requirments Document Version 1.0, 13 July 2000.

Note: *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1335.1]

Is the application configured to operate only with approved PKI Certificate Authorities?

Procedure:

Visually inspect that only the DoD PKI certificates are trusted by the application.

Example:

Applications typically allow one to view the trust points via the administrative interface to the application. CA certificates are typically located under Certificate Management, SSL, or Security.

G1338

Statement:

Applications and **Certificates** need to be able to support multiple organizational units.

Rationale:

DoD requirements dictate that certificates shall support multiple organizational units.

> ***Note:*** *This guidance is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Referenced By:

Certificate Processing

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1338.1]

Can the application process a **Certificate** that contains multiple organizational units in the Distinguished Name?

Procedure:

Visually inspect the DoD PKI CA certificates stored in the application. You will notice that each certificate contains multiple organizational units (OU=DoD, OU=PKI)

Example:

The majority of certificate request forms do not contain entries for multiple organizational units. In this case, include all of the organizational unit information in the single line. For example, for Navy, please enter the following information next to the Organizational Unit line: Navy, OU=DoD, OU=PKI.

Once the certificate is issued, visually inspect this certificate to verify that the certificate contains these Organizational Unit values.

G1339

Statement:

Practice defensive programming by checking all method arguments.

Rationale:

Data validation is not limited to Graphical User Interfaces. API(s) and library functions are also susceptible to corruption. The integrity of application can benefit from identifying invalid data as early as possible.

Referenced By:

API Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1339.1]

Does the application perform range validation?

Procedure:

Check for unit tests.

Check thrown exceptions.

Purposely send invalid data to API(s) to test the integrity and handling of invalid data.

Example:

G1340

Statement:

Log all exceptional conditions.

Rationale:

Logging exceptional conditions can help to identify security problems, trace the source of the exception, and trigger security alerts.

Referenced By:

API Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1340.1]

Does the application perform logging of exceptional conditions?

Procedure:

Check exception handlers for logging support.

Example:

G1341

Statement:

Use a security manager support to restrict application access to privileged system resources.

Rationale:

Desktop applications by default do not install a security manager. Installing a security manager could prevent unsecured access to system resources such as network and file system. Desktop applications can benefit from using a security manager to ensure that system resources are protected.

Referenced By:

Java Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1341.1]

Does an installed security manager restrict application access to privileged system resources?

Procedure:

Check application main method for installation of a security manager.

Example:

G1342

Statement:

Restrict direct access to class internal variables to functions or methods of the class itself.

Rationale:

One of the primary tenets in Object Oriented Programming is encapsulation. Restricting access to internal variables not only secure the Class/Object against corruption (no data validation), it is also a maintenance issue. Hiding the implementation details allows the flexibility of underlying implementation to change.

Referenced By:

Java Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1342.1]

Do classes directly expose internal data members?

Procedure:

Make sure all internal class variables are declared private or protected.

Example:

G1343

Statement:

Declare classes final to stop inheritance and prevent methods from being overridden.

Rationale:

Utility classes and classes that do not intend to be extended (classes used for user authentication) should lock down their implementation. Locking implementation can prevent methods from being overridden. Not locking down implementation can cause corruption of internal class data or allow errant code to run. For example, imagine the possibility of a class that performs credit card processing that can be overridden.

Class implementation can be locked down by declaring the class or methods final.

Referenced By:

Java Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1343.1]

Are sensitive, security related, and utility classes declared final?

Procedure:

Check classes used in Security related processing (authentication, authorization) final keyword.

Check classes that have sensitive data (social security numbers, medical data, and salary information) for final keyword.

Check Utility classes for final keyword.

Example:

G1344

Statement:

Encrypt sensitive data stored in configuration or resource files.

Rationale:

Sensitive data used for application configuration files (XML), user profiles, or resource files should be protected from tampering. The sensitive data should be encrypted and or a message digest or checksum should be calculated to check for tampering. Application should handle generation, accessing and storing data to these files.

Referenced By:

Application Resource Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1344.1]

Is sensitive data in configuration files and user profiles?

Procedure:

Check properties files, XML configuration files or user profiles for sensitive data in the clear.

Check for an application to edit, and creation of the file.

Example:

G1346

Statement:

Audit database access.

Rationale:

Auditing is critical for data access traceability. If the RDBMS was attacked, auditing is essential not only for figuring out what had occurred but also to recover lost data. Database access auditing provides logs for each access or change to the database by a given user (or an IP address for systems without user authentication).

Often current middle tier technologies (e.g., J2EE, .Net, CORBA, etc.) share database connections and may only have a single database user. Thus the burden is on the middle tier to know the identity of each user and be able to pass this information on the database (e.g., design each table to have data items such as updated by, created by, etc.).

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1346.1]

Does the application database include actual user rather than database connection owner?

Procedure:

Check system documentation, database tables, and audit logs to verify that database access audit entries are created for each database access.

Example:

None

G1347

Statement:

Secure remote connections to database.

Rationale:

Just because the database is behind the corporate firewall does not mean someone inside the firewall cannot access or listen in on the wire.

Net-centricity implies that a database should be on the network and not constrained to be sitting behind an application server. This means that many unanticipated users may eventually access the database. Thus, database security should not be based on isolation.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1347.1]

Is data exchanged between the database and client secure?

Procedure:

Check for secure protocol (e.g., SSL) between application and database.

Check for secure data access by IP address.

Check for configuration in the database (user) which limits user from a specified host.

Example:

G1348

Statement:

Log database **transactions**.

Rationale:

Transaction logging is generally handled by the database management system and records all changes made to the database, critical for data recovery and traceability.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1348.1]

Are database transactions logged?

Procedure:

Commercial database management systems have a feature to log database transactions. Check to determine whether the feature has been turned on in the database management system.

Example:

G1349

Statement:

Validate all input that will be part of any dynamically generated **SQL**.

Rationale:

Not validating or filtering parameters used in dynamically generated SQL statements can lead to SQL injection attacks.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1349.1]

Does the database use filtering or data validation code?

Procedure:

Filter out character like single quote, double quote, slash, back slash, semi colon, extended character like NULL, carry return, new line, etc, in all input strings.

Example:

G1350

Statement:

Implement a strong password policy for **RDBMS**.

Rationale:

Clean database installation often contains no passwords for root users. Also, new user accounts often defaults to no password or standard password. Having no passwords allows users access any data. Database users should always be given strong passwords. This implies a non null password, locking unused user accounts and ensuring that system user accounts are not using default passwords

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1350.1]

Does the database user table include passwords?

Procedure:

Check for null or empty values for passwords in the user table.

Use a commercially available or open source default password analysis tool to ensure that all user accounts do not retain default passwords and to ensure that all passwords are strong.

Example:

G1351

Statement:

Enhance database security by using multiple user accounts with constraints.

Rationale:

Constrain access to individual tables and functions by creating multiple user accounts for an application and constraining the accounts to specific functions. As a general policy, user accounts should be constrained to the minimal required database access. For example, creation of a read only account should be constrained by granting only select on the tables of interest to the read only user. This aids in password management as well as limiting the potential impact of SQL injection attacks. By granting only insert on a table, for example, and not granting select, the user could in effect create a write only database.

Each application will have different requirements in regards to grants and access to tables. If one application is compromised, it will not affect the other applications.

It also has traceability to determine which application has allowed a security violation.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1351.1]

Does each database application user have account constraints in accordance with the user function?

Procedure:

Check each database application user to ensure that the account constraints are in accordance with the user function and do not have unwarranted privileges. For example, check that read only application user accounts have only read access enabled.

Example:

G1352

Statement:

Use database clustering and redundant array of independent disks (RAID) for high availability of data.

Rationale:

Database clusters combined with RAID technology (e.g., data striping and mirroring) can help ensure continued operation of a system that suffers hardware or software failure.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1352.1]

Is the system designed to support high availability?

Procedure:

Check for the existence of a cluster and/or failover capability.

Check for the existence of RAID data storage for the database.

Example:

G1356

Statement:

Use the **Simple Object Access Protocol** (**SOAP**) standard for all **Web services**.

Rationale:

The Web services security specifications are designed as an extension of SOAP. The specs are unusable without SOAP.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1356.1]

Does the Web service user generate SOAP formatted XML messages?

Procedure:

Generate a test message and check it for SOAP compliance.

Example:

2) Test: [G1356.2]

Does the Web service provider generate SOAP formatted XML?

Procedure:

Generate a test message and check it for SOAP compliance.

Example:

G1357

Statement:

Do not rely on transport level security like **SSL** or **TLS**.

Rationale:

Web services inherently involve multiple intermediaries between the message sender and the ultimate destination. The intermediaries may not use transport level security. SSL and TLS do not provide end-to-end security, only security at the transport layer and only point-to-point.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1357.1]

Does the Web service user generate encrypted XML messages?

Procedure:

Generate a test message and check it for encryption.

Example:

2) Test: [G1357.2]

Does the Web service provider generate encrypted XML messages?

Procedure:

Generate a test message and check it for encryption.

Example:

G1359

Statement:

Bind **SOAP Web service** security policy assertions to the service by expressing them in the associated **WSDL** file.

Rationale:

A Web service may be registered in zero, one, or multiple **UDDI** registries. By placing the security policy assertions in the Web service's WSDL file, they are readily available to all the consumers of the service regardless how the service was discovered

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1359.1]

Are Web service security policy assertions bound in the service WSDL file?

Procedure:

Check the Web Service's WSDL file for policy assertions.

Example:

None

G1361

Statement:

Place the service provider **canonicalization** method inside the Web Services Description Language (**WSDL**) file.

Rationale:

This assures that all users have a consistent view of all non-critical information like line breaks, tabs and closing tags.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1361.1]

Does the **WSDL** file contain a method?

Procedure:

Inspect the WSDL file for the word "canonicalization."

Example:

None

G1362

Statement:

Use very intensive input validation (using a **schema**).

Rationale:

Prevent malicious agents from compromising the integrity of a service.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1362.1]

Does the Web service provider validate incoming messages?

Procedure:

Identify the existence of an XML Schema file and examine code to verify that all incoming messages are checked to be XML Valid.

Example:

None

G1363

Statement:

Do not use clear text passwords.

Rationale:

Prevent a hacker from intercepting and seeing a real password.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1363.1]

Does the Web service user utilize a username/password token?

Procedure:

Generate a test message and check it for clear text passwords.

Example:

None

G1364

Statement:

Hash all passwords using the combination of a timestamp, a **nonce** and the password for each **message** transmission.

Rationale:

This Guidance helps to prevent unwanted interception or discovery of clear-text-hashed passwords.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1364.1]

Does the Web service user utilize a username/password token?

Procedure:

Generate a test message and check it for a username/password token and verify that is contains a timestamp entry and a nonce entry.

Example:

None

G1365

Statement:

Specify a timeout value for all security tokens.

Rationale:

Limit a hacker's ability to intercept and use the entire security token (username, password, timestamp, password) in his own message.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1365.1]

Does the Web service user utilize a timeout for each security token?

Procedure:

Generate a test message and check it to make sure a timeout is associated with each security token.

Example:

None

G1366

Statement:

Digitally sign all **messages**.

Rationale:

Prevent hackers from changing intercepting and modifying a message.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1366.1]

Does the Web service user digitally sign all messages?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

2) Test: [G1366.2]

Does the Web service provider digitally sign all messages?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

G1367

Statement:

Digitally sign **message** fragments that must not change during transport.

Rationale:

Signing message fragments allows the consumer of the message fragment to verify the message fragment has not changed since the producer signed the message fragment.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1367.1]

Do message fragments sent between producers and subscribers have digital signatures when the message content must remain unchanged during transport?

Procedure:

Check system requirments for message fragments that must be transmitted unchanged between the producer and consumer. For these message frangments, check that digital signature are used to detect changes to the message fragments.

Example:

None

G1368

Statement:

Digitally sign any part of a **message** that is not **encrypted**.

Rationale:

Prevent hackers from changing intercepting and modifying a message.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1368.1]

Does the Web service user digitally sign all message parts that are not encrypted?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

2) Test: [G1368.2]

Does the Web service provider digitally sign all message parts that are not encrypted?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

G1369

Statement:

Digitally sign all requests made to a security token service.

Rationale:

Prevent hackers from intercepting a message and requesting a security token.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1369.1]

Does the Web service user digitally sign all messages?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

2) Test: [G1369.2]

Does the Web service provider digitally sign all messages?

Procedure:

Generate a test message and check it for digital signatures.

Example:

None

G1370

Statement:

Digitally sign all **WSDL** files.

Rationale:

Prevent hackers from changing parts of the WSDL file.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1370.1]

Does the Web service provider digitally sign all its **WSDL** files?

Procedure:

Check WSDL files for digital signatures.

Example:

None

G1371

Statement:

Use the **Digital Signature Standard** for creating **Digital Signatures**.

Rationale:

Following Industry standards ensures interoperability.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1371.1]

Does the Web service user generate signatures using the Digital Signature Standard?

Procedure:

Generate a test message and check it for compliance with the Digital Signature Standard.

Example:

None

2) Test: [G1371.2]

Does the Web service provider generate signatures using the Digital Signature Standard?

Procedure:

Generate a test message and check it for compliance with the Digital Signature Standard.

Example:

None

G1372

Statement:

Use an X.509 **Certificate** to pass a **Public Key**.

Rationale:

This ensures that the owner passing the key is who he says.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1372.2]

Does the Web service provider send a public key as part of its messages?

Procedure:

Generate a test message and check it for an X.509.

Example:

None

2) Test: [G1372.1]

Does the Web service user send a public key as part of its messages?

Procedure:

Generate a test message and check it for an X.509.

Example:

None

G1373

Statement:

Encrypt all **messages** that cross an **IA** boundary.

Rationale:

Prevent hackers from reading sensitive information.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1373.1]

Does the Web service user encrypt all messages?

Procedure:

Generate a test message and check it for encryption.

Example:

None

2) Test: [G1373.2]

Does the Web service provider encrypt all messages?

Procedure:

Generate a test message and check it for encryption.

Example:

None

G1374

Statement:

Individually **encrypt** sensitive **message** fragments intended for different intermediaries.

Rationale:

Individually encrypting message fragments allows targeting individual fragments at different intermediaries along the message path to the final destination.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1374.1]

Are sensitive fragments of the message encrypted?

Procedure:

Observe messages that are sent to see if the sensitive fragments of the message are encrypted.

Example:

None

G1376

Statement:

Do not **encrypt** key elements that are needed for correct **SOAP** processing.

Rationale:

It is possible to encrypt the entire SOAP message, various portions of the SOAP message or the contents of the data transported within the SOAP message. Encrypting the entire SOAP message requires that any intermediate processing of the SOAP message requires decryption of the entire message.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1376.1]

Does the Web service user encrypt the entire message?

Procedure:

Generate a test message and check it to make sure the XML tags are not encrypted.

Example:

None

2) Test: [G1376.2]

Does the Web service provider encrypt the entire message?

Procedure:

Generate a test message and check it to make sure the XML tags are not encrypted.

Example:

None

G1377

Statement:

Use **LDAP** 3.0 or later to perform all connections to LDAP repositories.

Rationale:

Using industry-proven LDAP standards helpe ensure interoperability of the directory repository with its consumers. LDAP v3 addresses some of the limitations of LDAP v2 in the areas of internationalization and authentication. It also allows adding new features without also requiring changes to the existing protocol through the use of using extensions and controls while maintaining backward compatibility with LDAP v2.

Referenced By:

LDAP Security

Acquisition Phase:

Oversight

Evaluation Criteria:

1) Test: [G1377.1]

Check port 636 if supporting secure LDAP (SLDAP)

Procedure:

Test the connection using an SLDAP client.

Example:

None

G1378

Statement:

Encrypt communication with **LDAP** repositories.

Rationale:

Encryption of communication to LDAP servers helps prevent disclosure of data during transmission.

Referenced By:

LDAP Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1378.1]

Are connections to LDAP repositories encrypted?

Procedure:

Verify that connections to LDAP repository use Transport Layer Security (TLS) or Secure Sockets Layer (SSL).

Example:

G1379

Statement:

Use **SAML** version 2.0 for representing security assertions.

Rationale:

SAML 2.0 supports **XML** assertions for supporting cross domain access and Web services. The value of this type of access is that the passing of an assertion eliminates the need to create another account in another domain.

Referenced By:

Security Assertion Markup Language (SAML)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1379.1]

Can the SAML message be validated against SAML V2.0 schema?

Procedure:

Validate SAML message against SAML V2.0.

Example:

G1380

Statement:

Use the **XACML** 2.0 standard for **SAML**-based rule engines.

Rationale:

XACML-based rules can define the mechanism for creating the rule and policy set that enable meaningful authorization decisions. XAMCL is also integrated with **SAML** to support Role based access control or hierarchical resources, such as portions of XML documents.

Referenced By:

Security Assertion Markup Language (SAML)

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1380.1]

Does the SAML-based rules engine use the XACML 2.0 standard?

Procedure:

Emulate a rule and run against rule engine using SOAP messaging.

Example:

G1381

Statement:

Encrypt all sensitive persistent data.

Rationale:

When data is persisted, there is always a chance that the security of the system that stores the data may be compromised. To minimize the risk, all sensitive data such as passwords and personal information should be encrypted when it is persisted.

Referenced By:

Data Tier

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1381.1]

Is all sensitive data that is persisted encrypted?

Procedure:

Look at all data stores and check for encrypted passwords and other sensitive data..

Example:

G1382

Statement:

Be associated with one or more **Communities of Interest** (**COIs**).

Rationale:

The DoD Net-Centric Data Strategy emphasizes the establishment of Communities of Interest (**COI**s). This strategy introduces management of data within Communities of Interest (COIs) rather than standardizing **data elements** across the DoD. Thus all DoD Programs must map to one of more COIs. DoD Programs should participate in COIs as a normal course of doing business. They will identity relevant COIs; actively collaborate with them to promote reuse and cross-coordination of **metadata**; sponsor participation of system developers in the COI process and where appropriate contribute engineering expertise to the COI as a stakeholder. New programs should include community collaboration requirements in acquisition documents as required.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1382.1]

Is the Program associated with a **COI**?

Procedure:

Check the DoD Metadata registry to determine whether program is associated with any **COI**(s).

Example:

None

G1383

Statement:

Use a **registered namespace** in the XML Gallery in the **DoD Metadata Registry**.

Rationale:

The use of the **DoD Metadata Registry** helps to avoid name collisions and conflicts.

The assignation of a unique **registered namespace** permits a program to be uniquely identified and categorized. The DoD's Net-Centric Data Strategy requires that data products be stored in shared spaces to provide access to all authorized users and that these data products be tagged with **metadata** to enable discovery of data by authorized users. The use of a unique registered namespace provides an absolute identifier to products associated with a particular product and is an **XSD** schema requirement.

Referenced By:

Using XML Namespaces
ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1383.1]

Does the Program have an assigned namespace for its XML data assets?

Procedure:

Check **DoD Metadata Registry** to determine whether the Program is associated with **COI**(s).

Example:

None

G1384

Statement:

Review **XML Information Resources** in the **DoD Metadata Registry**, using those which can be reused.

Rationale:

The DoD Net-Centric Data Strategy requires that **XML** information resources within a **COI** in the **DoD Metadata Registry** be examined by DoD projects for possible reuse to help foster common standards within a **COI** and promote interoperability.

> ***Note:*** *The proposed DoD Metadata Registry tools have not been formally released. The Beta version thereof is in testing. Automatic Waivers of this requirement will be permitted until the tools are formally released.*

Referenced By:

Using XML Namespaces
ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1384.1]

Has the program reused information resources from the **DoD Metadata Registry**?

Procedure:

Check the **XSDs** associated with the program to determine whether XSDs referenced by other namespaces have been used. Check the **DoD Metadata Registry** to determine whether the Program has registered the reuse of XML information resources belonging to other namespaces. Reuse is indicated by formally subscribing to selected components in the registry.

Example:

None

G1385

Statement:

Identify **XML Information Resources** for registration in the XML Gallery of the **DoD Metadata Registry**.

Rationale:

The DoD Net-Centric Data Strategy requires that **XML Information Resources** developed during the course of a program be identified, examined for usefulness by other DoD Programs in the same or related **COIs** and be submitted for inclusion in the XML Gallery of the **DoD Metadata Registry**.

Referenced By:

Using XML Namespaces
ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1385.1]

Has the Program submitted new information resources to the **DoD Metadata Registry**?

Procedure:

Check the **XSDs** associated with the program namespace to determine whether they have been registered in the **DoD Metadata Registry** XML Gallery.

Example:

None

G1386

Statement:

Review predefined commonly used **data elements** in the **Data Element Gallery** of the **DoD Metadata Registry**, using those in the **relational database** technology which can be reused in the Program.

Rationale:

The DoD Net-Centric Data Strategy requires that DoD Programs examine data element information resources within a **COI** in the **DoD Metadata Registry** for possible reuse to help foster common standards within a **COI** and promote interoperability. Elements include **`US State Codes`** and **`Country Codes`**. This reuse is preferential to reusing existing industry standard **data elements** or developing new **data elements**.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1386.1]

Has the Program reused common database elements?

Procedure:

Check the DoD Metadata Registry Data Element Gallery to determine whether the program has registered database elements for reuse. Reuse is indicated by formally subscribing to selected components in the registry.

Check the program database to see whether registered have been included therein.

Example:

None

G1387

Statement:

Identify **data elements** created during Program development for registering in the **Data Element Gallery** of the **DoD MetaData Registry**.

Rationale:

The DoD Net-Centric Data Strategy requires that Programs identify and examine developed **data elements** for usefulness by other DoD Programs in the same or related **COIs** and submit the data elements for inclusion in the **Data Element Gallery** of the **DoD Metadata Registry**.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1387.1]

Has the Program submitted common database elements to the **DoD Metadata Registry**?

Procedure:

Check the DoD Metadata Registry Data Element Gallery to determine whether the program has submitted database elements for reuse.

Example:

None

G1388

Statement:

Use predefined commonly used database tables in the **DoD Metadata Registry**.

Rationale:

The DoD Net-Centric Data Strategy requires that DoD Programs examine data table information resources within a **COI** in the **DoD Metadata Registry** for possible reuse to help foster common standards within a COI and promote interoperability. This reuse is preferable to reusing existing industry standard **data elements** or developing new data elements. Some examples are **`Country Code`**, **`US State Code`**, **`Purchase Order Type Code`**, **`Security Classification Code`**. These tables are found in the **Reference Data Set** Gallery of the DoD Metadata Registry.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1388.1]

Has the Program reused common database tables?

Procedure:

Check the DoD Metadata Registry to determine whether the program has registered database tables for reuse. Reuse is indicated by formally subscribing to selected components in the registry.

Check the program database to see whether registered data tables have been included therein.

Example:

None

G1389

Statement:

Publish database tables which are of common interest by registering them in the **Reference Data Set** Gallery of the **DoD Metadata Registry**.

Rationale:

The DoD Net-Centric Data Strategy requires that DoD Programs identify and examine developed data tables for usefulness by other DoD Programs in the same or related **COIs** and be submit the data elements for inclusion in the **Reference Data Set** Gallery of the **DoD Metadata Registry**.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1389.1]

Has the Program submitted common database tables to the DoD Metadata Registry?

Procedure:

Check the DoD Metadata Registry Reference Data Set Gallery to determine whether the program has submitted database tables for reuse.

Example:

None

G1390

Statement:

Standardize on the terminology published by relevant **COIs** listed in the **Taxonomy Gallery** of the **DoD Metadata Registry**.

Rationale:

A **taxonomy** partitions the body of knowledge associated with a **COI** and defines the relationships among component parts. A taxonomy permits classification of concepts associated with a COI. This in turn provides categories and definitions for discovery tags which aids in information use and retrieval by authorized users. Program use of COI taxonomies occurs in several places:

1. Taxonomy used to describe information services for discovery.
2. Taxonomies created by the COI as a means to extend the **DDMS** for data asset discovery.
3. Taxonomies used to support mediation.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1390.1]

Has the Program adhered to the standard **taxonomies** for the **COIs** associated with the program?

Procedure:

Check the DoD Metadata Registry and Taxonomy Gallery to determine whether taxonomies exist for the COI in which the Program resides.

Example:

None

G1391

Statement:

Identify **taxonomy** additions or changes in conjunction with the **COIs** during the Program development for potential inclusion in the **Taxonomy Gallery** of the **DoD Metadata Registry**.

Rationale:

DoD Programs associated with a specific COI need to identify and submit potential taxonomy changes or additions to the **DoD Metadata Registry** to maintain an accurate and effective taxonomy within the **COI**.

Referenced By:

Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1391.1]

Has the Program submitted **taxonomy** additions or changes to the **DoD Metadata Registry**?

Procedure:

Check the DoD Metadata Registry and to determine whether the program has submitted taxonomy changes for reuse.

Example:

None

G1392

Statement:

Adhere to a common mechanism of service location.

Rationale:

Program information services are provided via a shared space for use by consumers. In order to locate these services and access the corresponding information provided, the services should be registered in the **service registry** per direction of the shared information space manager.

Referenced By:

ASD(NII) Checklist
Metadata Registry

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1392.1]

Has the Program generated default service definitions and registered them in the DoD service registry?

Procedure:

Review that there is a service definition (URLs, WSDL entries, etc.) for each of the program information services and that they have been registered accordingly.

Example:

None

G1566

Statement:

Use **`alt`** attributes to provide alternate text for non-text items such as images.

Rationale:

This usage aids users in understanding the Web page even if their browsers cannot display images.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1566.1]

Are alt attributes provided for non-text content?

Procedure:

Check for the existence of alt attributes for all Web site non-text content.

Example:

G1713

Statement:

Use an Operating Environment (OE) for all SCA applications that includes middleware that, at a minimum, provides the services and capabilities specified by Minimum CORBA Specification version 1.0.

Rationale:

Using a CORBA provider that adheres to the minimum CORBA v1.0, specification improves the interoperability between SCA Operating Environments.

Referenced By:

Software Communication Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1713.1]

Does the OE contain middleware that provides the services and capabilities of minimum CORBA?

Procedure:

Check for minimum CORBA compliance in the CORBA provider's documentation.

Example:

G1714

Statement:

Develop SCA application to only use Operating Environment functionality defined by the SCA Application Environment Profile.

Rationale:

The SCA Application Environment Profile (AEP) is a subset of the Portable Operating System Interface (POSIX) specification. Functionality that is not part of the AEP is not guaranteed to be part of the operating environment. Applications that rely on functionality that is not part of the AEP will require changes to deploy or port to other SCA platforms.

Referenced By:

Software Communication Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1714.1]

Does the SCA application use Operating Environment functions not defined by a Application Environment Profile?

Procedure:

Check to see that all Operating Environment calls in the SCA application are listed in an Application Environment Profile.

Example:

G1717

Statement:

Use constants instead of hard-coded numbers for characteristics that may change throughout the lifetime of the model.

Rationale:

Constants increase the usefulness and lifetime of a design because the model can adapt to a variety of environments by postponing or modifying those parameters late in the design cycle. This makes the code more readable, maintainable and reusable.

Note: *This practice has been adapted from Cohen, section 1.6.1.1.3.*

Referenced By:

VHDL Coding and Design

Evaluation Criteria:

1) Test: [G1717.1]

Are there any characteristics that are susceptible to modification that are directly given a value?

Procedure:

Parse the code and look for hard-coded characteristics that are susceptible to change and consider replacing them with a constant.

Example:

None

G1718

Statement:

Design circuits to be synchronous.

Rationale:

The preferred method of engineering today's digital ICs is based on a synchronous design. The main advantages of this are simplicity and reliability. Creating synchronous pieces of code increases interoperability and reusability when they are used with other synchronous modules.

Referenced By:

VHDL Synchronous Design

Evaluation Criteria:

1) Test: [G1718.1]

Are all flip-flops clocked by the same, common clock signal?

Procedure:

Check to make sure a single external clock signal triggers the design to go from a well defined and stable state to the next one. On the active edge of the clock, all input and output signals and all internal nodes are stable in either the high or low state. Between two consecutive edges of the clock, the signals and nodes are allowed to change and may take any intermediate state.

Example:

None

G1719

Statement:

Automate testbench error checking in VHDL development.

Rationale:

Manual verification is subject to human error and is time consuming. In addition, automation promotes increased maintainability, because it enables fast and reliable verification of a model when modifications are made.

> ***Note:*** *This practice has been adapted from Cohen, section 11.1.1.*

Referenced By:

VHDL Testbench

Evaluation Criteria:

1) Test: [G1719.1]

Does the testbench automatically report success or failure for each sub-test that it runs through?

Procedure:

Run the testbench to see if it automatically reports successes or failures for each sub-test.

Example:

None

G1724

Statement:

Develop XML documents to be well formed.

Rationale:

By W3C definition, XML documents must be well formed. However, documents that contain XML tags that are not well formed has no name and is often still referred to as an XML Document in common vernacular. Therefore, this guidance statements helps to clarify the need for well-formed documents. Well formed XML documents are those documents which have a proper XML syntax. This is essential if the XML is to be parsed using common, readily available open source and commercial XML parsers.

Justifies:

G1725

Referenced By:

XML Syntax

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1724.1]

Can the XML Document be parsed using a common, readily available XML Parser?

Procedure:

Open the XML document in a browser such as Mozilla Firefox or Microsoft Internet Explorer or use the XML Validator available from the W3 Schools at: http://www.w3schools.com/xml/xml_validator.asp

Example:

None

G1725

Statement:

Develop XML documents to be valid XML.

Rationale:

A valid document's content conforms to a specific set of user-defined content rules contained in XML schemas. XML schemas describe data values correctness using predefined datatypes as base types and assigning values to the datatype specific attributes of those datatypes. For example, if an element in a document is required to contain text that can be interpreted as being an integer numeric value, and instead contains: alphanumeric text such as "hello"; is empty; or has other elements in its content, then the document is considered not valid.

Derived From:

G1724

Referenced By:

Defining XML Schemas
XML Instance Documents
XML Validation

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1725.1]

Does the document validation tool indicate that the xml document is valid?

Procedure:

Use a validating parser and verify that the document is valid.

Example:

None

G1726

Statement:

Define XML Schemas using **XML Schema Definition** (XSD).

Rationale:

While it is possible to use **Document Type Definitions** (DTD) to convey much of the same information as the **XML Schema Definition** (XSD), XSDs have a several distinct advantages which are very useful in terms of interoperability. For example, DTDs do not capture domain or type range information very well (i.e. elevation in meters is from 0 to 12,000).

XML Schemas are a tremendous advancement over DTDs. Here are some of the reasons to use XSDs versus DTDs as delineated by Roger Costello in an XML tutorial (see the ***XML Schema Tutorial*** available at http://www.xfront.com):

- Enhanced datatypes support:
  - 44+ in XSDs versus 10 in DTDs
  - Support for user defined datatypes. For example, a user can define a new type based on the string type. Elements declared of this type must follow this specific pattern ddd-dddd, where d represents a numeric digit.
- Written using the same syntax as other XML instance documents. This means there is less to remember and more consistency with the same rules applying to all XML instance documents.
  XSDs support a limited Object-oriented (OO) paradigm. For example, new types can be derived from previously defined types with more or more stringent restrictions.
- Supports a kind of polymorphism where elements can be interchanged with parent or child elements. For example, a "Book" element can be substituted for the "Publication" element.
- Supports the definition of elements that are unordered collections or sets of other elements.
- Support for the identification of elements as part of a unique key.
- Support for elements that have the same name but different content
- Support for elements that have a null (i.e., nil) value.

Referenced By:

Defining XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1726.1]

Are XML schemas defined using XML Schema Definitions?

Procedure:

Verify that XML schemas are defined using W3C XML Schema Definitions rather than Document Type Definitions.

Example:

G1727

Statement:

Provide names for type definitions.

Rationale:

By naming type definitions in a schema, the type definitions can be reused in any number of other definitions. For example:

Can be reused anywhere a Point-Of-Contact needs to used. For Example:

Referenced By:

Versioning XML Schemas
Defining XML Types

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1727.2]

Do all simpleTypes have names associated with them?

Procedure:

Examine all the simpleType elements in the schema and verify that they have a name associated with them.

Example:

2) Test: [G1727.1]

Do all complexTypes have names associated with them?

Procedure:

Examine all the complexType elements in the schema and verify that they have a name associated with them.

Example:

G1728

Statement:

Define types for all elements.

Rationale:

There are two ways to associate the type-like information within an XML Schema. The first way is define an element as a global element of the schema element and the second is to define a complex or simple type. The first method violates [GXml1004] and it does not support the clean separation of the definition of types from the use of the types.

By separating the definition of the types from the definition of the elements within structures, the types can be reused and are loosely coupled from any particular instance of the domain. The definitions of the type information can be maintained by a community that wishes to share the definition rather than any particular implementation or instance.

An example using an element as the basis of a type is:

An example of correctly using a type definition might is:

Referenced By:

Defining XML Types

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1728.1]

Does the schema define any elements that are defined using references to other elements that are not part of a substitutionGroup rather than types?

Procedure:

Look for the use of an element's ref attribute.

Example:

G1729

Statement:

Annotate type definitions.

Rationale:

Types in a schema represent a particular concept or aspect within a particular subject domain. Providing documentation about the type within the schema itself helps prevent disconnects between the documentation and the implementation as captured by the type definition.

Referenced By:

Defining XML Types

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1729.1]

Do all the types defined within a schema have annotation that describes the nuances of type?

Procedure:

Look for an annotation for each simple type and complex type defined in the schema.

Example:

The complex type warranty includes an annotation that describes the purpose of the type and any caveats on when/how to use it.

G1730

Statement:

Follow an XML coding standard for defining schemas.

Rationale:

There are any number of coding standards that are defined for coding XML Schemas. Here are some areas covered by the most popular:

- Elements and Types are Upper Camel Case (UCC) convention.
- Type names end with the word Type.
- Attributes start with a lowercase letter and then revert to Lower Camel Case (LCC) convention.

Justifies:

BP1732 BP1733 BP1734

Referenced By:

Defining XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1730.1]

Is there a consistent XML coding convention followed when schemas are defined?

Procedure:

Look for the occurrence of a XML coding standard and verify that the XML Schemas follow the standard.

Example:

G1731

Statement:

Only reference Elements defined by a Type in substitution groups.

Rationale:

The 35mm, disk, and 3x5 components are simply declared as standalone elements which may be substituted for the abstract RecordingMedium element.

Note: *All of these RecordingMedium components have a type that is the same as, or derived from, the RecordingMediumType.*

Note: *The abstract RecordingMedium is associated with a type, RecordingMediumType, rather than defining the structure as part of the RecordingMedium element. This allows the definition of the RecordingMedium structure (i.e. type) to evolve independently.*

Referenced By:

Using XML Substitution Groups

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1731.1]

Do substitutionGroup references point to an abstract element that has a structures defined by a type?

Procedure:

Ensure that all substitutionGroups point to an abstract element that has a structures defined by a type.

Example:

G1735

Statement:

Use the .xsd file extension for files that contain XML Schema definitions.

Rationale:

It is possible to use any name for a schema file extension. However, using any extension other than xsd causes confusion for humans as well as tools and utilities which rely on MIMEs often mapped to file extensions.

Referenced By:

XML Schema Files

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1735.1]

Is the file extension that contains the schema definition .xsd?

Procedure:

Make sure that all XML documents that contain the xml <schema> tag have a file extension of .xsd.

Example:

None.

G1736

Statement:

Separate document schema definition and document instance into separate documents.

Rationale:

Separating the definition of the schema from the document instance supports the modularity by separating the definition of structure from the actual data. Each is allowed to evolve and change independently. In most cases, the definition of the structure of the data should be relatively static compared with the number of documents that are shared using that schema.

Document name: Camera.xsd

```
<xsd:schema
    targetNamespace="http://www.camera.org"
    elementFormDefault="qualified">
 <xsd:include schemaLocation="Nikon.xsd"/>
 <xsd:include schemaLocation="Olympus.xsd"/>
 <xsd:include schemaLocation="Pentax.xsd"/>
 <xsd:element name="Camera">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element
          name="Body"
          type="BodyType"/>
       <xsd:element
          name="Lens"
          type="LensType"/>
       <xsd:element
          name="ManualAdapter"
          type="ManualAdapterType"/>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Document name: Camera.xml

```
<?xml version="1.0"?>
<Camera xmlns ="http://www.camera.org"

        xsi:schemaLocation=
                   "http://www.camera.org
                    Camera.xsd">
 <Body>
   <Description>
      Ergonomically designed casing for easy handling
   </ Description>
 </Body>
 <Lens>
   <Zoom>300mm</Zoom>
   <F-Stop>1.2</F-Stop>
 </Lens>
 <ManualAdapter>
   <speed>1/10,000 sec to 100 sec</speed>
 </ManualAdapter>
</Camera>
```

Referenced By:

XML Instance Documents

XML Schema Files

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1736.1]

Does the instance document have a <schema> tag?

Procedure:

Check the instance document and look for the use of the schema tag or the use of the XMLSchema namespace.

Example:

None.

G1737

Statement:

Define a target namespace in schemas.

Rationale:

A target namespace describes the namespace for all the schema components defined by the schema. Without a target namespace, all enclosed schema components are not associated with a namespace and if a namespace prefix is not associated with the target namespace then all references to these schema components must be unqualified. By not specifying a target namespace, ambiguity can arise when the schema is integrated with other schemas. This can cause unnecessary naming collisions.

Note: *http://www.library.org is the target namespace as well the lib namespace. See the third targetNamespace line of the following code sample.*

```
<?xml version="1.0"?>
<xsd:schema
      targetNamespace="http://www.library.org"

      elementFormDefault="qualified">
 <xsd:include schemaLocation="BookCatalogue.xsd"/>
 <xsd:element name="Library">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="BookCatalogue">
         <xsd:complexType>
           <xsd:sequence>
             <xsd:element ref="lib:Book"
                           maxOccurs="unbounded"/>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Referenced By:

Using XML Namespaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1737.1]

Does the schema declare a target namespace?

Procedure:

Check the definition of all schemas and look for the assignment of the targetNamespace attribute.

Example:

```
<xsd:schema

  targetNamespace="http://www.library.org"
 >
  . . .
</xsd:schema>
```

G1738

Statement:

Define a qualified namespace for the target namespace.

Rationale:

To force all schema components defined by the schema to be qualified and to belong to a namespace, associate a qualified namespace with the target namespace. This causes all components defined within the namespace to be explicitly associated with a namespace. In other words, all components are always qualified.

Note: *http://www.library.org is the target namespace as well the lib namespace. See the forth xmlns:lib line of the following code sample.*

```
<?xml version="1.0"?>
<xsd:schema
      targetNamespace="http://www.library.org"

      elementFormDefault="qualified">
 <xsd:include schemaLocation="BookCatalogue.xsd"/>
 <xsd:element name="Library">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="BookCatalogue">
         <xsd:complexType>
           <xsd:sequence>
             <xsd:element ref="lib:Book"
                          maxOccurs="unbounded"/>
           </xsd:sequence>
         </xsd:complexType>
       </xsd:element>
     </xsd:sequence>
   </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Referenced By:

Using XML Namespaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1738.1]

Does the schema declare a qualified namespace for the target namespace?

Procedure:

Check the definition of all schemas and look for the assignment of the targetNamespace attribute and make sure there is also a qualified namespace with the same name.

Example:

In this example, the targetNamespace and the qualified namespace lib both have the same URI associated with them.

```
<xsd:schema

  targetNamespace="http://www.library.org"
 >
  . . .
</xsd:schema>
```

G1740

Statement:

Append the suffix Type to type names.

Rationale:

Syntactically, XML allows names within a namespace to be reused as long as they do not define the same XML Schema component. Therefore, a type and an element can both have the same name. A parser can easily differentiate the components, but a human can not. In order to maintain maintainable "user-friendly" code, differentiate types and elements by adding a type suffix for types.

Referenced By:

Defining XML Types

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1740.1]

Do all the complex type names end in the type suffix?

Procedure:

Examine all the complex and simple type schema component definitions and verify that they end in the suffix type.

Example:

G1744

Statement:

Only reference abstract Elements in substitution groups.

Rationale:

An abstract element can not have its type instantiated in an instance document. This means that the element used as the basis for the substitution group and all the members of the substitution group must be derived from the same type.

Referenced By:

Using XML Substitution Groups

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1744.1]

Is the element used as the basis for the substitution group declared to be abstract and is it derived from a type?

Procedure:

Examine all the elements used as the basis for substitution groups and verify that they have been declared as abstract.

Example:

```
<xsd:element name="RecordingMedium"
      abstract="true"
      type="RecordingMediumType"/>
```

G1745

Statement:

Append the suffix Group to substitution group element names.

Rationale:

Syntactically, XML allows names within a namespace to be reused as long as they do not define the same XML Schema component. Therefore, a type and an element can both have the same name. A parser can easily differentiate the components, but a human can not. In order to maintain maintainable "user-friendly" code, differentiate types and elements by adding a type suffix for types.

Referenced By:

Using XML Substitution Groups

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1745.1]

Do all the complex type names end in the type suffix?

Procedure:

Examine all the complex and simple type schema component definitions and verify that they end in the suffix type.

Example:

G1746

Statement:

Develop XSLT stylesheets that are XSLT version agnostic.

Rationale:

There are never any guarantees as to the XSLT environment that a stylesheet will be used in. There are ways of writing code as recommended by the W3C so that the stylesheets operate in XSL Version 1.0, 2.0 and future releases. See W3C Extensibility and Fallback for XSL Transformations (XSLT) 2.0 for details.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1746.2]

Does the stylesheet support 2.0 and future version portability as defined by the W3C Extensibility and Fallback for XSL Transformations (XSLT) 2.0?

Procedure:

Look for the use of the use-when attribute in the xsl:value element.

Example:

```
<xsl:value-of
   select="pad($input, 10)"
   use-when="function-available('pad', 2)"
/>
<xsl:value-of
  select
    ="concat
       ( $input,
         string-join
          ( for $i in
              1 to
              10 - string-length($input)
              return ' ',
            ''
          )
       )"
  use-when="not(function-available('pad', 2)
"/>
```

2) Test: [G1746.1]

Does the stylesheet support version 1.0 and 2.0 portability as defined by the W3C Extensibility and Fallback for XSL Transformations (XSLT) 2.0?

Procedure:

Look for the use of the xsl:when and xsl:otherwise construct where the 2.0 functions are tested for availability in the xsl:when branch and the 1.0 functionality is defined in the xsl:otherwise branch. For a comprehensive list of 2.0 functions see the W3Schools site on XPath, XQuery and XSLT Functions.

Example:

```
<out xsl:version="2.0">
 <xsl:choose>
   <xsl:when
      test="function-available('matches')">
     <xsl:value-of
        select="matches($input, '[a-z]*')"/>
   </xsl:when>
   <xsl:otherwise>
     <xsl:value-of
        select=
          = "string-length
              ( translate
               ( $in,
                 'abcdefghijklmnopqrstuvwxyz',
                   ''
               )
              )
             = 0"
      />
   </xsl:otherwise>
 </xsl:choose>
</out>
```

G1751

Statement:

Document all XSLT code.

Rationale:

XSLT is source code and should be internally documented including a file header that describes the purpose of the transform and any restrictions or caveats associated with the transform.

Derived From:

G1027

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1751.1]

Doe the XSLT have internal comments that document the transform?

Procedure:

Look inside the XSLT code and look for internal comments.

Example:

```
<xsl:for-each
  select="/transactions/transaction">
 <!--
    NOTE: Since dates are currently in
    ISO format they are in a sorted format
    and need no multi-level sorting
  -->
 <xsl:sort
    order="ascending"
    select="@startdate"/>
 <tr>
   <td>
     <xsl:value-of
        select="@startdate"/>
   </td>
   <td>
     <xsl:value-of
        select="@description"/>
   </td>
   <td>
     <!—  Get year
           1234567890
```

```
            yyyy/mm/dd
       -->
      <xsl:value-of
         select="substring(@startdate, 1,4)"
       />
    </td>
    <td>
      <!—  Get month
            1234567890
            yyyy/mm/dd
       -->
      <xsl:value-of
         select="substring(@startdate, 6,2)"/>
    </td>
    <td>
      <!—  Get day
            1234567890
            yyyy/mm/dd
       -->
      <xsl:value-of
         select="substring(@startdate, 9,2)"/>
    </td>
  </tr>
</xsl:for-each>
```

G1753

Statement:

Declare the XML schema version with an attribute in the root element of the schema definition.

Rationale:

Formalizing the schema version number through the use of a required XML attribute helps automate the process of validating the versions. This will reduce unexpected runtime errors that occur when assumptions are made about the schema that may change over time. http://www.xfront.com/SchemaVersioning.html

Derived From:

G1018

Referenced By:

Versioning XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1753.1]

Does the schema definition define a required attribute that captures the version information?

Procedure:

Look at the schema definition file and look for the inclusion of a required attribute that captures the schema version number. In the following example, the schemaVersion attribute is defined.

Example:

```
<xs:schema

  targetNamespace="http://www.exampleSchema"
  xmlns: xs ="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  version="1.3"
>
 <xs:element name="Example">
   <xs:complexType>
      . . .
     <xs:attribute
        name="schemaVersion"
        type="xs:decimal"
        use="required"
      />
   </xs:complexType>
 </xs:element>
```

G1754

Statement:

Give each new XML schema version a unique URL.

Rationale:

This allows the previous versions of the schema to be made available to support uninterrupted processing and supports an orderly transition. It also allows the users of the schemas to compare and contrast the evolving schema. http://www.xfront.com/SchemaVersioning.html

Referenced By:

Versioning XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1754.1]

Look for the multiple schemas that represent different versions with different URLs.

Procedure:

Look for XSDs that all define a particular schema but can be found at different locations. This can be done by changing the path to the schema definition or that change the name of the file by adding the version number.

Example:

Changing the file path:

```
http://www.some.org/schema/1999/CoiSchema
http://www.some.org/schema/2003/CoiSchema
http://www.some.org/schema/2006/CoiSchema
```

Changing the file name:

```
http://www.some.org/schema/CoiSchema_1999
http://www.some.org/schema/CoiSchema_2003
http://www.some.org/schema/CoiSchema_2006
```

G1755

Statement:

Use accepted file extensions for all files that contain XSL code.

Rationale:

It is possible to use any name for an XSL file extension. However, using any extension other than xsl or XSLT causes confusion for humans as well as tools and utilities which rely on MIMEs often mapped to file extensions.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1755.1]

Is the file extension that contains the XSL files .xsl or .xslt?

Procedure:

Make sure that all XSL files have a file extension of .xsl or xslt.

Example:

None.

G1756

Statement:

Isolate XPath expression statements into the configuration data.

Rationale:

XPath expression statements are dependent on the XML Schemas that are associated with the documents. Consequently they need maintained independently from the applications that use them. Storing the XPath expression statements externally as part of the configuration data ensures a clean separation of the maintenance tasks and supports traceability using configuration management tools.

Referenced By:

XPath

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1756.1]

Are there XPath expression statements embedded as string literals in the application source code?

Procedure:

Look for the occurrence of XPath expression statements or XML Element names defined as strings within the source code.

Example:

```
void main ( String args)
{ . . .
  String titleSearchExpression
    = "/library/books/book/title";
  . . .
} // End main
```

G1759

Statement:

Use a style guide when developing Web portlets.

Rationale:

Portals contain portlets from different sources, and it is important for usability for the portal to have a common look and feel across all portlets.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1759.1]

Do all portlets comply with a style guide.

Procedure:

Look at development documentation to determine if a style guide exist for web portlets and look for code reviews that show it was used during development.

Example:

- Ahlstrom, V. & Allendoerfer, K. Web-Based Portal Computer-Human Interface Guidelines, 2004. Retrieved from: http://hf.tc.faa.gov/products/bibliographic/tn0423.htm (July 2006).
- Web Portal Design Guide , Fernandes, K., Space and Nabal Warfare Systems Center San Diego 2006

G1760

Statement:

Solicit feedback from users on user interface usability problems.

Rationale:

Active testing and solicitation of input from users helps identify usability problems with the user interface and helps to identify areas that may reduce performance or require excessive cognitive attention by the user.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1760.1]

Does the program solicit user feedback for user interface usability problems?

Procedure:

Determine if user surveys are conducted on the usability of the system.

Example:

G1761

Statement:

Provide units of measurements when displaying data.

Rationale:

Displayed units for measurable data provide for better understanding the data and enable reuse of the data. (This guidance is derived from MIL-STD 1472F)

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1761.1]

Does the system display units for all measurable data?

Procedure:

Inspect the user interfaces for system and check that units are shown for all measurable data.

Example:

Length displayed as meters
Distance displayed as miles.

G1762

Statement:

Indicate all simulated data as simulated.

Rationale:

Simulated data that is not marked as simulated may be of misinterpreted and can decrease system, user, or system safety. (This guidance is derived from MIL-STD 1472F)

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1762.1]

Is all simulated data clearly marked as simulated?

Procedure:

Check system inputs and outputs including user interfaces and check that the simulated data is properly labeled as simulated.

Example:

G1763

Statement:

Indicate the security classification for all classified data.

Rationale:

Displaying classified data without clearing marking the classification can lead to incorrect assumptions about the data. This can lead to improperly use of the data or prevent the data from being reused due to lack of clear understanding of the classification. (This guidance is derived from MIL-STD 1472F)

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1763.1]

Does the system display classification markings for all classified data?

Procedure:

Check the system outputs and user interfaces for classification marking for all classified data or systems.

Example:

Classification banners on monitors
Classification banners on printouts

G1770

Statement:

Explicitly define the **Data Distribution Service** (DDS) **Domains** for the system.

Rationale:

DDS uses Domains to separate the **Global Data Spaces** into independent areas. **Topics** written to one DDS Domain are completely hidden from the other DDS Domains. Use DDS Domains for isolation (hiding subsystem data from other parts of the system), modularity, and scalability. In order for systems to benefit from these advantages, they must explicitly define their own DDS Domains rather than use the default DDS Domain.

Referenced By:

DDS Domains - Global Data Spaces

Acquisition Phase:

Design

Evaluation Criteria:

1) Test: [G1770.1]

Is the system using different **`DomainId`** values to isolate the subsystems?

Procedure:

Look for multiple calls to **`create_participant()`** operation on the **`DomainParticipantFactory`**.

Example:

```
participantFactory
  = TheParticipantFactory;
quickQuoterParticipant
  = participantFactory->create_participant
      ( QUICK_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
realtimeQuoterParticipant
  = participantFactory->create_participant
      ( REALTIME_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111

G1771

Statement:

Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of a **publisher**.

Rationale:

DDS relies on the use of QoS characteristics to match publishers with **subscribers**. If the publishers do not specify a QoS policy other than the default, much of the power of DDS publishing is lost and the capabilities of the publisher are not documented.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1771.1]

Are values other than the **`PUBLISHER_QOS_DEFAULT`** value used to create publishers?

Procedure:

Verify that the **`PUBLISHER_QOS_DEFAULT`** constant is not used within the code.

Example:

```
DDS::Publisher publisher
  = participant->create_publisher
      ( PUBLISHER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111

2) Test: [G1771.2]

Is the **`get_default_publisher_qos`** operation used to create publisher?

Procedure:

Look for the use of the **`get_default_publisher_qos`** operation within the code.

Example:

```
participant
  = participantFactory->create_participant
      ( QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
```

```
        DDS::STATUS_MASK_ALL
      );
DDS::PublisherQos publisherQos;
Participant->get_default_publisher_qos
  ( publisherQos );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111

G1772

Statement:

Assign a unique identifier for each **Data-Distribution Service** (DDS) **Domain** within the system.

Rationale:

DDS uses Domains to separate the **Global Data Spaces** into independent areas. Within DDS, a unique identifier called the **`DomainId`** identifies each DDS Domain.

Referenced By:

DDS Domains - Global Data Spaces

Acquisition Phase:

Design

Evaluation Criteria:

1) Test: [G1772.1]

Is there a single value for the **`DomainId`** used for each Domain when the **`create_participant`** operation is used?

Procedure:

Look for the use of the **`create_participant`** operation within the code.

Example:

```
participantFactory
  = TheParticipantFactory;
quickQuoterParticipant
  = participantFactory->create_participant
      ( QUICK_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
realtimeQuoterParticipant
  = participantFactory->create_participant
      ( REALTIME_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111

G1773

Statement:

Use **`#include`** guards for all headers.

Rationale:

Including a guard prevents including a header file more than once. There are two basic kinds of guards: internal and external. Internal guards occur in each header file that is to be included. External guards occur in a file that includes a header file. In the past, there were compiling performance issues using internal guards because the file had to be scanned each time the file was included. This has been optimized away by most modern compilers. Furthermore, external guards are fragile and tightly coupled since the file including the header and header file must use the same guard name.

> ***Note:*** *This practice has been adapted from Sutter and Alexandrescu, standard practice 24.*

Referenced By:

C++ Header Files

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1773.1]

Do all header files contain include guards?

Procedure:

Check each file that is included using a **`#include`** statement to make sure it has an include guard.

Example:

An internal guard looks like this:

G1774

Statement:

Make header files self-sufficient.

Rationale:

To enable code reuse, each unit of code should be able to be compiled independently without having to follow a predetermined build order or having to know the dependencies. Code is difficult to reuse when the dependencies are not clearly documented. Therefore, ensure each header is capable of being used by itself (i.e, it can be compiled standalone) by having it include all the headers upon which it depends.

> ***Note:*** *This practice has been adapted from Sutter and Alexandrescu, standard practice 23.*

Referenced By:

C++ Header Files

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1774.1]

Can each class be compiled by itself without having to compile other units?

Procedure:

Compile each class as a standalone file and check compile output for errors caused by missing definitions.

Example:

None

G1775

Statement:

Do not overload the logical **AND** operator.

Rationale:

The logical **AND** operator has a special relationship with the compiler. When a logical **AND** operator is written to overload the inherent operators, the precedence of operation (i.e., left side of operator or right side of operator) is undefined. This can result in compiler dependency. In the following code, it is not clear whether the **DisplayPrompt** will execute first or the **GetLine** operation will executed first.

```
if ( DisplyPrompt() && GetLine() )
```

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 30.*

Referenced By:

C++ Operator Overloading

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1775.1]

Is the logical **AND** operator defined?

Procedure:

Look for the overloading of the logical **AND** operator.

Example:

None

G1776

Statement:

Do not overload the logical **OR** operator.

Rationale:

The logical **OR** operator has a special relationship with the compiler. When a logical **OR** operator is written to overload the inherent operators, the precedence of operation (i.e., left side of operator or right side of operator) is undefined. This can result in compiler dependency.

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 30.*

Referenced By:

C++ Operator Overloading

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1776.1]

Is the logical **OR** operator defined?

Procedure:

Look for the overloading of the logical **OR** operator.

Example:

None

G1777

Statement:

Do not overload the **`comma`** operator.

Rationale:

The **`comma`** operator has a special relationship with the compiler. When a **`comma`** operator is written to overload the inherent operators, the precedence of operation (i.e., left side of operator or right side of operator) is undefined. This can result in compiler dependency.

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 30.*

Referenced By:

C++ Operator Overloading

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1777.1]

Is the **`comma`** operator defined?

Procedure:

Look for the overloading of the **`comma`** operator.

Example:

None

G1778

Statement:

Place all **`#include`** statements before all namespace **`using`** statements.

Rationale:

Files that are included can contain their own **`using`** clauses. In order to make sure that the **`using`** statements are not overridden by these subsequent using definitions, place all using statements after all include statements.

> ***Note:*** *This practice has been adapted from Sutter and Alexandrescu, standard practice 59.*

Referenced By:

C++ Namespaces and Modules

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1778.1]

Are all the **`using`** statements defined after all the **`#include`** statements?

Procedure:

Scan all files and make sure that all the **`using`** statements occur after all **`using`** statements.

Example:

None

G1779

Statement:

Explicitly namespace-qualify all names in header files.

Rationale:

Header files are meant to be included by other files. A header file inclusion should not alter the meaning of code that it is included in as this behavior is unexpected. Therefore, use fully-qualified names in header files and do not use using directives or declarations. This also promotes clarity in the header file whose main purpose is to communicate the interface to the implementation class.

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 59.*

Referenced By:

C++ Header Files
C++ Namespaces and Modules

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1779.1]

Are named fully namespace qualified throughout the header files?

Procedure:

Scan all header files and make sure that all namespaces are fully qualified.

Example:

None

2) Test: [G1779.2]

Are all header files free from using directives or declarations?

Procedure:

Scan all header files to determine that they do not contain using directives or declarations.

Example:

None

G1796

Statement:

Explicitly define all the **Data Distribution Service** (DDS) **Domain Topics**.

Rationale:

DDS uses Topics to define the information model. Topics are identified by an application-defined string and an associated **data type**. Topics represent collections of object sin the **Global Data Space**; individual data-objects within a Topic are identified by the value of the key fields which are some special fields inside the data-type. Applications use Topics to publish the information and subscribe to the information they want.

In a DDS system information exchange happens as a result of **publishers** and **subscribers** agreeing to use the same Topics. Therefore the selection of the Topic names and their semantic meaning is an important part of system design.

Referenced By:

Messaging within a DDS Domain

Acquisition Phase:

Design and Development

Evaluation Criteria:

1) Test: [G1796.1]

Are all the Topics (and Topic names) the system uses explicitly defined and captured in a centralized document (e.g., Excel table, XML file, dedicated tool)?

Procedure:

Look for documentation that contains listings for all Topics the system uses.

Example:

```
<topic>
 <name>Temperature</name>
 <type>TemperatureData</type>
 <description>
   This topic contains a reading of
   a temperature sensor
 </description>
</topic>
<topic>
  . . .
</topic>
```

G1798

Statement:

Explicitly define all the **Data Distribution Service** (DDS) **Domain data types**.

Rationale:

DDS provides support for writing and reading typed data. For each application data type, DDS creates the necessary objects that allow manipulation of the data object. For example, for a given data type named **`MyDT`**, DDS creates a **`MyDTDataWriter`** and **`MyDTDataReader`**.

Knowing the data type of the object allows DDS to marshal the data properly. Consequently, any computer platform and/or language can process the data properly . For example, DDS performs the proper endianess transformations, alignment, and adjustment for 32 versus 64 bit platforms.
Knowing the data type is also required for the proper functioning of **`ContentFilteredTopics`**.

Moreover, explicit definition of the data types is required for the tools provided by DDS vendors to display and manipulate the data properly. Visualization tools, logging and replay, automatic bridging to other middleware, etc., all depend on data type transparency.

Referenced By:

Messaging within a DDS Domain

Acquisition Phase:

Design and Development

Evaluation Criteria:

1) Test: [G1798.1]

Are all the data types the system uses explicitly defined using IDL which is either manually written or generated from equivalent UML or XML representations?

Procedure:

Look for the IDL (or equivalent XML) files used to define the types used by the system.

Example:

```
// File MyTpes.idl
struct MyType
{
   long x;
   long y;
   string<10> units;
};
```

G1799

Statement:

Explicitly associate data types to the **Data Distribution Service** (DDS) **Topics** within a DDS **Domain**

Rationale:

A DDS Topic represents a homogeneous collection of data-objects in the **Global Data Space**. All data-objects within a Topic share a common **data-type**. Knowledge of the type associated with the Topic is required for an application to be able to publish and subscribe data on the Topic.

Referenced By:

Messaging within a DDS Domain

Acquisition Phase:

Design and Development

Evaluation Criteria:

1) Test: [G1799.1]

Do all Topics have an explicit association to a data type.

Procedure:

Look for documentation that lists the Topics in use by the system and verify that each Topic has a data type associated with it

Example:

```
<topic>
 <name>Temperature</name>
 <type>TemperatureData</type>
 <description>
   This topic contains a reading of
   a temperature sensor
 </description>
</topic>
<topic>
  . . .
</topic>
```

G1800

Statement:

Explicitly identify Keys within the **Data Distribution Service** (DDS) **data type** that uniquely identify an instance of a data object.

Rationale:

Within each DDS **Domain** (i.e., **Global Data Space**) a data-object is identified by the tuple (**Topic**, Key). The Key is a set of fields within the data type associated with the Topic that the application has tagged to indicate their role in uniquely identifying the data object. For example, if the Topic represents a person to the IRS, the Key may be simply the field containing the social security number.

The proper definition of the key is necessary to allow DDS to implement the **`KEEP_LAST HISTORY`** QoS properly as well as to enforce QoS policies such as **`DEADLINE`**, and **`OWNERSHIP`**. It is also necessary in order for DDS to supply the proper Sample information to the **DataReader**.

All data types require Keys except in the case where the Topic logically represents a single object, for example when the Topic represents a Message Queue.

Referenced By:

Messaging within a DDS Domain

Acquisition Phase:

Design and Development

Evaluation Criteria:

1) Test: [G1800.1]

Does the declaration of the data-type associated with the Topic explicitly designate using one or more of the fields as a Key?

Procedure:

Examine the IDL (or equivalent XML) files used to define the types used by the system to identify the declaration of the data-type associated with each Topic (i.e., see if there are any tags that designate which fields form the Key).

Example:

```
For data types defined using IDL:
struct SensorData
{
  long    sensor_id; //@key
  float   value;
  string<32> units;
  string<64> location;
};
struct DepartingFlightData
{
  string<8>    airline_code;  //@key
  long         flight_number; //@key
  string<8>    destination_airport_code;
  string<2>    departing_terminal;
  long         departing_gate;
```

```
  FlightTime   scheduled_departure_time;
  FlightTime   expected_departure_time;
  string<32>   status;
};
```

G1801

Statement:

Explicitly define a **Topic Quality of Service** (QoS) for each **Data Distribution Service** (DDS) Topic within a DDS **Domain**.

Rationale:

DDS Topics define the information model of the system. The QoS Policies associated with the Topics define expectations and constraints that all users (**publishers** or **subscribers**) of the Topic should know. Consequently, definition of the Topic QoS is an important part of the system design.

Referenced By:

DDS Quality of Service
Messaging within a DDS Domain

Acquisition Phase:

Design

Evaluation Criteria:

1) Test: [G1801.1]

Is there a document that defines the QoS Policies that each Topic uses and does the document that describes the Topics and their associated data types also provide information on the Topic QoS?

Procedure:

Look at the documents that define the Topics in use and their associated data-types and see if they also define the Topic QoS.

Example:

```
Topic: DepartingAircraft
Type: DepartingAircraftStruct
QoS: HISTORY kind=KEEP_LAST
QoS: RELIABILITY kind=RELIABLE
QoS: DEADLINE duration=15minutes
QoS: LIFESPAN duration = 1 hour
Etc.
```

G1802

Statement:

Catch all **Data Distribution Service** (DDS) events.

Rationale:

DDS uses **listeners** to notify the application of relevant events such as mis-matched Topic definitions, **QoS** violations, lost samples, etc. Normally these events are dispatched to the most specific entity to which they apply (e.g., the affected **DataReader** in the case of the lost sample notification). However under application control the **`DataReader`** can "mask" certain events such that they are propagated to the enclosing container entity (e.g. the **Subscriber** to which the affected **`DataReader`** belongs). The **DomainParticipant** is the ultimate container of all DDS entities and it is therefore important that it handles (e.g., logs) any events that the contained entities have not handled.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1802.1]

Is a non-nil listener specified when the **`DomainParticipant`** is created?

Procedure:

Look at the arguments passed to the **`create_domain_participant`** operation on the **`DomainParticipantFactory`** and check the values of the listener and mask arguments.

Example:

```
participantFactory
  = TheParticipantFactory;
participant
  = participantFactory->create_participant
      ( QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1803

Statement:

Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe real-time messaging criteria for **Publishers**.

Rationale:

DDS relies on the use of a QoS set of characteristics to match publishers with **subscribers**. If the publishers do not specify a QoS policy other than the default, much of the power of DDS publishing is lost and the capabilities of the publisher are not documented.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1803.2]

Is the **`get_default_publisher_qos`** operation used to create publisher?

Procedure:

Look for the use of the **`get_default_publisher_qos`** operation within the code.

Example:

```
participant
  = participantFactory->create_participant
      ( QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
DDS::PublisherQos publisherQos;
Participant->get_default_publisher_qos
  ( publisherQos );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

2) Test: [G1803.1]

Is the **`PUBLISHER_QOS_DEFAULT`** value used to create publishers?

Procedure:

Look for the use of the **`PUBLISHER_QOS_DEFAULT`** constant within the code.

Example:

```
DDS::Publisher publisher
```

```
  = participant->create_publisher
      ( PUBLISHER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1804

Statement:

Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe **DataWriter**.

Rationale:

DDS relies on the use of QoS characteristics to match a **`DataWriter`** with each **DataReader** of the same **Topic**. If the **`DataWriter`** does not specify a QoS policy other than the default, much of the power of DDS publishing is lost and the capabilities of the **`DataWriter`** are not documented.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1804.2]

Is the **`get_default_datawriter_qos`** operation used to create participant?

Procedure:

Look for the use of the **`get_default_datawriter_qos`** operation within the code.

Example:

```
DDS::DataWriterQos dataWriterQos;
publisher->get_default_datawriter_qos
  ( dataWriterQos );
DDS::DataWriter dataWriter
  = publisher ->create_datawriter
      ( myTopic,
        dataWriterQos,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

2) Test: [G1804.1]

Is the **`DATAWRITER_QOS_DEFAULT`** value used to create **`DataWriter?`**

Procedure:

Look for the use of the **`DATAWRITER_QOS_DEFAULT`** constant within the code.

Example:

```
DDS::DataWriter dataWriter
  = participant->create_datawriter
```

```
      ( myTopic,
        DATAWRITER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1805

Statement:

Explicitly define the **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of the **Subscriber**.

Rationale:

DDS relies on the use of QoS set of characteristics to match subscribers with **publishers**. If the subscribers do not specify a QoS policy other than the default, much of the power of DDS subscription and publishing is lost and the requirements of the subscriber are not documented.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1805.2]

Is the **`get_default_subscriber_qos`** operation used to create subscribers?

Procedure:

Look for the use of the **`get_default_subscriber_qos`** operation within the code.

Example:

```
participant
  = participantFactory->create_participant
      ( QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
DDS::SubscriberQos subscriberQos;
Participant->get_default_subscriber_qos
  ( subscriberQos );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

2) Test: [G1805.1]

Is the **`SUBSCRIBER_QOS_DEFAULT`** value used to create subscribers?

Procedure:

Look for the use of the **`SUBSCRIBER_QOS_DEFAULT`** constant within the code.

Example:

```
DDS::Publisher publisher
```

```
  = participant->create_subscriber
      ( SUBSCRIBER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1806

Statement:

Explicitly define the Request-Offered **Data Distribution Service** (DDS) **Quality of Service** (QoS) Policies to describe the behavior of the **DataReader**.

Rationale:

DDS relies on the use of QoS characteristics to match a **DataWriter** with each **`DataReader`** of the same Topic. If the **`DataReader`** does not specify a QoS policy other than the default, much of the power of DDS subscription and publishing is lost and the requirements of the **`DataReader`** are not documented.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1806.1]

Is the **`DATAREADER_QOS_DEFAULT`** value used to create **`DataReader`**?

Procedure:

Look for the use of the **`DATAREADER_QOS_DEFAULT`** constant within the code.

Example:

```
DDS::DataResder dataReader
  = participant->create_datareader
      ( DATAREADER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

2) Test: [G1806.2]

Is the **`get_default_datareader_qos`** operation used to create participant?

Procedure:

Look for the use of the **`get_default_datareader_qos`** operation within the code.

Example:

```
DDS::DataReaderQos dataReaderQos;
publisher->get_default_datareader_qos
  ( dataReaderQos );
DDS::DataReader dataReader
  = publisher ->create_datareader
```

```
      ( myTopic,
        dataReaderQos,
        NULL,
        DDS::STATUS_MASK_ALL
      );
```

G1807

Statement:

Check the return values of **Data Distribution Service** (DDS) functions.

Rationale:

Many of the DDS operations return a nil value when the operation does not work. Not checking for these **`nil`** values can cause unexpected and potentially non-deterministic behavior. Different implementations of the DDS may even behave differently when these values are used. The following is a list of operations that can return **`nil`**:

- create_publisher
- create_subscriber
- create_topic
- create_contentFilteredtopic
- create_multitoic
- find_topic
- lookup_topicdescription
- create_participant
- lookup_participant
- create_datawriter
- lookup_datawriter
- create_datareader
- lookup_datareader
- create_readcondition
- create_querycondition

One operation returns **`HANDLE_NIL`** when the operation fails.

- **`lookup_instance`**

The remaining operations return a **`DDS::ReturnCode_t`** enumerated value that indicates whether the operation succeeded (**`DDS::RETCODE_OK`**) of else the reason for failure.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1807.2]

Do all invocations of the DDS operations lookup_instance check for a return value of **`HANDLE_NIL`**?

Procedure:

Examine the code for the use of the lookup_instance operations and make sure they check for the return of a **`DDS::HANDLE_NIL`** value immediately after the operation.

Example:

```
DDS::InstanceHandle_t instanceHandle
  = DDS::HANDLE_NIL;
instanceHandle
  = writer->lookup_instance( instance )
if ( instanceHandle == DDS::HANDLE_NIL )
  { cerr << "..."
        << endl;
    exit(1);
  } // End if
```

2) Test: [G1807.3]

Are all invocations to DDS operations that return a **`DDS::ReturnCode_t`** checked for **`DDS::RETCODE_OK`**?

Procedure:

Examine the code for the use of the operations with prototype returning **`DDS::ReturnCode_t`** to make sure they check for the return of a **`DDS::RETCODE_OK`** immediately after the operation.

Example:

```
retcode
  = writer->write( … )
if ( retcode != DDS::RETCODE_OK )
  { cerr << "..."
        << endl;
    //  handle error
} // End if
```

3) Test: [G1807.1]

Are all of the DDS operations that can return **`nil`** values checked for the return of a **`nil`** values?

Procedure:

Examine the code for the use of the following operations and make sure they check for the return of a **`nil`** value immediately after the operation.

- create_publisher
- create_subscriber
- create_topic
- create_contentFilteredtopic

- create_multitoic
- find_topic
- lookup_topicdescription
- create_participant
- lookup_participant
- create_datawriter
- lookup_datawriter
- create_datareader
- lookup_datareader
- create_readcondition
- create_querycondition

Note: *Examine the return of any other operation and make sure they check for* ***`DDS:RETCODE_OK`*** *immediately after the operation.*

Example:

```
DDS::Publisher publisher
  = participant->create_publisher
      ( PUBLISHER_QOS_DEFAULT,
        NULL,
        DDS::STATUS_MASK_ALL
      );
if ( publisher == NULL ) )
  { cerr << "create_publisher failed."
        << endl;
    exit(1);
  } // End if
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1808

Statement:

Handle all **Data Distribution Service** (DDS) **Quality of Service** (QoS) contract violations using one of the **Subscriber access APIs**.

Rationale:

QoS contract violations typically indicate either a system mis-configuration, or else a transient failure (e.g., a network that has been temporarily disconnected). Either way the application must monitor these events to determine if they are relevant to their operation and consequently take proper corrective action.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1808.1]

Are all the DDS QoS-related status change events are captured via a DDS **Listener** or a DDS **WaitSet**?

Procedure:

Specifically ensure that the following DDS events are handled. Look at the arguments passed to the **`create_domain_participant`**, **`create_datawriter`**, and **`create_datareader_operations`** and check that the listener and mask parameters to verify that the following events are being handled:

- OFFERED_DEADLINE_MISSED_STATUS
- REQUESTED_DEADLINE_MISSED_STATUS
- OFFERED_INCOMPATIBLE_QOS_STATUS
- REQUESTED_INCOMPATIBLE_QOS_STATUS
- LIVELINESS_LOST_STATUS
- LIVELINESS_CHANGED_STATUS

Example:

```
participantFactory
  = TheParticipantFactory;
quickQuoterParticipant
  = participantFactory->create_participant
      ( QUICK_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        participantListener,
        DDS::STATUS_MASK_ALL
```

```
    );
```

`DDS::STATUS_MASK_ALL` is part of DDS 1.3, prior releases require application to use 0x11111111.

G1809

Statement:

Handle all **Data Distribution Service** (DDS) events using one of the **subscriber access APIs**.

Rationale:

Listeners and the dual **Condition**/**WaitSet** infrastructure allow applications to be notified when changes occur in a **DCPS** communication.

Listeners provide a generic mechanism for the middleware to notify the application of relevant asynchronous events, such as arrival of data corresponding to a **subscription**, violation of a **QoS** setting, etc. Each DCPS entity supports its own specialized kind of listener. Listeners are related to changes in status conditions. Listener operations are invoked using a middleware-provided thread.

Conditions and **`WaitSets`** provide the means for an application thread to block waiting for the same events that can be received via a Listener. Using a **`WaitSet`**, the application can handle the event in its own thread instead of the middleware provided thread used for Listeners.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1809.1]

Are all DDS status change events are captured via a DDS Listener or a DDS WaitSet?

Procedure:

Verify that the following DDS events are handled. Look at the arguments passed to the **`create_domain_participant`**, **`create_datawriter`**, and **`create_datareader_operations`** checking that the listener and mask parameters to verify that the following events are handled:

- **`INCONSISTENT_TOPIC_STATUS`**
- **`SAMPLE_LOST_STATUS`**
- **`SAMPLE_REJECTED_STATUS`**
- **`DATA_ON_READERS_STATUS`**
- **`DATA_AVAILABLE_STATUS`**
- **`OFFERED_DEADLINE_MISSED_STATUS`**
- **`REQUESTED_DEADLINE_MISSED_STATUS`**
- **`OFFERED_INCOMPATIBLE_QOS_STATUS`**

- **REQUESTED_INCOMPATIBLE_QOS_STATUS**
- **LIVELINESS_LOST_STATUS**
- **LIVELINESS_CHANGED_STATUS**

Example:

```
participantFactory
  = TheParticipantFactory;
quickQuoterParticipant
  = participantFactory->create_participant
      ( QUICK_QUOTER_DOMAIN_ID,
        PARTICIPANT_QOS_DEFAULT,
        participantListener,
        DDS::STATUS_MASK_ALL
      );
```

DDS::STATUS_MASK_ALL is part of DDS 1.3, prior releases require application to use 0x11111111.

G1810

Statement:

Use **data models** to document the data contained within the **Data Distribution Service** (DDS) **Data-Centric Publish Subscribe** (DCPS).

Rationale:

DCPS contains static and raw data that can be used is any number of views or objects. As a consequence, changes in the definition of the data, its DDS **Domains** or its structure can have a huge cascading effect. To minimize the impact of these changes, data needs to be documented in a data model that is not subject to implementation.

Referenced By:

Decoupling Using DDS and Publish-Subscribe
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [G1810.1]

Is there a conceptual data model that captures the data within the DCPS?

Procedure:

Look for a data model that captures the data within the Data-Centric Publish-Subscribe (DCPS). The following is a very short list of some of the files extensions that may contain data models.

CDM	Conceptual model file (PowerDesigner)
PDM	Physical model file (PowerDesigner)
ER1	ERWin file
ERX	ERWin file
ERM	Entity Relationship Diagram Model file (Prosa)

Example:

BP1038

Statement:

Use a **sans serif** font (e.g., Arial, Verdana) in Web pages rather than a serif font (e.g., Times New Roman).

Rationale:

Web pages are easier to read with **sans serif** fonts.

Referenced By:

Style Sheets
Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

GUI DesignDevelopment

BP1039

Statement:

Do not underline any text unless it is a link.

Rationale:

Underlined text is the default behavior of an **HTML** link. Many users consider this the norm and may find a **Web page** difficult to read if other items are underlined.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

BP1040

Statement:

Use hex codes for all colors (e.g., #FFFF33), never the color name (e.g., yellow).

Rationale:

Using hex codes for colors is a common industry practice to increase compatibility between browsers.

For an online hexadecimal color chart, see http://webmonkey.wired.com/webmonkey/reference/color_codes/.

Referenced By:

Style Sheets
Browser-Based Clients

Acquisition Phase:

Development

BP1041

Statement:

Do not change the default colors of the links.

Rationale:

Web pages are easier to read because users have become accustomed to the default colors.

Referenced By:

Style Sheets
Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

BP1042

Statement:

Do not build a **Web page** where the horizontal width is greater than the screen (vertical scrolling is fine), planning for the lowest common denominator to be super-VGA resolution (800 x 600).

Rationale:

This enables a user to print pages on most printers and render pages on most displays.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

BP1054

Statement:

Use standard controls that provide input choices for the user. These controls might include radio buttons, check boxes, list boxes, and drop-downs.

Rationale:

Reduces user input errors.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

BP1075

Statement:

All application developers should use the Apache **Ant** build tool to build, package, and deploy **Java EE** applications.

Rationale:

There are several good Integrated Development Environments (**IDEs**) on the market to support developing J2EE applications. However, different IDEs tend to auto-generate code that does not port to other IDEs, creating a problem when sharing code between groups using different IDEs. To minimize these compatibility issues and development environment turf wars, common building tools need to be used.

Referenced By:

Automate the Software Build Process

Acquisition Phase:

Development

BP1076

Statement:

When **deploying** a new application to a WebLogic **application server** (e.g., **ear**, **war**, **rar**), do not edit the WebLogic startup file to add application-specific information. This file is used for **server** startup only and should not contain application-specific logic. The system administrator must approve and coordinate all updates to this file.

Rationale:

Server startup should not depend on an individual application.

Referenced By:

Java EE Environment

Acquisition Phase:

Development

BP1077

Statement:

Do not edit the **`config.xml`** file manually.

Rationale:

The **`config.xml`** file is the persistent store used by the WebLogic server to store runtime configuration parameters. Editing the **`config.xml`** file manually can introduce unpredictable server errors and cause loss of important configuration data. Instead, use the WebLogic management console to configure the WebLogic **server**. Any edits done through the management console will be written to **`config.xml`** .

Referenced By:

Java EE Environment

Acquisition Phase:

Development

BP1097

Statement:

Use the **`System.Text.StringBuilder`** class for repetitive string modifications such as appending, removing, replacing, or inserting characters.

Rationale:

Strings in **.NET** are immutable. This means that every time a string is created as a result of a string operation such as concatenation, a new string is created for each intermediate string in a set of operations. This has a lot of string management overhead. **`StringBuilder`** avoids these problems.

Referenced By:

.NET Framework

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1097.1]

Are there repetitive string operations that use string operations instead of **`StringBuilder`** operations?

Procedure:

Scan all C# code for repetitive string operations such as appending, removing, replacing, or inserting characters.

Example:

None

BP1098

Statement:

Write all **.NET** code in C#.

Rationale:

Because of the high degree of similarities between C# and Java, .NET code written in C# is easily ported to Java. .NET has removed most of the advantages of one language (C#, C++, J++, VB) over another.

Referenced By:

.NET Framework

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1098.1]

Are any .NET languages delivered other than C#?

Procedure:

Scan delivered code for registered .NET file extensions other than C#.

Example:

None

BP1100

Statement:

Compile all **.NET** code using the .NET **Just-In-Time compiler**.

Rationale:

There are two different ways to generate machine code within the .NET environment: **Just-In-Time (JIT)** and Native Image Generator (**NGEN**). The NGEN method provides performance advantages by using the native image cache portion of the global assembly cache, which is specific to the machine where the .NET **common language runtime** is installed. It is machine-dependent and is less portable.

Referenced By:

.NET Framework

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1100.1]

Is **`ngen.exe`** used?

Procedure:

Scan all delivered code for the use of **`ngen.exe`** or the **ngen** command.

Example:

None

BP1111

Statement:

Mark all **Microsoft Message Queue** (**MSMQ**) messages as recoverable.

Rationale:

MSMQ normally only stores the contents of messages in memory, which will be lost if a power, hardware, or software failure occurs. By marking messages as recoverable, messages are also stored to disk so the contents can be recovered after a failure.

Referenced By:

Messaging with MSMQ

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1111.1]

Are all messages and message queues marked as recoverable?

Procedure:

Scan the code for the creation of messages and message codes, and make sure each has the **`recoverable`** attribute set to true.

Example:

None

BP1112

Statement:

Specify all **Microsoft Message Queue** (**MSMQ**) queues as transactional if they support multiple-step processes.

Rationale:

Transactions allow multi-step processes to behave correctly when a **rollback** occurs.

Referenced By:

Messaging with MSMQ

Acquisition Phase:

Development

BP1116

Statement:

If using **Java**-based messaging (e.g., **JMS**), register destinations in **Java Naming and Directory Interface** (**JNDI**) so **message clients** can use JNDI to look up these destinations.

Rationale:

JNDI is an industry standard for Java-based applications.

Referenced By:

Message-Based Applications
JNDI Security

Acquisition Phase:

Development

BP1139

Statement:

Do not use proprietary **SQL** extensions.

Rationale:

The use of proprietary extensions increases vendor dependence.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1139.1]

Have the developers adhered to a core set of features and minimized use of proprietary extensions to the **SQL** standard?

Procedure:

Examine a representative sample of database scripts and stored procedures.

Example:

None

BP1140

Statement:

Use SQL-2003 features in preference to **SQL-92** or **SQL-99**.

Rationale:

SQL-2003 includes many **XML** and **OODB** extensions and features. Use it in preference to SQL-99 or SQL-92 entry-level features to justify the recommendations against using native XML databases and OODB databases.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1140.1]

Have the developers used SQL-2003 features rather than SQL-92 or SQL-99 features?

Procedure:

Examine a representative sample of database scripts and stored procedures.

Example:

None

BP1143

Statement:

Use a **database modeling** tool that supports a two-level model (**Conceptual**/Logical and **Physical**) and **ISO-11179** data exchange standards.

Rationale:

ISO-11179 is a **metadata** repository standard. Supporting tools store the model locally in an **XML** file or in a vendor-specific repository. For many applications, there is no need to use the repository at all. **Configuration Management** could be affected by checking the model in and out of a tool such as Source Safe. Entity-Relationship **data model** is synonymous with a **Conceptual data model**.

Referenced By:

RDBMS Internals
Database Development

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1143.1]

Is a database modeling tool being used and does it support the **ISO-11179** data exchange standards?

Procedure:

Verify that the requirement for a database modeling tool is included in the system requirements. If ISO-11179 standard-based repository products become available, determine whether the product provides an interface thereto.

Example:

None

BP1145

Statement:

Use vendor-neutral **conceptual/logical models**.

Rationale:

The leading database vendors do not have a common set of data types or object name length limitations, and there are no **ANSI** standards that address these issues. To maintain vendor-neutral models, do not accept vendor-specific features.

Referenced By:

RDBMS Internals
Data Modeling
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1145.1]

Has the data model been designed using vendor-neutral design criteria?

Procedure:

Examine the conceptual/logical data model.

Example:

None

BP1227

Statement:

Do not allow installation of **MSMQ**-dependent clients.

Rationale:

MSMQ-dependent clients require synchronous access to an MSMQ server and create performance issues on the server. Consequently, dependent clients cannot operate if they are disconnected from the rest of the **enterprise** networks.

Dependent clients cannot be run under local accounts.

Dependent clients leave all encrypted messages in plain text between the client and server.

Referenced By:

RDBMS Internals
Messaging with MSMQ

Acquisition Phase:

Development

BP1230

Statement:

Do not use the **MSMQ** **`SupportLocalAccountsOrNT4`** feature.

Rationale:

This entry enables weakened security for Active Directory on a **domain** controller, which is then replicated to all other domain controllers in every domain in your forest.

See the Microsoft Message Queuing Web site for additional information.

Referenced By:

Messaging with MSMQ

Acquisition Phase:

Development

BP1231

Statement:

Use **`CORBA::String_var`** in **IDL** to pass string types in C++.

Rationale:

Follow this practice to correct memory management and reduce memory leaks and runtime faults.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1231.1]

Is **`String_var`** used in the implementation code that was not auto generated?

Procedure:

Check implementation code that was not autogenerated for all occurrences of "string" and verify that they are **`String_var`** .

Example:

None

BP1232

Statement:

Do not pass or return a zero or null pointer; instead, pass an empty string.

Rationale:

Follow this practice to correct memory management and reduce memory leaks and runtime faults.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1232.1]

Are there any returns that contain pointers that are assigned zero?

Procedure:

Check code to make sure that all strings returned always have a safety check for zero or null pointers, and assign them to empty strings.

Example:

None

BP1233

Statement:

Do not assign **`CORBA::String_var`** type to **`INOUT`** method parameters.

Rationale:

Follow this practice to correct memory management and reduce memory leaks and runtime faults.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1233.1]

Are there any implementation classes using methods that contain **`CORBA::String_var`**?

Procedure:

Inspect CORBA code to make sure **`INOUT`** parameters are not assigned to **`CORBA::String_var`** values.

Example:

None

BP1234

Statement:

Assign string values to **`OUT`** , **`INOUT`** , or **`RETURN`** parameters using operations to allocate or duplicate values rather than creating and deleting values.

Rationale:

Correct memory management and reduce memory leaks and reduce runtime faults.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1234.2]

Are new and delete operators being used for strings being assigned to **`OUT`**, **`INOUT`**, or **`RETURN`** parameters?

Procedure:

Inspect CORBA code to make sure **`OUT`**, **`INOUT`**, and **`RETURN`** parameters are not using strings managed with the new and delete operators.

Example:

None

2) Test: [BP1234.1]

Are **`string_dup`**, **`string_alloc`** and **`string_free`** being used?

Procedure:

Search CORBA code for the use of **`string_dup`**, **`string_alloc`**, and **`string_free`**.

Example:

None

BP1235

Statement:

Assign string values to returned-as-attribute values using operations to allocate or duplicate values rather than creating and deleting values.

Rationale:

Follow this practice to correct memory management and reduce memory leaks and runtime faults.

Referenced By:

CORBA

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1235.1]

Are **`string_dup`**, **`string_alloc`**, and **`string_free`** being used?

Procedure:

Search CORBA code for the use of **`string_dup`**, **`string_alloc`**, and **`string_free`**.

Example:

None

2) Test: [BP1235.2]

Are new and delete operators being used for strings being returned-as-attribute?

Procedure:

Inspect CORBA code to make sure returned-as-attribute string values are not using strings managed with the new and delete operators.

Example:

None

BP1240

Statement:

Present complete and coherent sets of concepts to the user.

Rationale:

The **interface** should not require the consumer continually to implement multiple interfaces when a single interface can accomplish the same thing.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

BP1241

Statement:

Design statically typed **interfaces**.

Rationale:

Designing a statically typed interface allows consumers to use early binding rather than late binding. This minimizes the risk for runtime errors due to late binding.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

BP1242

Statement:

Minimize an **interface's** dependencies on other **interfaces**.

Rationale:

Minimizing the dependency of an interface on other interfaces simplifies the use of the interface by consumers.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

BP1243

Statement:

Express **interfaces** in terms of application-level types.

Rationale:

Use application-level types to maintain the meaning of values used with the interface. This enables data validation and other runtime safety checks against the data.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

BP1244

Statement:

Use assertions only to aid development and **integration**.

Rationale:

Assertions allow evaluating Boolean expressions to determine if the code is executing within the proper operating constraints. For example, if a calculated temperature is supposed to be between -273 degrees and +1,000 degrees, it is possible to test the results of the calculation with an assertion. Once the code is tested and/or integrated, this calculation no longer needs to occur after each calculation.

Assertion execution is integrated into the **compiler**. Consequently, it is possible to add it into the executable or eliminate it by setting compiler options (i.e., switches). Assertions are therefore ideal for adding code that is useful during development or integration, but wasteful in delivered code.

Referenced By:

Public Interface Design

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1244.1]

Do public methods that implement interfaces have assertions?

Procedure:

Check all implementations of public interfaces to ensure that all public methods that are part of the interface do not use the **`assert`** command.

Example:

The following example shows a correct implementation of a public method in a public interface.

```
public interface NameInterface is
public String getName
  ( int nameID )
  Throws IllegalArgumentException
  {
    /* precondition check */
    if ( nameID <= 0
         || nameID > MAX_NAMES
       )
    { throw new IllegalArgumentException
        ("Illegal id number: " + nameID);
    }
    . . .// Do the computation
    return theResult;
  } // End getName
} // NameInterface
```

The following example shows an incorrect implementation of a public method in a public interface. Do not use the implementation exemplified by the red code.

```
public interface NameInterface is
public String getName
  ( int nameID )
  {
    /* precondition check */
    assert nameID <= 0
         || nameID > MAX_NAMES
       : "Illegal id number: " + nameID);
... . . .// Do the computation
    return theResult;
  } // End getName
} // NameInterface
```

BP1246

Statement:

Base Java-based portlets on **JSR 168**.

Rationale:

JSR 168 enables **interoperability** between Java **portlets** and **portals**. This specification defines a set of **APIs** for portal computing that addresses the areas of aggregation, **personalization**, presentation, and security. http://www.jcp.org/en/jsr/detail?id=168

Referenced By:

Web Portals

Acquisition Phase:

Development

BP1247

Statement:

Encapsulate Java-based **portlets** in a **.war** file.

Rationale:

Storing JSR-168-compliant code in the portal container improves **interoperability** and code reuse.

Referenced By:

Web Portals

Acquisition Phase:

Development

BP1248

Statement:

Follow a naming convention.

Rationale:

The names of schemas, users, tables, and columns need to be unique and descriptive. Unfortunately, it is possible (but undesirable) to give the same name to multiple objects; for example, assigning the name "employee" to a database, table, and column. Many naming conventions get around this by appending a suffix that indicates the kind of object: for example, **`Employee_Db`**, **`Employee_Tbl`**, **`Employee_Id`**, **`Employee_Indx`**.

Avoid generic column names such as "ID." Systems often have many kinds of IDs, and even if the system really only does have a single ID, it will be more difficult to merge with other databases if they have also used the column name "ID."

Some DBMSs support mixed-case names of unlimited length, while others are case-insensitive. For portability, assume that names are case-insensitive and limited to 30 characters. Do not use reserved words from the **SQL-92**, **SQL:1999**, or SQL:2003 standards.

Justifies:

BP1249 BP1251 BP1250 BP1253 BP1252 BP1254

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1248.1]

Is there a naming convention?

Procedure:

Check for the existence of a document that governs naming conventions, or look for patterns in the database metadata.

Example:

Use database commands to look at the database metadata:

```
select username from all_users
select table_name from user_tables
select index_name from user_indexes
```

BP1249

Statement:

Do not use generic names for database objects such as databases, schema, users, tables, views, or indices.

Rationale:

Assigning generic names to user-defined objects within a database can lead to confusion and unexpected results. For example, naming a database "instance" within the **RDBMS** database is confusing to the humans who have to read commands that reference the database. In addition, the RDBMS software may parse it incorrectly.

> ***Note:*** *Although some RDBMS interpreters allow the use of a generic or reserved word to name objects if the name is surrounded with quotes, this is not a recommended practice.*

Derived From:

BP1248

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1249.1]

Are any generic names used for user-defined objects?

Procedure:

Examine the RDBMS metadata for generic names such as database, table, entity, column, attribute, select, view, etc.

Example:

```
select table_name from user_tables where table_name in ('database','entity',...)
select column_name from user_tab_columns where column_name in ('database','entity',...)
```

BP1250

Statement:

Use case-insensitive names for database objects such as databases, schema, users, tables, views, and indices.

Rationale:

The **SQL** standard does not require names to be case-sensitive. Consequently, some DBMSs are not case-sensitive. Using case-sensitive names, therefore, makes portability more difficult.

Derived From:

BP1248

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1250.1]

Are the names of database objects case-sensitive?

Procedure:

Examine the database metadata for "run-on" names. If the database supports case-sensitive names, check to see if it is using camel-back capitalization.

Example:

```
EMPLOYEEBENEFITSTBL
EmployeeBenefitsTbl
```

BP1251

Statement:

Separate words with underscores.

Rationale:

The **SQL** standard does not require names to be case-sensitive. Consequently, some DBMSs are not case-sensitive. Using case-sensitive names, therefore, makes portability more difficult. To avoid these problems, use underscores to separate words (**`employee_benefits_tbl`**) rather than camel-back capitalization (**`EmployeeBenefitsTbl`**).

Derived From:

BP1248

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1251.1]

Are underscores used between the words in the names of database objects?

Procedure:

Examine the database metadata and look for names that do not have underscores separating words.

Example:

```
EMPLOYEEBENEFITSTBL versus
EMPLOYEE_BENEFITS_TBL
EmployeeBenefitsTbl versus
Employee_Benefits_Tbl
```

BP1252

Statement:

Do not use names with more than 30 characters.

Rationale:

Not all DBMSs support unlimited name lengths. For example, Oracle limits object names to 30 characters. Therefore, using names longer than 30 characters can reduce portability by limiting the DBMSs on which the system can be deployed.

Derived From:

BP1248

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1252.1]

Are any of the database object names more than 30 characters in length?

Procedure:

Examine the database metadata and look for names that are longer than 30 characters.

Example:

```
W2_EMPLOYEE_BENEFITS_FOR_FAMILIES_TBL
```

BP1253

Statement:

Do not use the **SQL:1999** or SQL:2003 reserved words as names for database objects such as databases, schema, users, tables, views, or indices.

Rationale:

Using reserved words as the names of database objects can cause ambiguities and errors. It limits the ability to upgrade or port the code to other systems.

Derived From:

BP1248

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1253.1]

Are any of the **SQL:1999** or SQL:2003 reserved words used to name objects in the database?

Procedure:

Examine the database metadata for names that are in the list of SQL:1999 or SQL:2003 reserved words

Example:

Look for any of these words:

```
ABS ABSOLUTE ACCESS ACQUIRE ACTION ADA ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALLOW ALTER AND ANY ARE
ARRAY AS ASC ASENSITIVE ASSERTION ASUTIME ASYMMETRIC AT ATOMIC AUDIT AUTHORIZATION AUX AUXILIARY AVG
BACKUP BEFORE BEGIN BETWEEN BIGINT BINARY BIT BIT_LENGTH BLOB BOOLEAN BOTH BREADTH BREAK BROWSE BUFFERPOOL
BULK BY
CALL CALLED CAPTURE CARDINALITY CASCADE CASCADED CASE CAST CATALOG CCSID CEIL CEILING CHAR CHAR_LENGTH
CHARACTER CHARACTER_LENGTH CHECK CHECKPOINT CLASS CLOB CLOSE CLUSTER CLUSTERED COALESCE COLLATE COLLATION
COLLECT COLLECTION COLLID COLUMN COMMENT COMMIT COMPLETION COMPRESS COMPUTE CONCAT CONDITION CONNECT
CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTAINS CONTAINSTABLE CONTINUE CONVERT CORR CORRESPONDING
COUNT COUNT_BIG COVAR_POP COVAR_SAMP CREATE CROSS CUBE CUME_DIST CURRENT CURRENT_COLLATION CURRENT_DATE
CURRENT_DEFAULT_TRANSFORM_GROUP CURRENT_LC_PATH CURRENT_PATH CURRENT_ROLE CURRENT_SERVER CURRENT_TIME
CURRENT_TIMESTAMP CURRENT_TIMEZONE CURRENT_TRANSFORM_GROUP_FOR_TYPE CURRENT_USER CURSOR CYCLE
DATA DATABASE DATALINK DATE DAY DAYS DB2GENERAL DB2SQL DBA DBCC DBINFO DBSPACE DEALLOCATE DEC DECIMAL DECLARE
DEFAULT DEFERRABLE DEFERRED DELETE DENSE_RANK DENY DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR
DETERMINISTIC DIAGNOSTICS DICTIONARY DISALLOW DISCONNECT DISK DISTINCT DISTRIBUTED DLNEWCOPY DLPREVIOUSCOPY
DLURLCOMPLETE DLURLCOMPLETEONLY DLURLCOMPLETEWRITE DLURLPATH DLURLPATHONLY DLURLPATHWRITE DLURLSCHEME
DLURLSERVER DLVALUE DO DOMAIN DOUBLE DROP DSSIZE DUMMY DUMP DYNAMIC
EACH EDITPROC ELEMENT ELSE ELSEIF END END-EXEC EQUALS ERASE ERRLVL ESCAPE EVERY EXCEPT EXCEPTION EXCLUSIVE
EXEC EXECUTE EXISTS EXIT EXP EXPLAIN EXTERNAL EXTRACT
```

```
FALSE FENCED FETCH FIELDPROC FILE FILLFACTOR FILTER FINAL FIRST FLOAT FLOOR FOR FOREIGN FORTRAN FOUND FREE
FREETEXT FREETEXTTABLE FROM FULL FUNCTION FUSION
GENERAL GENERATED GET GLOBAL GO GOTO GRANT GRAPHIC GROUP GROUPING
HANDLER HAVING HOLD HOLDLOCK HOST HOUR HOURS
IDENTIFIED IDENTITY IDENTITY_INSERT IDENTITYCOL IF IGNORE IMMEDIATE IMPORT IN INCLUDE INCREMENT INDEX
INDICATOR INITIAL INITIALIZE INITIALLY INNER INOUT INPUT INSENSITIVE INSERT INT INTEGER INTEGRITY INTERSECT
INTERSECTION INTERVAL INTO IS ISOBID ISOLATION ITERATE
JAR JAVA JOIN
KEY KILL
LABEL LANGUAGE LARGE LAST LATERAL LC_CTYPE LEADING LEAVE LEFT LESS LEVEL LIKE LIMIT LINENO LINKTYPE LN LOAD
LOCAL LOCALE LOCALTIME LOCALTIMESTAMP LOCATOR LOCATORS LOCK LOCKSIZE LONG LOOP LOWER
MAP MATCH MAX MAXEXTENTS MEMBER MERGE METHOD MICROSECOND MICROSECONDS MIN MINUS MINUTE MINUTES MOD MODE
MODIFIES MODIFY MODULE MONTH MONTHS MULTISET
NAME NAMED NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NHEADER NO NOAUDIT NOCHECK NOCOMPRESS NODENAME
NODENUMBER NONCLUSTERED NONE NORMALIZE NOT NOWAIT NULL NULLIF NULLS NUMBER NUMERIC NUMPARTS
OBID OBJECT OCTET_LENGTH OF OFF OFFLINE OFFSETS OLD ON ONLINE ONLY OPEN OPENDATASOURCE OPENQUERY OPENROWSET
OPENXML OPERATION OPTIMIZATION OPTIMIZE OPTION OR ORDER ORDINARILITY OUT OUTER OUTPUT OVER OVERLAPS OVERLAY
PACKAGE PAD PAGE PAGES PARAMETER PARAMETERS PART PARTIAL PARTITION PASCAL PATH PCTFREE PCTINDEX PERCENT
PERCENT_RANK PERCENTILE_CONT PERCENTILE_DISC PIECESIZE PLAN POSITION POSTFIX POWER PRECISION PREFIX PREORDER
PREPARE PRESERVE PRIMARY PRINT PRIOR PRIQTY PRIVATE PRIVILEGES PROC PROCEDURE PROGRAM PSID PUBLIC
QUERYNO
RAISERROR RANGE RANK RAW READ READS READTEXT REAL RECONFIGURE RECOVERY RECURSIVE REF REFERENCES REFERENCING
REGR_AVGX REGR_AVGY REGR_COUNT REGR_INTERCEPT REGR_R2 REGR_SLOPE REGR_SXX REGR_SXY REGR_SYY RELATIVE RELEASE
RENAME REPEAT REPLICATION RESET RESIGNAL RESOURCE RESTORE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE
ROLLBACK ROLLUP ROUTINE ROW ROW_NUMBER ROWCOUNT ROWGUIDCOL ROWID ROWNUM ROWS RRN RULE RUN
SAVE SAVEPOINT SCHEDULE SCHEMA SCOPE SCRATCHPAD SCROLL SEARCH SECOND SECONDS SECQTY SECTION SECURITY SELECT
SENSITIVE SEQUENCE SESSION SESSION_USER SET SETS SETUSER SHARE SHUTDOWN SIGNAL SIMILAR SIMPLE SIZE SMALLINT
SOME SOURCE SPACE SPECIFIC SPECIFICTYPE SQL SQLCA SQLCODE SQLERROR SQLEXCEPTION SQLSTATE SQLWARNING SQRT
STANDARD START STATE STATEMENT STATIC STATISTICS STAY STDDEV_POP STDDEV_SAMP STOGROUP STORES STORPOOL
STRUCTURE STYLESUBPAGES SUBSTRING SUCCESSFUL SUM SYMMETRIC SYNONYM SYSDATE SYSTEM SYSTEM_USER
TABLE TABLESPACE TEMPORARY TERMINATE TEXTSIZE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TOP
TRAILING TRAN TRANSACTION TRANSLATE TRANSLATION TREAT TRIGGER TRIM TRUE TRUNCATE TSEQUAL TYPE
UID UNDER UNDO UNION UNIQUE UNKNOWN UNNEST UNTIL UPDATE UPDATETEXT UPPER USAGE USE USER USING
VALIDATE VALIDPROC VALUE VALUES VAR_POP VAR_SAMP VARCHAR VARCHAR2 VARIABLE VARIANT VARYING VCAT VIEW VOLUMES
WAITFOR WHEN WHENEVER WHERE WHILE WIDTH_BUCKET WINDOW WITH WITHIN WITHOUT WLM WORK WRITE WRITETEXT
YEAR YEARS
ZONE
```

BP1254

Statement:

For **command-and-control** systems, use the names defined in the **C2IEDM** for data exposed to the outside communities.

Rationale:

The **Command-and-Control** (**C2**) **COI** has developed a **data model** to facilitate the exchange of data within the community and by consumers of their data outside the community. Therefore, data that is to be exposed from the database to the **COI** community or its data consumers should defer to the **data model** whenever possible. The C2IEDM [R1070] data model defines the data units as well as the names and structure of the data.

Derived From:

BP1248

Referenced By:

RDBMS Internals
Database Development
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

null

Evaluation Criteria:

1) Test: [BP1254.1]

If this is a system, does it use for the data that is exposed to the outside world?

Procedure:

Review all the data that is exposed to the outside world and confirm that it conforms to the C2IEDM specifications.

Example:

None

BP1255

Statement:

Use **surrogate keys**.

Rationale:

A surrogate key, also referred to as a system-generated key, database-sequence number, or arbitrary unique identifier, is a unique, arbitrary **primary key**. The **RDBMS** usually generates the surrogate key, but a database access layer such as the middle tier can also generate the surrogate key. The surrogate key is arbitrary because it is not derived from any data that exists within the table or the database. Two other options for surrogate keys are

Universally Unique Identifiers (UUIDs) (http://en.wikipedia.org/wiki/Universally_Unique_Identifier)

Globally Unique Identifiers (GUIDs) (http://en.wikipedia.org/wiki/Globally_Unique_Identifier)

Justifies:

BP1256 BP1257

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1256

Statement:

Use surrogate keys as the **primary key**.

Rationale:

Instead of using the natural keys to identify each record uniquely, use a surrogate key. This allows the natural key information to be modified independently of the primary key and any foreign-key references to the key.

Derived From:

BP1255

Referenced By:

RDBMS Internals
Database Development

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1256.1]

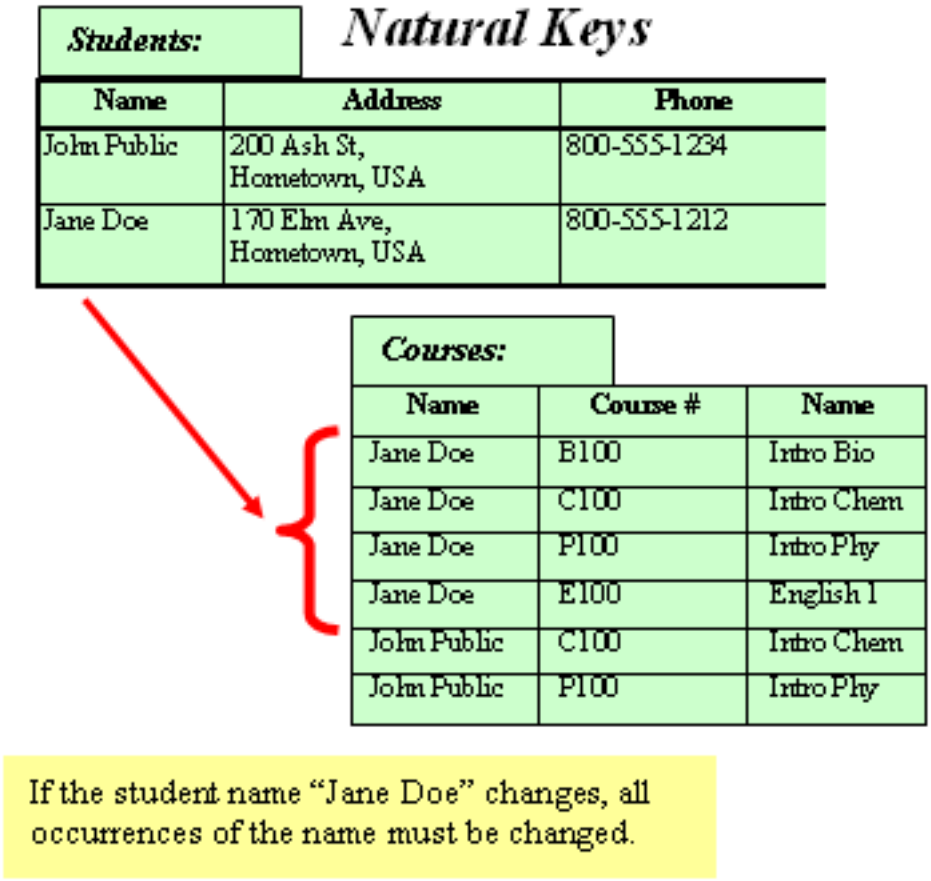
Are surrogate keys used instead of natural keys?

Procedure:

Look at the database metadata and determine if it uses surrogate or natural keys.

Example:

The following example shows natural keys. The primary keys are made up completely or in part from naturally occurring data in the tables.



The following example shows a surrogate key being used instead of a natural key. Maintaining data is less complex than it is with natural keys and consequently less error-prone.

BP1257

Statement:

Place a **unique key constraint** on the **natural key** fields.

Rationale:

Surrogate keys make it easier to maintain data. However, a column or set of columns should still uniquely identify the row in the table. This column or set of columns is the "natural key" or "secondary key." This natural key should still be protected by the uniqueness constraint normally associated with a **primary key**.

Derived From:

BP1255

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1257.1]

Is there a unique key index for all tables that includes a column or set of columns not including the primary key?

Procedure:

Look at the database metadata to ensure that each table has a unique key, and that the columns in the unique key are not also part of the primary key.

Example:

BP1258

Statement:

Explicitly define the encoding style of all data transferred via **XML**.

Rationale:

By default, **XML** is encoded using **Unicode**. Consequently, data transferred via XML should explicitly specify the encoding style. Assuming the default can cause **interoperability** problems between implementations.

Note: *Look for the following XML tag as the first line returned from queries that return XML from the database:*

```
<?xml version="1.0" encoding="UTF-8"?>
```

Referenced By:

RDBMS Internals
XML Syntax

Acquisition Phase:

Development

BP1259

Statement:

Use indexes.

Rationale:

An index in an **RDBMS** is a summary of information organized to minimize the search time. Indexes summarize the information in a table. So, an employee table might have an index of last names, or last name and first name.

Having additional indexes on tables involves a tradeoff between query performance and insert/update/delete performance, which requires underlying index maintenance.

Justifies:

BP1260 BP1261 BP1262

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1260

Statement:

Define a **primary key** for all tables.

Rationale:

By definition, a **primary key** uniquely defines each row within a table. To optimize the use of the table and to find records by the primary key, there should be an index that enforces the uniqueness of the key.

Derived From:

BP1259

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1260.1]

Is there a primary key defined for each table listed in the database?

Procedure:

Examine the database metadata to ensure there is a primary key for each table in the database.

Example:

BP1261

Statement:

Monitor and tune indexes according to the response time during normal operations in the production environment.

Rationale:

Index efficiency depends on the data being indexed. Common variables follow:

- A sparsely populated table versus a densely populated table
- Data added in an presorted order versus a random order

Consequently, as the data changes, the efficiency of the index changes.

Derived From:

BP1259

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1262

Statement:

In the case of Oracle, define indexes against the **foreign keys** (**FK**) columns to avoid contention and locking issues.

Rationale:

Derived From:

BP1259

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1263

Statement:

Gather storage requirements in the planning phase, and then allocate twice the estimated storage space.

Rationale:

Storage space on the disk always poses a problem for databases, so it is necessary to plan storage space carefully.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1264

Statement:

For **high availability**, use hardware solutions when geographic proximity permits.

Rationale:

There are many ways to achieve high availability. Some are based on hardware and others on software. As a general rule, hardware solutions use simple redundancy and are consequently less complex and fragile. If geographic proximity is not an issue, the hardware solution is preferable.

Referenced By:

RDBMS Internals

Acquisition Phase:

Development

BP1265

Statement:

Validate **XML** idocuments during document generation.

Rationale:

All **XML** passed between two systems or services must be valid. The XML document generator is responsible for ensuring that the document is valid and **well-formed**. If there are problems, the document generator is the only user that can effectively change the document.

Validity is checked via the use of a **W3C** Standard Validating parser. These parsers are built into most XML editors but are also available as stand alone products. Either the XML is valid or diagnostics are returned indicating where the XML is invalid.

Referenced By:

XML Validation

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1265.1]

Are all the documents exported from the system or service valid and **well-formed**?

Procedure:

Capture all the documents and validate them, using an XML editor or stand alone XML validation tool.

Example:

None

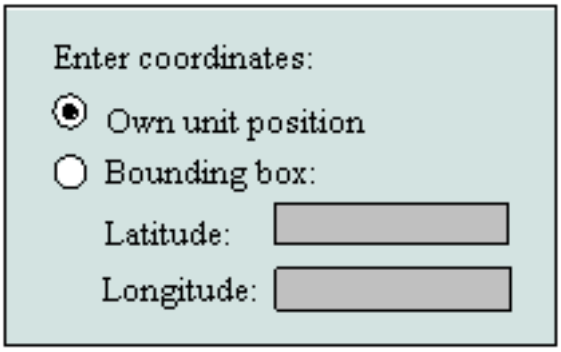
BP1272

Statement:

Disable dependent child controls when the parentcontrol is inactive.

Rationale:

This practice makes it easier for the user to understand that the child controls depend on the selection of the parent, contributing to data integrity.



Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

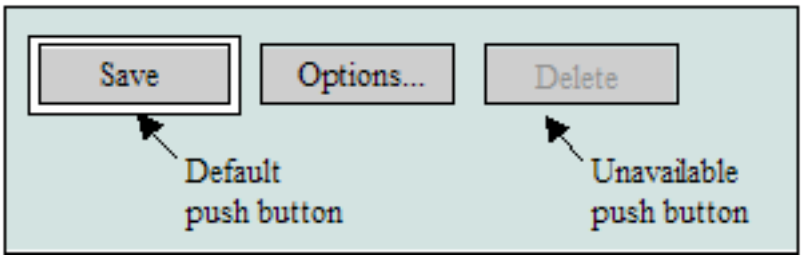
BP1273

Statement:

Gray out the push button label if a button is unavailable.

Rationale:

This practice makes it easier for the user to understand that the button cannot be used until other action is taken.



Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

BP1280

Statement:

In tabular data displays, right justify integer data.

Rationale:

Whole numbers, displayed in a column, are easier to read if the digits of the same magnitude (1's, 10's, 100's, etc.) are vertically aligned.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1280.1]

Are all tabular whole number data right-justified?

Procedure:

Search all style sheets for the word "text-align." Examine the results for tabular whole number data and make sure the "text-align" attribute is set to "right"; visual Web page inspection may necessary to see if a defined align style is used within the tabular data.

Example:

Correct usage:

Cascading style sheet:

```
.td-items {
   text-align:right;
}
```

HTML:

Incorrect usage:

No alignment or incorrect alignment used.

BP1281

Statement:

In tabular data displays, justify numeric data with decimals by using the decimal point.

Rationale:

It is common practice to align non-whole numbers by the decimal point for readability.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1281.1]

Are all tabular non-whole number data justified by decimal point?

Procedure:

Search all style sheets for the word "text-align." Examine the results for tabular non-whole number data and make sure the "text-align" attribute is set to "."; visual Web page inspection may be necessary to see if a defined align style is used within the tabular data.

Example:

Correct usage:

Cascading style sheet:

```
.td-subtotal {
   text-align:".";
}
```

HTML:

Incorrect usage:

No alignment or incorrect alignment used.

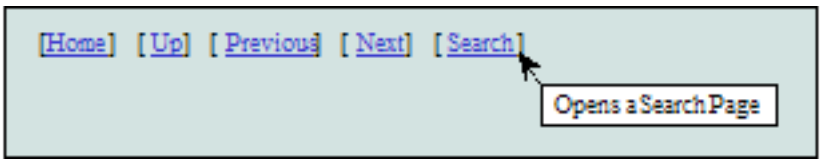
BP1290

Statement:

Use a tool tip to display help information about a control when the purpose of the control is not self-evident.

Rationale:

Using a tool tip increases user efficiency by preventing click errors. A mouse over event is the typical mapping for invoking a tool tip.



Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

BP1291

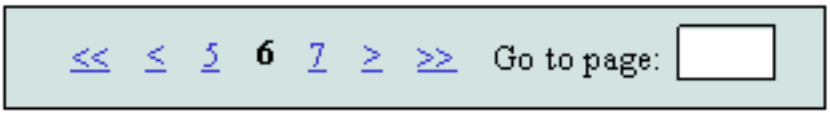
Statement:

Use obvious navigation controls for moving between pages in search results that span multiple pages.

Rationale:

Obvious navigation controls help a user to identify and use paging controls quickly. For example,

<	navigate back one page
>	navigate forward one page
<<	navigate back to the beginning page
>>	forward to the end page



Referenced By:

Browser-Based Clients
Human-Computer Interaction

Acquisition Phase:

Development

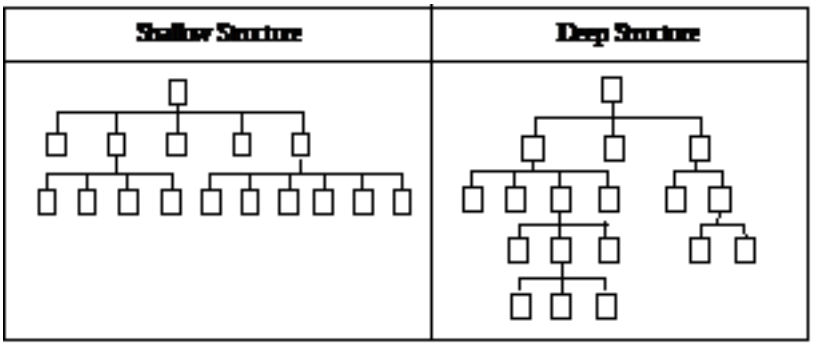
BP1297

Statement:

Structure a Web site hierarchy so users can reach important information and/or frequently accessed functions in a maximum of three jumps.

Rationale:

Use a shallow structure rather than a deep structure. A user's success at finding a target drops off sharply after three clicks.



Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

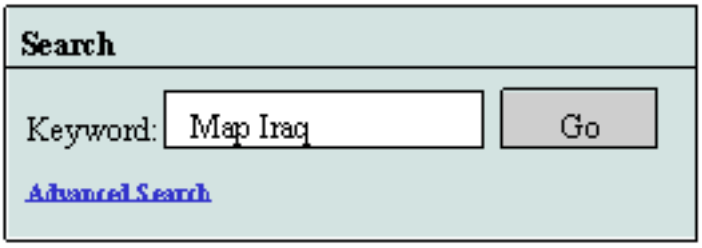
BP1298

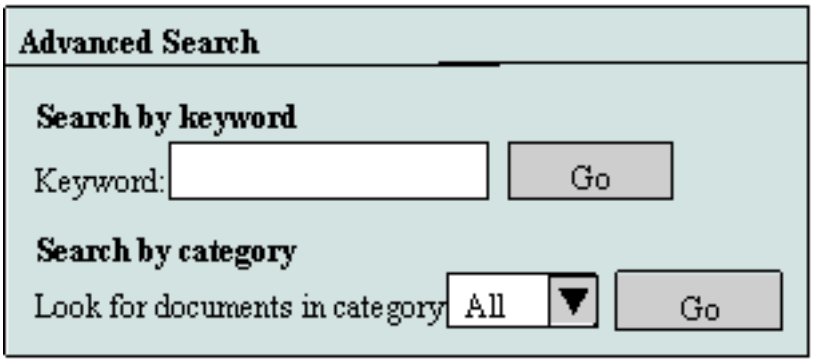
Statement:

Provide basic search functionality as the default with a link or button that provides more advanced search features.

Rationale:

This practice makes the search feature cleaner and easier to use because the advanced features are hidden.





Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

BP1299

Statement:

Include a link back to the home page on all Web pages.

Rationale:

A link back to a Web site home page, for example in the form of a logo and a regular HTML link called **`Home`**, helps users navigate the Web site.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

BP1353

Statement:

Use a data abstraction layer between the RDBMS and application for externally-visible applications to prevent the disclosure of sensitive data.

Rationale:

Large volume commercial online retailers often store customer data in an RDBMS, but they use a data abstraction layer with limited privileges to access that data from their Web services and other externally-visible applications. This more fully protects the data in the database from unauthorized access and modification.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1353.1]

Does the application protect sensitive data by using a data abstraction layer between the application and RDBMS?

Procedure:

Check that sensitive data is not readable and modifiable externally by the application.

Example:

BP1355

Statement:

Do not design the database around the requirements of an application.

Rationale:

Databases often outlive applications (i.e., legacy databases and evolution of applications). Database can also support multiple applications. If design of the database were around the application, it may present security holes that other applications could exploit. It is better to design the application around the rules set by the database.

Referenced By:

RDBMS Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1355.1]

Is application business logic or rules not found in the database?

Procedure:

Make sure data validation is done at database even if it is already being done at the application level.

Example:

None

BP1360

Statement:

Use the **XML** Infoset standard to serialize messages.

Rationale:

XML signatures rely on a character-by-character comparison for proper operations. A one character difference is a different result. So using a standard for serialization is very important to successful communications.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1360.1]

Does the Web service user serialize messages using the **XML** Infoset Standard?

Procedure:

Generate a test message and check it for compliance with the XML Infoset Standard.

Example:

None

2) Test: [BP1360.2]

Does the Web service provider serialize messages using the XML Infoset Standard?

Procedure:

Generate a test message and check it for compliance with the XML Infoset Standard.

Example:

None

BP1375

Statement:

Use Asymmetric Encryption.

Rationale:

Most Web services exchange very few messages so the fact that asymmetric encryption is computationally intensive is a non-issue. Symmetric encryption is more efficient, but it is done by sharing a secret key outside the SOAP message communication which is less portable.

Referenced By:

XML Web Service Security

Acquisition Phase:

Development

BP1394

Statement:

Identify, publish and validate data objects exposed to the enterprise early in the data engineering process and update in a spiral fashion as system development proceeds.

Rationale:

Referenced By:

Data Modeling

Acquisition Phase:

Development

BP1396

Statement:

Develop high-level conceptual data models for new systems prior to Milestone A based on the business process context in which the system will be used.

Rationale:

An early high-level understanding of the data objects/entities involved in a system can help to clarify the purpose and context of the system and identify potential downstream interoperability issues.

Referenced By:

Data Modeling

Acquisition Phase:

Development

BP1397

Statement:

For new systems, identify and develop use cases or reuse existing use cases as appropriate as early in the data engineering process as possible to support **data model** development.

Rationale:

Referenced By:

Data Modeling
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

BP1398

Statement:

Develop Interaction models as appropriate.

Rationale:

Referenced By:

Data Modeling

Acquisition Phase:

Development

BP1400

Statement:

Programs will use authoritative **metadata** established by the Joint Mission Threads (JMTs) when available.

Rationale:

Referenced By:

Data Modeling

Acquisition Phase:

Development

BP1404

Statement:

For DoD Programs requiring a **data model**, the **NATO** Generic Hub v.5 model (**LC2IEDM**) is an example of a successful **COI**-developed model.

Rationale:

The **Land C2 Information Exchange Data Model** (**LC2IEDM**), or Generic Hub (GH, now version 5) model has been under development in the **NATO** environment. This model is a rich Joint battlespace operational context model. Many NATO countries have developed prototypes. The U.S. Army has also been active in the Generic Hub efforts.

Derived From:

G1141

Referenced By:

Metadata Registry
Reading/Writing Objects within a DDS Domain

Acquisition Phase:

Development

BP1408

Statement:

Use a **semantic** description language such as **Web Ontology Language** (**OWL**) or **Resource Definition Framework** (**RDF**) to represent an **Ontology**.

Rationale:

Data producer recommendations are still maturing for how to handle data producers interaction with **Web Ontology Language** (**OWL**) or **Resource Definition Framework** (**RDF**).

Referenced By:

Metadata

Acquisition Phase:

Development

BP1409

Statement:

Register **Web services** using **Web Services Description Language** (**WSDL**) and **Universal Description, Discovery, and Integration** (**UDDI**).

Rationale:

Ontology languages such as **Web Ontology Language** (**OWL**) or **Resource Definition Framework** (**RDF**) are currently immature.

Referenced By:

Metadata

Acquisition Phase:

Development

BP1567

Statement:

Use the <**`abbr`**> and <**`acronym`**> tags to specify the expansion of acronyms and abbreviations.

Rationale:

Provides the user with easy access to the meaning of abbreviations and acronyms.

Referenced By:

Browser-Based Clients

Acquisition Phase:

Development

BP1568

Statement:

Use markup language (if available) and styles to represent mathematical equations.

Rationale:

Using a markup language such as MathML to display the equation rather than creating a separate image to display, makes the display easier to create and maintain and more flexible within the page layout.

Referenced By:

Browser-Based Clients

Acquisition Phase:

Development

BP1715

Statement:

Design SCA log services according to the OMG Lightweight Log Service Specification.

Rationale:

One component of the SCA framework is a central logging facility, enabling the asynchronous collection of informational messages from any component connected to the framework; and the controlled read access to this information. The Lightweight Logging Service is a free-standing, self-contained service which is not connected to an event channel or similar infrastructure. Using a standard log service specification between SCA implementations can improve interoperability and portability.

Referenced By:

Software Communication Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1715.1]

Is the logging service designed according to the OMG Lightweight Log Service Specification? Is the logging service designed according to the OMG Lightweight Log Service Specification?

Procedure:

Check the log service provider's documentation for compliance with the OMG Lightweight Log Service Specification.

Example:

BP1716

Statement:

Develop applications for SCA-compliant systems using a standard higher order language.

Rationale:

Developing SCA applications in higher order languages such as C enables independence from platform dependencies and helps ensure portability.

Referenced By:

Software Communication Architecture

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1716.1]

Does the application use a higher order language such as C rather than a lower order language such as Assembly?

Procedure:

Check what programming language is used to develop the SCA application.

Example:

BP1720

Statement:

Do not use commonly predefined VHDL identifier names for other identifiers.

Rationale:

The use of predefined identifiers causes confusion and some compilers and simulators have difficulty dealing with such identifiers. This reduces code portability.

> ***Note:*** *This practice has been adapted from Cohen, section 2.1.1.2.*

Referenced By:

VHDL Coding and Design

Evaluation Criteria:

1) Test: [BP1720.1]

Are any of the following predefined identifier names used, including the identifiers in the Std and IEEE design libraries: FF, Time, Min, Ns, Ms, ACK, Real, Std, On?

Procedure:

Check all other identifiers and make sure they are not the names of any predefined identifiers.

Example:

None

BP1721

Statement:

Define a VHDL package for closely related VHDL items that support an application function.

Rationale:

A package represents a module that allows the specification of groups of logically related declarations. Frequently used pieces of VHDL code are usually written in the form of components, functions, or procedures. These pieces are then placed into a package and compiled into the destination library. This technique allows code partitioning, code sharing, and code reuse.

> ***Note:*** *This practice has been adapted from Cohen, section 8.1, and Pedroni, section 10.2.*

Referenced By:

VHDL Coding and Design

Evaluation Criteria:

1) Test: [BP1721.1]

Do the packages contain functionally related components, functions and procedures?

Procedure:

Check the code and make sure all packages contain functionally related components, functions and procedures.

Example:

None

BP1722

Statement:

Employ VHDL components for commonly used VHDL described circuits.

Rationale:

A component is a special piece of conventional code that allows the construction of hierarchical designs. In other words, by declaring a piece of code as a component, that code can then be used within another circuit. This is just an additional way of partitioning a design and promoting code reuse and composability.

> ***Note:*** *This practice been adapted from Pedroni, section 10.3.*

Referenced By:

VHDL Coding and Design

Evaluation Criteria:

1) Test: [BP1722.1]

Are commonly used circuit modules described as components?

Procedure:

Check the code and make sure commonly used circuit modules are described as components.

Example:

None

BP1723

Statement:

Do not use guarded signals.

Rationale:

Guarded signals are not synthesizable and not commonly used. Guarded signals reduce the readability of code because the guards and drivers are not collected.

> ***Note:*** *This practice has been adapted from Cohen, section 6.2.7.1.*

Referenced By:

VHDL Synthesizable Design

Evaluation Criteria:

1) Test: [BP1723.1]

Does the signal kind (e.g. register, bus) appear in a signal declaration?

Procedure:

Check the signal declaration to see if the signal kind is stated. If so, the signal declared is a guarded signal of the kind indicated.

Example:

None

BP1732

Statement:

Follow the Upper Camel Case (UCC) naming convention for XML Type names.

Rationale:

The predominate style used by most programs or projects is to use the Upper Camel Case (UCC) for type names. Type names should be easy to differentiate from namespace prefixes and from attributes. Since the namespace prefix and the type name are separated by a non-whites character (i.e., the colon, :), it is easier to identify the type name from the namespace name if the type name follows the UCC.

Derived From:

G1730

Referenced By:

Defining XML Schemas
Defining XML Types

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1732.1]

Do type names follow the Upper Camel Case (UCC) naming convention?

Procedure:

Examine the schema definition and verify that the type names follow the Upper Camel Case (UCC) name convention.

Example:

```
<xsd:complexType
   name="MyType"
   . . .
</ xsd:coplexType>
```

BP1733

Statement:

Follow the Upper Camel Case (UCC) naming convention for XML Element names.

Rationale:

The predominate style used by most programs or projects is to use the Upper Camel Case (UCC) for element names. Element names should be easily differentiable from namespace prefixes and from attributes. Since the namespace prefix and the element name are separated by a non-whites character (i.e., the colon, :), it is easier to identify the element name from the namespace name if the element name follows the UCC.

Derived From:

G1730

Referenced By:

Defining XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1733.1]

Do element names follow the Upper Camel Case (UCC) naming convention?

Procedure:

Examine the schema definition and verify that the element names follow the Upper Camel Case (UCC) name convention.

Example:

BP1734

Statement:

Follow the Lower Camel Case (LCC) naming convention for XML Attributes.

Rationale:

The predominate style used by most programs or projects is to use the Lower Camel Case (LCC) for attribute names. Attributes are part of an attribute list which is a set of name="value" expressions separated by whitespace. Therefore, it is easy to find the beginning of the attribute name.

Derived From:

G1730

Referenced By:

Defining XML Schemas

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1734.1]

Do type names follow the Upper Camel Case (UCC) naming convention?

Procedure:

Examine the schema definition and verify that the type names follow the Upper Camel Case (UCC) name convention.

Example:

BP1739

Statement:

Use the xsd qualifying prefix for XML Schema namespace.

Rationale:

Syntactically there is no reason why the XML Schema namespace can not be given any qualifier. However, for readability on the part of humans, using the xsd qualifier is clear, precise, concise and widely accepted.

Referenced By:

Using XML Namespaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1739.1]

Does the XML schema use the xsd prefix for the XMLSchema namespace?

Procedure:

Look for the use of the XMLSchema namespace declaration and verify that the prefix is xsd.

Example:

The following is an example of using the xsd prefix for the XML Schema namespace:

```
<xsd:schema>
```

BP1741

Statement:

Do not provide a schema location in import statements in schemas.

Rationale:

An import statement allows schema components from other schemas to be added to the current schema. The added schema components are associated with a namespace defined in the import statement. The import statement provides for the imported schema to also be optionally associated with a location where the schema can be found. Associating a schema location with a namespace during the import is referred to as early binding. This locks the definition to a specific implementation.

The following example highlights these points:

WeatherStation Schema Definition

A weather station is defined as a collection of sensors with definitions that are to-be-determined.

Note: *The import of the http://www.Sensor.org without specifying the optional schema location.*

Note: *The use of the dangling type SensorType for the element Sensor. SensorType is bound later to a schema definition.*

SensorSupplier Schema Definition

A sensor supplier creates a sensor specific definition for a sensor.

WeatherStation Instance Document

A weather station instance document is created which binds the sensor suppliers definition of a sensor to the weather station. This allows the definition of the sensor to change or the location of the sensor definition (i.e. xsd) to change independently of the definition of the weather station.

Referenced By:

Using XML Namespaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1741.1]

Does the schema definition provide location for the imported schemas?

Procedure:

Examine the schema definition and make sure the schemaLocation attribute is not used in the import statement.

Example:

BP1742

Statement:

Use the xsi qualifying prefix for XML Schema instance namespace uses.

Rationale:

Syntactically there is no reason why the XML Schema instance namespace can not be given any qualifier. However, for readability on the part of humans, using the xsi qualifier is clear, precise, concise and widely accepted.

Referenced By:

Using XML Namespaces
XML Instance Documents

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1742.1]

Does the schema use the xsd prefix for the XMLSchema instance namespace?

Procedure:

Look for the use of the XMLSchema instance namespace declaration and verify that the prefix is xsi.

Example:

The following is an example of using the xsi prefix for the XML Schema instance namespace:

```
<xsd:schema xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance">
```

BP1743

Statement:

Use .xml as the file extension for files that contain XML Instance Documents.

Rationale:

By using the .xml extension for XML Instance Documents that are not associated with an application that requires another file extension (e.g., html, xslt):

- Readily identifies the file as containing XML to users
- Associates the XML file with various tools that work with XML Documents (i.e., browsers, parsers, validators, etc.)

Referenced By:

XML Instance Documents

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1743.1]

Are there XML files that do not have the XML file extension or that are associated with specific applications?

Procedure:

Scan the files looking for files that contain XML that are not associated with an application. Examples of files that are associated with applications or services are .wsdl, .html, .htm and .xsl.

Example:

None.

BP1747

Statement:

Use the xsl qualifying prefix for XSLT namespace.

Rationale:

Syntactically there is no reason why the XSLT namespace can not be given any qualifier. However, for readability on the part of humans, using the xsl qualifier is clear, precise, concise and widely accepted.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1747.1]

Does the schema use the xsl prefix for the XSLT namespace?

Procedure:

Look for the use of the XSLT namespace declaration and verify that the prefix is xsl. Make sure there is only one namespace associated with the Transform XSD: http://www.w3.org/1999/XSL/Transform

Example:

The following is an example of using the xsl prefix for the XSL Transform namespace:

```
<xsl:stylesheet
xmlns: xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns: xalan="http://xml.apache.org/xalan"
    xmlns: my-ext="ext1"
    extension-element-prefixes="my-ext">
```

BP1748

Statement:

Separate static content from transformational logic in XSLTs.

Rationale:

Static XML content is content is copied verbatim from a static source, either internally or externally. Internal static content usually is found within the same input stream as the XSLT content. External static content is obtained from a different input stream and often comes from files or from data returned from a service.

Separating the static content from the transform logic facilitates maintenance by reducing the risk of unexpected side effects during the maintenance. In other words, maintenance to the transformational logic is isolated from the content. Content modifications have no affect on the transformation logic.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1748.1]

Is static content imported using the xsl:copy element that selects a document?

Procedure:

Look for the intermixing of static content with the XSLT transform code.

Example:

BP1749

Statement:

Use xsl:include for including XSL transforms.

Rationale:

Xsl:include includes other transforms and assigns the same precedence to the imported nodes as the importing document. This is the preferred method for including entire XSL transforms to allow for composition of multiple transforms into one that is much bigger.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1749.1]

Procedure:

Example:

```
<xsl:include href="Guidance.xsl"/>
```

BP1750

Statement:

Use xsl:import for reusing XSL code.

Rationale:

Since xsl:import includes other XSL code with a lower precedence than the importing document, it is best to just include small snippets of reusable XSL code. Also, xsl:import is inefficient versus xsl:include when dealing with large documents.

Referenced By:

XSLT

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1750.1]

Procedure:

Example:

```
<xsl:import href="Guidance.xsl"/>
```

BP1752

Statement:

Place dynamic element data within a CDATA section.

Rationale:

The content of dynamic data can not be predicted and could contain the XML special reserved characters < and & or the other characters that may cause parse errors; it is best to embed this data within an XML Character Data (CDATA) section that is ignored by parsers.

The following is an example of the use of a CDATA section that contains source code. Since the code could contain the < or & characters and be runtime dependent, a parse error could occur at runtime.
Please refer to the following example:

```
<![CDATA[
Public bool lessThan (a,b)
{ if (a!= null && b!=null a < b ) then
  { return true;
  } // End if
  else
  { return false;
  } // End else
} // End lessThan
]]>
```

Referenced By:

XML Syntax

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1752.1]

Do Element Data sections that are dynamically generated or are provided by external data surround the Element Data within a CDATA section?

Procedure:

Look for areas within XML instance documents or XML schemas that are candidates for dynamic content that can not be expected to be under the control of the XML instance document generator.

Example:

The following is an example of the use of a CDATA block that contains source code. Since the code could contain the < or & characters, a parse error could occur at runtime.
Please refer to the following example:

```
<![CDATA[
Public bool lessThan (a,b)
```

```
{ if (a < b ) then
  { return 1;
  } // End if
else
  { return 0;
  } // End else
} // End lessThan
]]>
```

BP1757

Statement:

Do not ignore namespace prefixes in XPath expressions.

Rationale:

Ignoring namespaces can have undesired consequences. Some namespaces can contain nodes (elements) with the same name that contain different data structures. Consequently, if names bypass the use of the associated namespace, runtime errors can occur when attempts to process nodes of differing types occur.

Referenced By:

XPath

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1757.1]

Do any XPath statements ignore namespaces?

Procedure:

Check for the existence of XPaths similar to the following:

```
//*[local-name()='location']
```

location is a node name defined in two different namespaces. For example, a geographic namespace may define location as latitude and longitude. It may also be defined in the display namespace as a x and y pixel coordinate.

Example:

None.

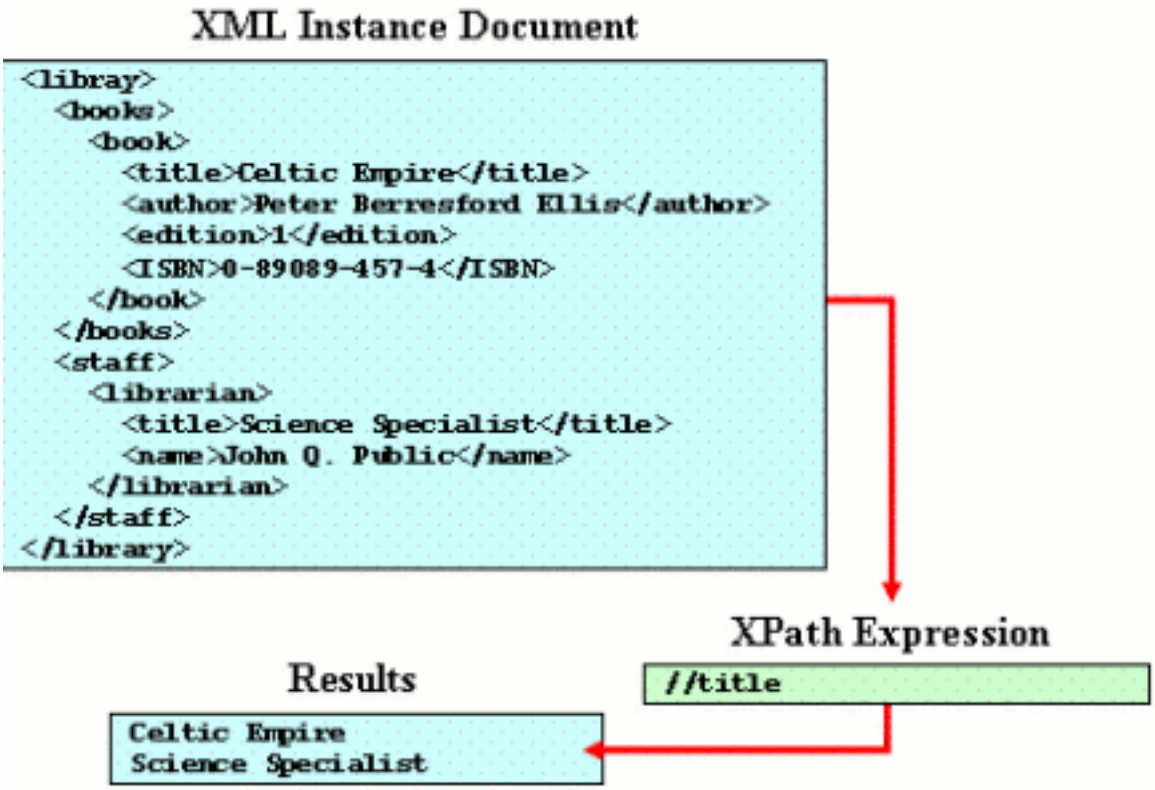
BP1758

Statement:

Make names in descendant expressions unique within an XML document.

Rationale:

The descendant operator, when misused, can have unintended consequences since nodes of the same name could possibly be included in multiple places in the XML Document. The XPath need to be written to eliminates unwanted nodes of the same name from other parts of the document.



In the above example, the <title> element can occur in multiple places within the document. Using the descendent operator '//' with the title element name returns all the titles.

Referenced By:

XPath

Acquisition Phase:

Development

BP1764

Statement:

Make all localizable user interface elements such as text and graphics externally configurable.

Rationale:

Externally configurable user interface elements allow for changing the supported language(s) at deploy-time or run-time without recompilation.

Referenced By:

Designing User Interfaces for Internationalization

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1764.1]

Are all localizable presentation elements such as user interface text and graphics externally configurable?

Procedure:

Check for external configuration files for localizable presentation user interface elements.

Example:

BP1765

Statement:

Declare the encoding type for all user interface content.

Rationale:

Declaring the encoding type allows for an application to determine the encoding type programmatically and make necessary display configuration settings at run-time. Also, for Unicode there are multiple ways to encode a character set.

Referenced By:

Designing User Interfaces for Internationalization

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1765.1]

Do the user interface components (such as HTML pages) declare the encoding type?

Procedure:

Check to see that user interface components declare the encoding type.

Example:

Send the charset parameter in the Content-Type of HTTP header:

```
Content-Type: text/html; charset=utf-8
```

For XML (including XHTML), use the encoding pseudo-attribute in the XML declaration at the start of a document:

For HTML or XHTML served as HTML, use the tag inside :

BP1766

Statement:

Develop user interfaces to accommodate variable syntactic structure for messages.

Rationale:

Different languages form sentence structures in different ways. Composing messages in code from multiple substrings in order to display the messages to the user may cause problems when porting the code to a language that uses a different sentence structure.

Referenced By:

Designing User Interfaces for Internationalization

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1766.1]

Are messages displayed on the user interface constructed in code using multiple substrings?

Procedure:

Check code for messages displayed to the user to see if the messages are composed from multiple substrings.

Example:

BP1767

Statement:

Follow a standard process for human systems integration engineering such as the one defined by the International Organization for Standardization in ISO 13407:1999 on human-centered design processes for interactive systems.

Rationale:

Using a standard well-defined process increased the chance that required steps and procedures are completed during system development and lead to better usability.

Referenced By:

Human-Computer Interaction

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1767.1]

Was a process for human systems integration followed during system development?

Procedure:

Look for documentation stating the human systems integration process.

Example:

BP1768

Statement:

Use design patterns for application navigation.

Rationale:

Using common design patterns for application navigation builds on lessons learned, increases probability of user understand of the navigation pattern, and may result in better performance and a reduction in training.

Referenced By:

Human Factor Considerations for Web-Based User Interfaces

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1768.1]

Does the application navigation follow design patterns?

Procedure:

Identify the design patterns used for application navigation.

Example:

- Use a hub navigation pattern for tasks that consist of multiple independent steps performed in any order
- Use wizard navigation pattern for tasks that consist of multiple interdependent steps that are defined in a predefined order.
- Use a pyramid navigation pattern when it is necessary to navigate to sibling, child, or parent pages while completing tasks.

BP1769

Statement:

Provide wrapper or adapter classes to isolate XML parser implementations.

Rationale:

Referenced By:

Parsing XML

Acquisition Phase:

Development

BP1780

Statement:

Only overload arithmetic operators for objects that are arithmetic in nature.

Rationale:

In languages such as C++, it is possible to extend the intrinsic syntactical structure by defining overloaded operators. Operators that are naturally considered mathematical in nature (i.e., **`add`**, **`subtract`**, **`multiply`**, **`divide`**, etc.) should behave as expected. For example, if the addition operator **`+`** is defined, it should represent the mathematical addition operation.

Referenced By:

C++ Operator Overloading

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1780.1]

Do overloaded mathematical operators perform any mathematical operations?

Procedure:

Review any mathematical operators that have been defined for any classes and ensure that they are mathematical in nature.

Example:

The following is an example of an addition operator:

```
class Imaginary
{ double value_;
  bool imaginary_;
  Imaginary
    ( double value,
      bool imaginary
    )
  { value_ = value;
    imaginary_ = imaginary;
  } // End Imaginary constructor
```

```
  Imaginary operator+

    ( Imaginary leftSideOfOperator )

  { ... // do what needs to be done

  } // End operator+

} // End Imaginary class
```

BP1781

Statement:

Allocate and de-allocate all module objects within the module that contains the objects.

Rationale:

Sutter and Alexandrescu define a module as any cohesive unit of release maintained by a single person or team that is typically compiled with the same compiler, compiler version and compiler switches.

Because the memory allocation and de-allocation can change between these compiler instances, memory leaks and memory corruption can occur. Anytime memory allocation and de-allocation conflicts occur, there is a potential security issue.

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 60.*

Referenced By:

C++ Namespaces and Modules

Acquisition Phase:

Development

BP1782

Statement:

Do not propagate exceptions across module boundaries.

Rationale:

Because the underlying definition of exceptions can vary between instances of a compiler, the resulting executable code could also vary resulting in not being able to properly communicate the exception.

> ***Note:*** *This practice has been adapted from Sutter and Alexandrescu, standard practice 62.*

Referenced By:

C++ Namespaces and Modules

Acquisition Phase:

Development

BP1783

Statement:

Use portable types in a module's interface.

Rationale:

Because the types define the data that flows between modules and each compiler instance can vary these definitions, the types that define this data needs to be uniform in order to ensure proper data transfer.

Note: *This practice has been adapted from Sutter and Alexandrescu, standard practice 63.*

Referenced By:

C++ Namespaces and Modules

Acquisition Phase:

Development

BP1811

Statement:

Isolate all use of vendor specific extensions to the **Data Distribution Service** (DDS).

Rationale:

Vendor specific extensions may be required to perform certain configuration actions, take advantage of features that are in the process of becoming standard (e.g., version 1.3, expected to be adopted by late 2007), or simply use additional capabilities provided by a vendor that would otherwise require significant application work.

Vendor-specific extensions should only be used if there is no standard API from the DDS specification that accomplishes the same function.
One method of isolating vendor-specific extensions is to enclose the code within conditional compile instructions (e.g., **`#ifdef #endif`** for C/C++) such that portability is not compromised.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1811.2]

Does the implementation use wrappers or façade patterns to isolate vendor specific code?

Procedure:

Is vendor specific code contained within a limited number of classes or objects?

Example:

None

2) Test: [BP1811.1]

Does the implementation annotate vendor specific code?

Procedure:

Look for the use of compiler instructions that isolate vendor specific code.

Example:

```
#ifdef DDS_VENDOR_XXXX
…. <vendor specific code
#endif
```

BP1812

Statement:

Use the **RELIABILITY** **Quality of Service** (QoS) kind **BEST_EFFORT** for **Data Distribution Service** (DDS) **Topics** that are written frequently where missing an update is not important because new updates occur soon thereafter.

Rationale:

The use of the **RELIABILITY** QoS kind **BEST_EFFORT** allows the middleware to use a lower-latency, lighter-weight protocol to send data that avoids the need for extraneous Acknowledgement and Heartbeat traffic. This protocol also exploit multicast more efficiently because there is never a need to send any acknowledgments back to the sender. Consequently, this protocol should be preferentially used whenever the nature of the Topic is such that occasionally missing a message has no adverse consequence to the system.

Data that is continually published and represents updates to data-objects or where only the most current value is of interest to the system are prime candidates for **BEST_EFFORT** communication.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Design and Development

Evaluation Criteria:

1) Test: [BP1812.1]

Is the **RELIABILITY** QoS selection properly justified for each Topic? Is **BEST_EFFORT** kind used whenever the nature of the Topic allows it?

Procedure:

Review the system documentation for proper justification of the **RELIABILITY** QoS assigned to each Topic.

Example:

None

BP1813

Statement:

Use the **RELIABILITY** **Quality of Service** (QoS) kind **RELIABLE** for **Data Distribution Service** (DDS) **Topics** written sporadically or where it is important that the current data in the Topic is received reliably.

Rationale:

The **RELIABILITY** QoS kind **RELIABLE** ensures the service will make all necessary attempts to deliver the information. The DDS protocol employs Heartbeats and Acknowledgments to accomplish this task.

Data that is rarely written or which the system requires never to be lost should be published with **RELIABILITY** QoS kind **RELIABLE.**

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1813.1]

Is the **RELIABILITY** QoS selection properly justified for each Topic? Is **RELIABLE** kind used whenever the nature of the Topic requires it?

Procedure:

Review the system documentation for proper justification of the **RELIABILITY** QoS assigned to each Topic.

Example:

None

BP1814

Statement:

Use the **`DEADLINE`** **Quality of Service** (QoS) to for **Data Distribution Service** (DDS) **DataWriters** for which data is published at a constant rate.

Rationale:

The frequency with which a particular data-object is updated may affect the logic of the overall system. For example some radar processing algorithms may have been written under the assumption that each track is updated every five seconds after the radar completes a new sweep.

If the **`DataWriter`** specifies a **`DEADLINE`** QoS, DDS can monitor that each data-object is indeed written at least once per stated period. Furthermore, DDS can propagate the **`DataWriter`** deadline to the **DataReaders** such that they can realize whether their expectation matches what the **`DataWriter`** provides. If the expectation cannot be met the application is notified of an incompatible QoS.

By using this QoS the modules can remain de-coupled, yet provide the essential information required for the integrated system to operate as expected.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1814.1]

Is the **`DEADLINE`** QoS used in all the **`DataWriters`** where it could?

Procedure:

Review the system documentation for proper justification of the **`DEADLINE`** QoS assigned to each **`DataWriter`**.

Example:

BP1815

Statement:

Use the **`DEADLINE`** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **DataReaders** that expect data to be sent to them at a constant rate.

Rationale:

The frequency with which a particular data-object is updated may affect the logic of the overall system. For example some radar processing algorithms may have been written under the assumption that each track is updated every five seconds after the radar completes a new sweep.

If the **`DataReader`** specifies a **`DEADLINE`** QoS then DDS can monitor that an update to each data-object is indeed received at least once per stated period and if not notify the application. Furthermore, DDS can propagate the **`DataReader`** deadline to the **DataWriters** such that they can realize whether they can meet the expectation of the **`DataReader`**. If the expectation cannot be met the application is notified of an incompatible QoS.

By using this QoS the modules can remain decoupled, yet provide the essential information required for the integrated system to operate as expected.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1815.1]

Is the **`DEADLINE`** QoS used in all the **`DataReaders`** where it could?

Procedure:

Review the system documentation for proper justification of the **`DEADLINE`** QoS assigned to each **`DataReader`**.

Example:

BP1816

Statement:

Use the **`LIVELINESS`** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **Topics** where data is not sent sporadically; that is, it is sent with no fixed period.

Rationale:

Some data (e.g., alarms or commands) are sent without a fixed period. In these cases the fact that updates are not received could indicate that there is either no new data, or alternatively that there is a system malfunction and the writer is not able to send the data. The DDS **`LIVELINESS`** QoS allows the application to discern between these two situations.

Setting the **`LIVELINES`** QoS indicates to DDS that in the event that there is no data to send, periodic liveliness messages should be exchanged to notify the **DataReader** that the **DataWriter** is still active, capable of communication, and therefore that if it receives no data then it is in fact because there is none to send. The DDS monitors the **`LIVELINESS`** and informs the application when a **`DataWriter`** loses its ***liveliness*** via the proper status message dispatched to the Listener.

Proper settings of the **`LIVELINESS`** QoS is also required to receive proper **`InstanceState`** information with the received Samples as well as to manage **`OWNERSHIP`** in the presence of failures.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1816.1]

Are all **`DataWriters`** or **`DataReaders`** that do not set a **`DEADLINE`** setting a **`LIVELINESS`**?

Procedure:

Check the QoS used to create **`DataReaders`** and **`DataWriters`** and ensure that if the **`DEADLINE`** QoS is not set, then the **`LIVELINESS`** QoS is set to a non-infinite value

Example:

BP1817

Statement:

Use the **MANUAL_BY_TOPIC** setting of the **LIVELINESS** **Quality of Service** (QoS) for **Data Distribution Service** (DDS) **Topics** where the presence and health of the **DataWriter** is critical to the proper operation of the system.

Rationale:

Certain Topics are monitoring functions so critical to the health of the system that reliance on the health of the process that writes the Topic does not offer sufficient assurance that the application is performing the proper monitoring functions. In these situations the **MANUAL_BY_TOPIC** setting of the **LIVELINESS** QoS requires the **DataWriter** to either write the data at least once per liveliness period or invoke the **DataWriterasset_liveliness()** operation to indicate proper functioning.

The **MANUAL_BY_TOPIC** setting of the **LIVELINESS** QoS can be thought of as the distributed system equivalent to the mechanical dead man's switches used to monitor that the operator of a system (e.g., a train locomotive) is still present and able to function.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1817.1]

Are all critical **DataWriters** either setting a deadline or using a **LIVELINESS** set to **MANUAL_BY_TOPIC**?

Procedure:

Check the QoS used to create **DataReaders** and **DataWriters** and ensure that if the **DEADLINE** QoS is not set, then the **LIVELINESS** QoS is set to **MANUAL_BY_TOPIC** and has a non-infinite value.

Example:

BP1818

Statement:

Use the **HISTORY** **Quality of Service** (QoS) kind **KEEP_LAST** for **Data Distribution Service** (DDS) **Topics** that represent system state, in that new data-values replace the old values for each Keyed data-object.

Rationale:

Some Topics represent system state. The readers of the Topic need only know the most current value (or last set of N values) of each data-object published under the Topic. An example of this may be a Topic representing the reading of different temperature sensors. Applications only care to read the most recent value of each sensor. The same may be said of a Topic representing the expected arrival times of aircraft at a given airport.

The **HISTORY** QoS setting of **KEEP_LAST** indicates to the middleware that it should not attempt to store or propagate old values of data objects; instead, only the most recent value(s) are of interest. This allows DDS to conserve system resources (memory) as well as to save the bandwidth required to send information that is no longer relevant. Reader applications also benefit as they do not waste time reacting to data values that are no longer current.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1818.1]

Is the **HISTORY** QoS properly sent on all Topics?

Procedure:

Check the QoS used to create **DataReaders** and **DataWriters** and check how the **HISTORY** QoS is set. Ensure that a kind **KEEP_LAST** is used whenever the Topic represents system state.

Example:

BP1819

Statement:

Use the **`HISTORY`** **Quality of Service** (QoS) kind **`KEEP_ALL`** for **Data Distribution Service** (DDS) **Topics** that represent events or commands where all values written should be delivered to the readers (i.e., new values do not replace old values).

Rationale:

Some Topics represent events, commands, or messages in that new data written never replaces previously-written values, rather they should all be delivered to the **DataReader**.

The **`HISTORY`** QoS setting of **`KEEP_ALL`** indicates to the middleware that it should not replace old values with new values on the topic. Subject to other QoS (such as filters, ownership, lifespan) they should all be delivered to the **`DataReaders`**.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1819.1]

Is the **`HISTORY`** QoS properly sent on all Topics?

Procedure:

Check the QoS used to create **`DataReaders`** and **DataWriters** and check how the **`HISTORY`** QoS is set. Ensure that a kind **`KEEP_ALL`** is used whenever the Topic represents 'events', commands or messages.

Example:

BP1820

Statement:

Use **`TIME_BASED_FILTER`** **Quality of Service** (QoS) to protect **DataReaders**that cannot handle all the traffic that could be written by the writers on that **Data Distribution Service** (DDS)**Topic** and just need periodic updates on the most current data-values.

Rationale:

The **`TIME_BASED_FILTER`** QoS allows a **`DataReader`** to specify that it is interested only in (potentially) a subset of the values of the data. The filter states that the **`DataReader`** does not want to receive more than one value each minimum_separation, regardless of how fast the changes occur. The default setting is minimum_separation=0 indicating that the **`DataReader`** is potentially interested in all values.

In heterogeneous systems, it is common that some subsystems either cannot handle or do not choose to handle all the information available on a Topic. For example a high-level display at an airport control tower may not need to update the location of aircraft more often than each second as the human operators looking at the display would not be able to take advantage of faster refreshes. Nevertheless, the data is published at much higher rate to allow for algorithmic processing on other subsystems.

By setting the **`TIME_BASED_FILTER`** properly an application that has a well defined maximum refresh rate can protect itself from system reconfigurations which may result in a Topic being published faster than originally anticipated.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1820.1]

Is the **`TIME_BASED_FILTER`** QoS properly sent on all **`DataReaders`**?

Procedure:

Check the QoS used to create **`DataReaders`** and check whether the **`TIME_BASED_FILTER`** QoS is set. Ensure it is set to a proper non-zero minimum_separation whenever the application can be in a system where it is not expected to handle all the updates on the Topic.

Example:

BP1821

Statement:

Use the **Data Distribution Service** (DDS) **`LIFESPAN`** **Quality of Service** (QoS) to indicate that data is only valid for a finite time period and stale data is discarded after a certain expiration time elapses.

Rationale:

Some **Topics** represent data with a natural expiration. For example the location of an aircraft during flight becomes less relevant as the information ages and may not have any tactical value after a certain time elapses.

The setting of the **`LIFESPAN`** QoS indicates to DDS the maximum time duration during which the information is relevant. After this time elapses, DDS is no longer required to maintain the information or provide it to the **DataReaders**. Proper setting of this QoS can therefore save resources and bandwidth as well as save **`DataReaders`** from being notified of information that is no longer relevant.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1821.1]

Is the **`LIFESPAN`** QoS properly sent on all Topics?

Procedure:

Check the QoS used to create **DataWriters** and check whether the **`LIFESPAN`** QoS is set. Ensure it is set to a proper non-infinite duration whenever appropriate.

Example:

BP1822

Statement:

Use the **`PARTITION`** **Quality of Service** (QoS) to limit the scope of the data written/read on a **Data Distribution Service** (DDS) **Topic** to only the writer/readers that have a common partition.

Rationale:

The **`PARTITION`** QoS is used to introduce logical partitions within a Topic. A **DataWriter** only communicates with a **DataReader** if (in addition to matching the Topic and having compatible QoS) they share a common partition

The **`PARTITION`** QoS is set on the **Publisher** and **Subscriber** and affects all the **`DataWriters`** in the Publisher and **`DataReaders`** on the Subscriber.

The **`PARTITION`** QoS can be used to introduce a logical scope and the fact that it is adjustable at run-time makes it possible to perform system reconfigurations. For example, a **`DataReader`** could be temporarily isolated from the rest of the system by switching its Partition to something that nobody matches. Similarly a **`DataWriter`** and **`DataReader`** could be reconfigured to have an "isolated session" by switching to a partition that nobody else uses.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1822.1]

Is the **`PARTITION`** QoS used to simplify application logic where appropriates?

Procedure:

Check the QoS used to create Publisher and Subscriber and check whether the **`PARTITION`** QoS is used. Verify that the application does not use some other non-standard way to implement a use-case that could be supported using the **`PARTITION`** QoS.

Example:

BP1823

Statement:

Use the **Data Distribution Service** (DDS) **`RESOURCES_LIMITS`** **Quality of Service** (QoS) in platforms with limited memory or in real-time systems to properly configure the resources that will be utilized and avoid exhaustion of system resources at run-time.

Rationale:

The **`RESOURCE_LIMITS`** QoS on the **DataWriter** and **DataReader** specifies the resources that DDS can consume in order to meet the requested QoS.

While these limits can be left to their default "auto-grow" settings proper configuration of these limits is important in any system that has limited resources and is expected to operate reliably for long time spans. By setting the limits the developer can balance the resources consumed for each topic and protect the system against a mis-configuration when a **Topic** that produces too much data exhausts the resources needed to manage other Topics. This is especially important if other QoS do not limit the amount of data that the system would need to store (e.g. if **`HISTORY`** is set to **`KEEP_ALL`** and **`LIFESPAN`** is set to infinite).

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1823.1]

Is the **`RESOURCE_LIMITS`** QoS set on the **`DataWriter`** and **`DataReader`**?

Procedure:

Check the QoS used to create **`DataWriters`** and **`DataReaders`** and check whether the **`RESOURCE_LIMITS`** are set to some finite limits. Ensure that any **`DataWriters`** and **`DataReaders`** that have if **`HISTORY`** kind **`KEEP_ALL`** and **`LIFESPAN`** duration set to infinite use the **`RESOURCE_LIMITS`** to control the maximum resource utilization.

Example:

BP1824

Statement:

Use the **`USER_DATA`** **Quality of Service** (QoS) to communicate metadata on the **DomainParticipant** that may be used to authenticate the application trying to join the Data **Distribution Service** (DDS) **Domain**.

Rationale:

In many cases the application needs to send additional information that describes the **`DomainParticipant`** to other participants in the DDS Domain. This information can be used to authenticate the participant or to meet any other application-specific need.

The **`USER_DATA`** QoS on the **`DomainParticipant`** allows the application to store un-interpreted bytes that will be propagated via the DDS built-in discovery mechanism and will be accessible to the other **`DomainParticipants`** on the system.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1824.1]

Is the **`USER_DATA`** QoS set on the **`DomainParticipant`**?

Procedure:

Check the creation of the **`DomainParticipant`** and determine whether the **`USER_DATA`** QoS is used. Ensure that the application does not use another non-standard way to accomplish the same function.

Example:

BP1825

Statement:

Use the **`ignore_participant`** operation on the **DomainParticipant** to deny access to another DomainParticipant trying to join a **Data Distribution Service** (DDS) **Domain**.

Rationale:

The **`ignore_participant`** operation can be used by a **`DomainParticipant`** to prevent another **`DomainParticipant`** from communicating with the first participant. In combination with the **`USER_DATA`** QoS on the participant this mechanism can be used to authenticate **`DomainParticipants`**.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1825.1]

Is the **`ignore_participant`** operation used whenever there is a requirement to prevent arbitrary participants from accessing the information the first participant publishes or subscribes?

Procedure:

Check the code for any occurrences of the **`ignore_participant`** operation.
Ensure that the application does not use another non-standard way to accomplish the same function.

Example:

BP1826

Statement:

Use the **USER_DATA** **Quality of Service** (QoS) on the **DataWriters** and **DataReaders** to communicate metadata that may provide application-specific information of the entity writing/reading data in a **Data Distribution Service** (DDS) **Domain**.

Rationale:

In many cases the application needs to send additional information that describes the **DataWriter** or the DataReader to other entities in the DDS Domain. This information can be used to authenticate the **DataWriter**/Reader or to meet any other application-specific need.

The **USER**_DATA QoS on the **DataWriter** and the **DataReader** allows the application to store un-interpreted bytes that will be propagated via DDS's built-in discovery mechanism and will be accessible to the other **DataWriters** and **DataReaders** on the system.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1826.1]

Is the **USER_DATA** QoS set on the **DataWriter** and **DataReader**?

Procedure:

Check the creation of the **DataWriter** and **DataReader** and determine whether the **USER_DATA** QoS is used. Ensure that the application does not use another non-standard way to accomplish the same function.

Example:

BP1827

Statement:

Use the **`ignore_publication`** and **`ignore_subscription`** on the **DomainParticipant** to deny access to a **Data Distribution Service** (DDS) **Topic** by a specific **DataWriter** or **DataReader**.

Rationale:

The ignore_publication and ignore_subscription operation can be used by a **`DomainParticipant`** to prevent a **`DataWriter`** or **`DataReader`** from communicating with the entities in the participant. In combination with the **`USER_DATA`** QoS on the **`DataWriter`** and **`DataReader`** this mechanism can be used to check that the **`DataWriter`** and **`DataReader`** have the proper access control to the Topic.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

Evaluation Criteria:

1) Test: [BP1827.1]

Are the **`ignore_publication`** and **`ignore_subscription`** operation used whenever there is a requirement to prevent arbitrary **`DataWriters`** or **`DataReaders`** from accessing the information on a Topic?

Procedure:

Check the code for any occurrences of the **`ignore_publication`** and **`ignore_subscription`** operation. Ensure that the application does not use another non-standard way to accomplish the same function.

Example:

BP1828

Statement:

Use the **Data Distribution Service** (DDS) **`OWNERSHIP`** **Quality of Service** (QoS) kind set to **`SHARED`** when each unique data-object within a DDS **Topic** to which multiple **DataWriters** can write.

Rationale:

A primary intent of DDS is to support a loosely coupled publish and subscribe paradigm where the publishing is isolated from subscribing through autonomous topics. As a result, an implementation that requires a single data publisher currently may evolve to require multiple data publishers in the future. By using a **`OWNERSHIP`** QoS kind set to **`SHARED`** and allowing the DDS infrastructure to connect the **publisher** and the **subscriber** together, the implementation may be extended to another DDS profile without having to modify the original source code.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

BP1829

Statement:

Use the **Data Distribution Service** (DDS) **`OWNERSHIP`** **Quality of Service** (QoS) kind set to **`EXCLUSIVE`** when multiple **DataWriters** cannot write each unique data-object within a DDS **Topic** simultaneously.

Rationale:

DDS easily supports multiple **publishers** adding data to the same topic without impacting the **subscribers**. Using the DDS **`OWNERSHIP`** QoS kind set to **`EXCLUSIVE`** places the entire burden off supporting the multiple publishers on the DDS implementation rather than the publisher or subscriber code. This results in an increase of modularity, portability and the maintainability.

Referenced By:

DDS Quality of Service

Acquisition Phase:

Development

BP1830

Statement:

Use the **Data Distribution Service** (DDS) Content Profile to tailor subscription message data.

Rationale:

The DDS Content Profile allows for the **subscribers** to select and refine the data that is retrieved from a **Topic**. This tailoring code is part of the DDS infrastructure and is well tested and reliable. Not using the DDS Content Profile and using code within the subscriber increases the complexity of the subscriber and causes tight coupling between the subscriber code and the Topic.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

BP1831

Statement:

Use the **Data Distribution Service** (DDS) Persistence Profile to ensure durable data delivery.

Rationale:

The DDS Persistence Profile allows for data persistence within a **Topic** independent of hardware platform and operating system (OS) and to retrieve the data using the standard **Structured Query Language** (SQL). As a result, the publisher, subscriber and the topic remain loosely coupled from each other as well as the hardware platform or the OS.

Referenced By:

Decoupling Using DDS and Publish-Subscribe

Acquisition Phase:

Development

BP1832

Statement:

Handle all **Data Distribution Service** (DDS) **Data Local Reconstruction Layer** (DLRL) Exceptions.

Rationale:

The DLRL API may raise Exceptions under certain conditions. The following is an extensive list of all possible Exceptions and the conditions in which they will be raised:

`DCPSError`	If an unexpected error occurred in the DCPS
`BadHomeDefinition`	If a registered **`ObjectHome`** has dependencies to other, unregistered **`ObjectHomes`**.
`NotFound`	If a reference is encountered to an object that has not (yet) been received by the **DCPS**.
`AlreadyExisting`	If a new object is created using an identify that is already in use by another object.
`AlreadyDeleted`	If an operation is invoked on an object that has already been deleted
`PreconditionNotMet`	If a precondition for this operation has not (yet) been met.
`NoSuchElement`	If an attempt is made to retrieve a non-existing element from a Collection.
`SQLError`	If an **SQL** expression has bad syntax, addresses non-existing fields or is not consistent with its parameters.

Note: *DLRL, a recent addition to the DDS specification is particularly rich; implementations using this upper level profile of the specification are still emerging.*

Referenced By:

DDS Data Local Reconstruction Layer (DLRL)

Acquisition Phase:

Development

BP1833

Statement:

Use the **Data Distribution Service** (DDS) Object Model Profile for accessing message data as objects.

Rationale:

The DDS **Data Local Reconstruction Layer** (DLRL) is intended to provide an abstraction layer between the actual underlying data and the higher level object level concepts used in applications. The Object Model Profile defines how applications interact with the abstract object layer. Applications that are bound directly to the actual underlying data are tightly coupled to the layer and are subject to its evolutionary changes.

Note: *DLRL, a recent addition to the DDS specification is particularly rich; implementations using this upper level profile of the specification are still emerging.*

Referenced By:

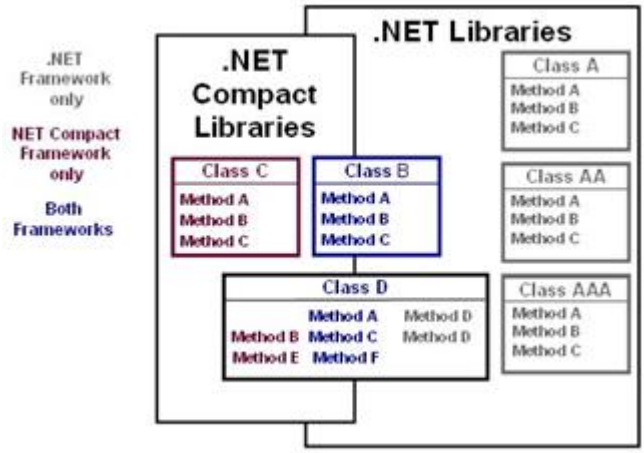
DDS Data Local Reconstruction Layer (DLRL)

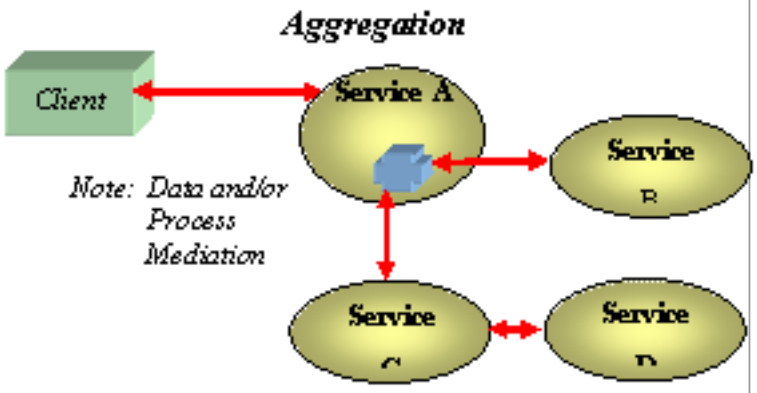
Acquisition Phase:

Development

Glossary

.NET		To address the confusing maze of computer languages, libraries, tools, and toolkits that were necessary for creating multi-tier applications, Microsoft developed the .NET Framework and integrated it into Microsoft Windows as a component. It supports building and running multi-tier and service-oriented architectures, including Web services and client and server applications. It simplifies the process of designing, developing, and testing software, allowing individual developers to focus on core, application-specific code.
.NET Compact Framework		The Microsoft .NET Compact Framework is a streamlined version of the .NET Framework that is designed to run on mobile devices with limited memory, resources, and battery power, including smart devices like Personal Digital Assistants (PDAs), mobile phones, and set-top boxes. The .NET Compact Framework includes the base class libraries from the full .NET Framework and a few libraries designed specifically for mobile devices such as Windows CE InputPanel. Developers can create applications for the .NET Compact Framework in Visual Studio .NET 2003, using Microsoft Visual C# .NET or Microsoft Visual Basic .NET. The resulting applications are designed to run on a special, mobile-device, high performance JIT compiler. To run .NET Compact Framework applications, the platform must support the Microsoft .NET Compact Framework runtime. This includes Windows CE.NET, Windows CE 4.1, Microsoft Pocket PC, Microsoft Pocket PC 2002, or Smartphone 2003. **Architecture** The .NET Compact Framework is a subset of the .NET Libraries. It includes only those aspects of the .NET Library that are essential for the functionality. Several namespaces and classes are used exclusively in the .NET Library. Other namespaces, classes and methods are in both the .NET Library and the .Net Compact Library, and there are namespaces and classes that are exclusive to the .Net Compact Library.

		
Accredited Standards Committee Standard X12	ANSI ASC X12	Numbered set of commercial **EDI** transactions defined by the American National Standards Institute's Accredited Standards Committee X12. Uniform rules for the interchange of business documents defined for cross industry EDI use.
Active Server Page	ASP	A script that is executed by Microsoft Internet Information Services. The output is returned to the user as **HTML**. Typically, an ASP script generates a customized Web page on the fly before sending it to the user. ASPs are specific to Microsoft, only run on **IIS** or **PWS**, can contain HTML, **JScript**, and **VBScript**, and can access **COM** components.
ActiveX		An ActiveX control is similar to a Java **applet**. However, ActiveX controls have full access to the Windows OS. This gives them much more power than Java applets, plus a risk that the applet may damage software or data on your machine. To control this risk, Microsoft developed a registration system so that browsers can identify and authenticate an ActiveX control before downloading it. Another difference between Java applets and ActiveX controls is that Java applets can be written to run on all platforms, whereas ActiveX controls are currently limited to Windows environments.
Adapter		An intermediary that translates between incompatible components interfaces, allowing them to communicate.
Aggregation		When information is derived from multiple sources a mediator service may aggregate the data and thus make many services appear to be one.

		 Note: See **Mediation**.
American National Standards Institute	ANSI	Administrator and coordinator of the United States private-sector voluntary standardization system. ANSI facilitates the development of American National Standards (ANS) by accrediting the procedures of standards-developing organizations. The Institute remains a private, nonprofit membership organization supported by a diverse constituency of private and public sector organizations. (Source: http://web.ansi.org/)
American National Standards Institute Standard for Electronic Data Interchange	ANSI X12	Numbered set of commercial electronic data interchange (**EDI**) transactions defined by the American National Standards Institute's Accredited Standards Committee X12. Uniform rules for the interchange of business documents defined for cross industry EDI use.

American Standard Code for Information Interchange	ASCII	ASCII is a character set and a character encoding based on the Roman alphabet as used in modern English (see English alphabet). ASCII codes represent text in computers, in other communications equipment, and in control devices that work with text. Most often, nowadays, character encoding has an ASCII-like base.
		
		ASCII defines the following printable characters, presented here in numerical order of their ASCII value:
		
		`!"#$%'()*+,-./0123456789:; ?`
		`@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`
		`` `abcdefghijklmnopqrstuvwxyz{|}~( ``
		
		(Source: http://en.wikipedia.org/wiki/ASCII)

Apache Ant		A Java-based build tool that automates the build process using XML descriptor files to capture the build process.
Applet		A J2EE component that typically executes in a Web browser. Applets can also execute in a variety of other applications or devices that support the applet programming model. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Application Environment Profile	AEP	The AEP describes the exact functionality supported by the Operating Environment of the **SCA** specification.

Application Programming Interface	API	A special type of interface that specifies the calling conventions with which one component may access the resources and services provided by another component. APIs are defined by sets of procedures or function-invocation specifications. An API is a special case of an interface.
Application Server		A platform for developing and deploying multi-tier distributed enterprise applications.
Assistant Secretary of Defense for Networks and Information Integration	ASD (NII)	(Source: http://www.dod.mil/nii/)
Asymmetric Key Cryptography		Synonym for **Public Key Cryptography**.
Attribute		A distinct characteristic of an object. Real-world object attributes are often specified in terms of their physical traits, such as size, shape, weight, and color. Cyberspace object attributes might describe size, type of encoding, and network address. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Authentication		The process that verifies the identity of a user, device, or other entity in a computer system, usually as a prerequisite to allowing access to resources in a system. The Java servlet specification requires three types of authentication (basic, form-based, and mutual) and supports digest authentication. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Basic Object Adapter	BOA	The Basic Object Adapter was an early (v1) CORBA component; see the **Portable Object Adapter** (**POA**).
Binary XML		
Business Logic		The code that implements the functionality of an application. In the Enterprise JavaBeans architecture, this logic is implemented by the methods of an enterprise bean. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Business Process Execution Language	BPEL	BPEL is emerging as the standard for assembling a set of discrete services into an end-to-end process flow, radically reducing the cost and complexity of process integration initiatives. (Source: http://www.oracle.com/technology/products/ias/bpel/index.html)
Business Process Execution Language for Web Services	BPEL4WS	
Canonicalization		The process of converting data that has more than one possible representation into a "standard" canonical representation. This can be done to compare different representations for equivalence, to count the number of distinct data structure , to improve the efficiency of various algorithms by eliminating repeated calculations, or to make it

		possible to impose a meaningful sorting order. (Source: http://en.wikipedia.org/wiki/Canonicalization) When referring to XML, the process of converting an XML document to a form that is consistent to all parties. Used when signing documents and interpreting signatures. Any XML document is part of a set of XML documents that are logically equivalent within an application context. Generally, if two documents have the same canonical form, then the two documents are logically equivalent within the given application context. Methods exist for generating a physical representation, the canonical form, of an XML document that accounts for the permissible changes. Note that two documents may have differing canonical forms yet still be equivalent in a given context based on application-specific equivalence rules for which no generalized XML specification could account.
Cascading Style Sheet	CSS	Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing) to Web documents. (Source: http://www.w3.org/Style/CSS/)
Certificate	CERT	A certificate which uses a digital signature to bind together a public key with an identity information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual. (Source: http://en.wikipedia.org/wiki/Certificate_%28cryptography%29)
Certificate Authority	CA	Certification Authority (CA) is an trusted organization which issues digital public key certificates for use by other parties. It is an example of a trusted third party. CAs are characteristic of many public key infrastructure (PKI) schemes. (Source: http://en.wikipedia.org/wiki/Certificate_authority)
Certificate Revocation List	CRL	A list of certificates (more accurately, their serial numbers) which have been revoked, are no longer valid, and should not be relied upon by any system user. (Source: http://en.wikipedia.org/wiki/Certificate_Revocation_List)
Check Constraint		A constraint based on a user-defined condition - generally documented in a database domain - that has to evaluate to true for the contents of a data base column to be valid.
Client		A system entity that accesses a Web service. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Client-Certificate Authentication		An authentication mechanism that uses HTTP over SSL, in which the server and (optionally) the client authenticate each other with a public key certificate that conforms to a standard that is defined by X.509 Public Key Infrastructure. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Collaboration		Portal members can communicate synchronously through chat or messaging, or asynchronously through threaded discussion, blogs, and email digests (forums).

Command and Control	C2	(DoD) The exercise of authority and direction by a properly designated commander over assigned and attached forces in the accomplishment of the mission. Command and control functions are performed through an arrangement of personnel, equipment, communications, facilities, and procedures employed by a commander in planning, directing, coordinating, and controlling forces and operations in the accomplishment of the mission. (Source: http://www.dtic.mil/doctrine/jel/doddict/data/c/01093.htm)
Command and Control Information Exchange Data Model	C2IEDM	A data model that is managed by the Multilateral Interoperability Programme (MIP). It originated with experts from various NATO partners and from the Partnership-for-Peace nations. This data model is in the process of being submitted to OMG for consideration as the standard for information exchange. It falls under the shared operational picture exchange service. (Source: http://www.mip-site.org/MIP_DMWG.htm)
Commercial Off-The-Shelf	COTS	A term for systems that are manufactured commercially, and may be tailored for specific uses. (Source: http://en.wikipedia.org/wiki/Commercial_off-the-shelf)
Common Business Oriented Language	COBOL	COBOL is a third-generation programming language. Its name is an acronym, for COmmon Business Oriented Language, defining its primary domain in business, finance, and administrative systems for companies and governments. (Source: http://en.wikipedia.org/wiki/COBOL)
Common Gateway Interface Script	CGI Script	CGI is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real time, so it can output dynamic information.
Common Language Runtime	CLR	CLR, at the very core of the **.NET** Framework, encapsulates all the services used from the operating system by compilers of higher level languages such as Visual Basic .NET, Visual C++ .NET, Visual J# .NET and Visual C# .NET. The higher level languages ultimately are translated into native code that directly accesses the CLR.
Common Object Request Broker Architecture	CORBA	CORBA "wraps" code written in another language into a bundle containing additional information on the capabilities of the code inside, and explaining how to call it. The resulting wrapped objects can then be called from other programs (or CORBA objects) over the network. The CORBA specification defines APIs, communication protocol, and object/service information models to enable heterogeneous applications written in various languages running on various platforms to interoperate. (Source: http://en.wikipedia.org/wiki/CORBA)
Community of Interest	COI	A COI is a collaborative group of users that must exchange information in pursuit of its shared goals, interests, missions, or business processes nd therefore must have shared vocabulary for the information it exchanges. (Source: DoDD

		8320.02, 2 December 2004, *Data Sharing in a Net-Centric Department of Defense*)
Community of Interest Service		A service that may be offered to the enterprise, but is owned and operated by a **Community of Interest** to provide or support a well-defined set of mission functions and associated information.
Compiler		A computer program that translates programs expressed in a high-order language into their machine language equivalent. (Source: IEEE Std 610.12-1990)
Complex Semi-Structured Data		Complex Semi-Structured Data has partial metadata. It includes data defined in COBOL copybooks and **Electronic Data Interchange standards ANSI X.12** and Health Level 7 (HL7). Semi-structured data can be as complex or more so as any Complex Structured data. It can map into or be XML. It may also be missing some metadata or an XSD.
Complex Structured Data		Complex Structured Data has well-defined metadata. It includes data represented in XML documents with deeply hierarchical and recursive structures. Complex data can be represented in a complex data structure or can be mapped into a relational or flat structure with additional metadata provided to represent the complex relationships. Although complex structured data is generically a property of object oriented databases, the Complex Data Structures can be filled from any source.
Complex Unstructured Data		Complex Unstructured Data has little or no metadata. It includes data in binary files, spreadsheets, documents, and print streams.
Component		One of the parts that make up a system. A component may be hardware or software and may be subdivided into other components. Note the terms ***module***, ***component***, and ***unit*** are often used interchangeably or defined to be sub-elements of one another in different ways depending on the context. The relationship of these terms is not yet standardized. (Source: IEEE Std 610.12-1990) ***Note:*** *See **system component** and **software component**.*
Component-Based Software		Mission applications that are architected as components integrated within a component framework.
Component Object Model	COM	A Microsoft software architecture for building component-based applications. COM objects are discrete components, each with a unique identity, which expose interfaces that allow applications and other components to access their features. COM objects are more versatile than Win32 DLLs because they are completely language-independent, have built-in inter-process communications capability, and easily fit into an object-oriented program design. COM was first released in 1993 with OLE2, largely to replace the inter-process communication mechanism DDE used by the initial release of OLE. **ActiveX** is based on COM.

Conceptual Model		Captures the concepts of the relational database and can help enforce the first three normalization rules.
Condition		A variable of the operational environment or situation in which a unit, system, or individual is expected to operate that may affect performance. A **DDS** Condition is attached to a **WaitSet** and indicates which condition the application is waiting for asynchronously: **`StatusCondition`**, **`ReadCondition`** or **`QueryCondition`**.
Configuration Control Board	CCB	Also Change Control Board. Duties include reviewing change requests, making decisions, and communicating decisions made to affected groups and individuals. Represents the interests of program and project management by ensuring that a structured process is used to consider proposed changes and incorporate them into a specified release of a product.
Consumer		A system entity invoking producers in a manner conforming to a specification. For example, a portal aggregating content from portlets accessed using the **WSRP** protocol is a type of consumer. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Container		A standard extension mechanism for containers that provides connectivity to enterprise information systems. A connector is specific to an enterprise information system. It consists of a resource adapter and application development tools for enterprise information system connectivity. The resource adapter is plugged in to a container through its support for system-level contracts defined in the Connector architecture. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Core Enterprise Services	CES	Ubiquitous, common solution **services** that provide capabilities essential to the operation of the enterprise. Generic information services that apply to any **COI**, provide the basic ability to search the **enterprise** for desired information, and then establish a connection to the desired service. (Source: http://www.defenselink.mil/nii/org/cio/doc/GIG_ES_Core_Enterprise_Services_Strategy_V1-1a.pdf)
Customized Delivery		Smart push-and-pull of data reduces overload and provides the requested data to operators when they need it. Tailored discovery, publish, and subscribe capabilities allow operators to register for specific data and services in specific timeframes.
Data		Unprocessed information; information without context.
Data Architect		A Data Architect is a job title associated with a person within an organization responsible for making sure the organization's strategic goals are optimized through the use of enterprise data standards. This frequently involves creating and maintaining a centralized registry of **metadata**.

		Data Architecture includes topics such as metadata management, business semantics, data modeling and metadata workflow management. A Data Architect's job frequently includes the set up a **metadata registry** to allow domain-specific stakeholders to maintain their own **data elements**.
Data Asset		Any entity that is composed of data. For example, a database is a data asset that contains data records (e.g., system or application output files, databases, documents, or Web pages). The term data asset also refers to services that provide access to data. For example, a service that returns individual records from a database is considered a data asset since it deals mainly in the function of providing data. Similarly, a Web site that returns data in response to specific queries (e.g., www.defenselink.mil) is considered a data asset. (Source: DoD Net-Centric Data Strategy, 9 May 2003)
Database Data		Data stored in database columns in database tables in a relational database. The set of data records which a relational database is populated. Generally understood to refer to application data and not metadata.
Database Management System	DBMS	A system, usually automated and computerized, for managing any collection of compatible, and ideally normalized, data. (Source: http://en.wikipedia.org/wiki/DBMS)
Data Categorization		
Data-Centric		An approach for the design and implementation of systems, applications, services or software that emphasis the data rather than the operations. It implies that the data is physically separated from the code and consequently can be maintained independently (loose coupling between code and data).
Data-Centric Publish-Subscribe	DCPS	The Data-Centric Publish-Subscribe is a lower level layer of the **DDS** infrastructure that is targeted towards the efficient delivery of the proper information to the proper recipients.
Data Dictionary		A data dictionary is set of metadata that contains definitions and representations of **data elements**. Within the context of a DBMS, a data dictionary is a read-only set of tables and views. The data dictionary may be considered a database in its own right.
Data Distribution Service for Real-Time Systems	DDS	DDS is a recently-adopted OMG standard that is the first open international middleware standard directly addressing publish-subscribe communications for real-time and embedded systems. DDS introduces a virtual Global Data Space where applications can share information by simply reading and writing data-objects addressed by means of an application-defined name (Topic) and a key. DDS features fine and extensive control of QoS parameters, including reliability, bandwidth, delivery deadlines, and resource limits. DDS also supports the construction of local object models on top of the Global Data Space. (Source: OMG Data Distribution Portal, http://portals.omg.org/dds)

Data Element		A data element is an atomic unit of data that has the following: • an identification such as a data element name • a clear data element definition • one or more representation terms • optional enumerated values
Data Element Gallery		The Data Element Gallery is an important component of the Metadata Registry and Clearinghouse. The Data Element Gallery provides its users with access to **data elements** that are commonly used by the Department of Defense such as country codes and U.S. state codes. Users may search the registry, compare data elements, and download an Access database containing the available elements. See the DoD Metadata Registry, http://metadata.dod.mil.
Data Exposure		The steps necessary to set up the metadata infrastructure associated with a net-centric data strategy.
Data Integrity		A measure of the consistency and accuracy of computer data. Integrity can be threatened by hardware problems, power outages, and disk crashes, but most often is threatened by application software or viruses. In a database program, data integrity can be threatened if two users are allowed to update the same item or record at the same time. Record or File Locking, whereby only a single user is allowed access to a given record at any one point in time is one method of ensuring data integrity. (Source: http://www.courts.state.ny.us/ad4/lib/gloss.html#D)
Data Local Reconstruction Layer	DLRL	The Data Local Reconstruction Layer is an optional part of the **DDS** specification that provides a higher level layer allowing for a simpler integration of the DDS into the application layer.
Data Modeling	DM	Modeling is an essential step in understanding the data that will comprise a system. The end products of data modeling can be XML schemas or RDBMS schema definitions. Many COIs create their own data models, such as **C2IEDM** for the **C2** community.
Data Publishing		The steps necessary to make data available within the net-centric data strategy infrastructure.
Data Type		A data type is a constraint placed upon the interpretation of data in a type system in computer programming. Common types of data in programming languages include primitive types (such as integers, floating point numbers or characters), tuples, records, algebraic data types, abstract data types, reference types, classes and function types. A data type describes representation, interpretation and structure of values manipulated by algorithms or objects stored in computer memory or other storage device. The type system uses data type information to check correctness of computer programs that access or manipulate the data. (Source: http://en.wikipedia.org/wiki/Data_type)

DDS DataReader		The **DDS** **`DataReader`** acts as a typed (i.e., dedicated to only one application data type) accessor to a subscriber. The **`DataReader`** class allows the application to declare the data it wishes to receive (i.e., make a subscription) and access the data received by the attached **Subscriber**.
DDS DataWriter		A **DDS** **`DataWriter`** acts as a typed (i.e., dedicated to only one application data type) accessor to a publisher. The **`DataWriter`** class allows the application to set the value of the data to be published under a given **Topic**.
DDS DomainParticipant		A **DDS** domain participant represents the local membership of the computer process in a domain. A **domain** is a distributed concept that links all the computer processes able to communicate with each other. It represents a communication plane; only the **publishers** and the **subscribers** attached to the same domain may interact. A computer process can run on the behalf of some user or application.
DDS Global Data Space		Underlying any **data-centric** publish subscribe system is a data model. In **DDS**, this model defines the global data space and specifies how **Publishers** and **Subscribers** refer to portions of this space. (See DDS **Domain**)
DDS Listener		A **DDS** **`Listener`** is used to provide a callback for synchronous access. Listeners provide a generic mechanism for the middleware to notify the application of relevant asynchronous events, such as arrival of data corresponding to a subscription, violation of a **QoS** setting, etc. Each **DCPS** entity supports its own specialized kind of listener. **`Listener`** operations are invoked using a middleware-provided thread.
DDS Publication		A **DDS** publication is defined by the association of a **DataWriter** to a **publisher**. This association expresses the intent of the application to publish the data described by the DataWriter in the context provided by the publisher.
DDS Publisher		A **DDS** publisher is an object responsible for data distribution. It may publish data of different data types. The **DataWriter** is the object the application must use to communicate to a publisher the existence and value of data-objects of a given type. When data-object values have been communicated to the publisher through the appropriate **`DataWriter`**, it is the publisher's responsibility to perform the distribution (the publisher will do this according to its own QoS, or the **QoS** attached to the corresponding **`DataWriter`**).
DDS Subscriber		A **DDS** subscriber is an object responsible for receiving published data and making it available (according to the Subscriber's **QoS**) to the receiving application. It may receive and dispatch data of different specified types. To access the received data, the application must use a typed **DataReader** attached to the subscriber.
DDS Subscriber Access API		**DDS** defines two **APIs** that provide subscriber access: **Listeners** and the dual **Condition/WaitSet** infrastructure allow applications to be notified when changes occur in a **DCPS** communication.

DDS Subscription		A **DDS** subscription is defined by the association of a **DataReader** with a subscriber. This association expresses the intent of the application to subscribe the data described by the **`DataReader`** in the context provided by the **subscriber**.
DDS WaitSet		A **DDS** **`WaitSet`** associated with one or several **Condition** objects provides asynchronous data access. **`WaitSets`** and their associated **`Conditions`** provide the means for an application thread to block waiting for the same events that can be received via a **Listener**. Using a **`WaitSet`** the application can handle the event in its own thread instead of the middleware provided thread used for **`Listeners`**.
Defense Information Systems Agency	DISA	Combat support agency responsible for planning, engineering, acquiring, fielding, and supporting global net-centric solutions to serve the needs of the President, Vice President, the Secretary of Defense, and other DoD Components, under all conditions of peace and war. (Source: http://www.disa.mil/main/about/missman.html)
Department of Defense	DoD	A civilian Cabinet organization of the United States government. The Department of Defense controls the U.S. military and is headquartered at The Pentagon. It is headed by the Secretary of Defense. (Source: http://en.wikipedia.org/wiki/United_States_Department_of_Defense)
Deployment		The process whereby software is installed into an operational environment. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Deployment Descriptor		An XML file provided with each module and J2EE application that describes how they should be deployed. The deployment descriptor directs a deployment tool to deploy a module or application with specific container options and describes specific configuration requirements that a deployer must resolve. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Deprecate		Deprecation is the gradual phasing-out of features such as guidance, software or programming language features. Guidance, features or methods marked as deprecated are considered obsolete, and further use is discouraged. The guidance features or methods are still valid although error messages as warnings may occur when they are referenced. These serve to alert the user to the fact that the feature may be removed in future releases. Features get marked as deprecated, rather than simply removed, in order to provide backward compatibility end users.
Deserialization		Deserialization is the reverse process of **serialization**. A stream of data is converted back into a complex object. ***Note:*** *The process of transferring data using* ***serialization*** *and deserialization is called* ***marshalling****.*

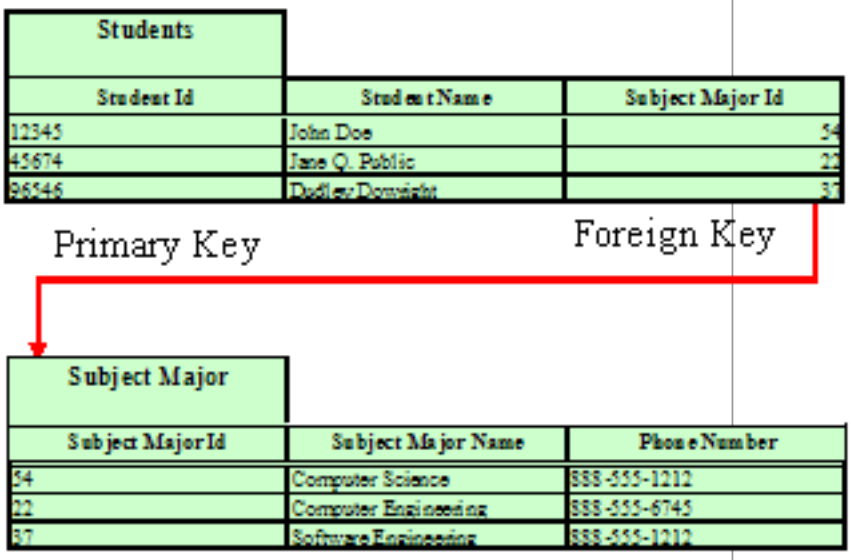
Digital Signature		A value computed with a cryptographic algorithm and bound to data in such a way that intended recipients of the data can use the signature to verify that the data has not been altered and/or has originated from the signer of the message, providing message integrity and authentication. The signature can be computed and verified with symmetric key algorithms, where the same key is used for signing and verifying, or with asymmetric key algorithms, where different keys are used for signing and verifying (a private and public key pair are used).
Digital Signature Algorithm	DSA	The Digital Signature Algorithm (DSA) is a United States Federal Government standard for digital signatures. It was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS), specified in FIPS 186, adopted in 1993. A minor revision was issued in 1996 as FIPS 186-1, and the standard was expanded further in 2000 as FIPS 186-2. (Source: http://en.wikipedia.org/wiki/Digital_Signature_Algorithm)
Directory Service		A directory service organizes computerized content and runs on a directory server computer. It is not to be confused with the directory itself, which is the database that holds the information about objects that are to be managed by the directory service. The directory service is the interface to the directory and provides access to the data that is contained in that directory. It acts as a central authority that can securely authenticate resources and manage identities and relationships between them. (Source: http://en.wikipedia.org/wiki/Directory_service)
Distributed Component Object Model	DCOM	Distributed Component Object Model (DCOM) is a Microsoft proprietary technology for software components distributed across several networked computers to communicate with each other. It extends Microsoft's COM, and provides the communication substrate under Microsoft's COM+ application server infrastructure. It has been deprecated in favor of Microsoft **.NET**.
Document Object Model	DOM	An API for accessing and manipulating XML documents as tree structures. DOM provides platform-neutral, language-neutral interfaces that enable programs and scripts to dynamically access and modify content and structure in XML documents. (Source: *J2EE 1.4 Glossary*, http://java.sun.com/j2ee/1.4/docs/glossary.html)
Document Type Definition	DTD	An optional part of the XML document prolog, as specified by the XML standard. The DTD specifies constraints on the tags and tag sequences that can be in the document. The DTD has a number of shortcomings, however, and this has led to various schema proposals. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
DoD Discovery Metadata Specification	DDMS	The DoD Discovery Metadata Specification (DDMS) defines discovery metadata elements for resources posted to

		community and organizational shared spaces. (Source: http://metadata.dod.mil/mdr/irs/DDMS/)
DoD Metadata Registry		As part of the overall **DoD Net-Centric Data Strategy**, the DoD CIO established the DoD Metadata Registry (http://metadata.dod.mil) and a related metadata registration process for the collection, storage and dissemination of structural metadata information resources (schemas, data elements, attributes, document type definitions, style-sheets, data structures, etc.). This Web-based repository is designed to also act as a clearinghouse through which industry and government coordination on metadata technology and related metadata issues can be advanced. As OASD's Executive Agent, **DISA** maintains and operates the ***DoD Metadata Registry and Clearinghouse*** under the direction and oversight of **OASD(NII)**. (Source: DoD Metadata Registry v6.0 Web site, https://metadata.dod.mil/mdr/about.htm)
DOD Net-Centric Data Strategy		This Strategy lays the foundation for realizing the benefits of net-centricity by identifying data goals and approaches for achieving those goals. To realize the vision for net-centric data, two primary objectives must be emphasized: (1) increasing the data that is available to communities or the Enterprise and (2) ensuring that data is usable by both anticipated and unanticipated users and applications. (Source: *Department of Defense Net-Centric Data Strategy*, DoD CIO, 9 May 2003, http://www.defenselink.mil/cio-nii/docs/Net-Centric-Data-Strategy-2003-05-092.pdf)
DoD PKI Class 3 Assurance Level		Applications handling unclassified medium value information in Moderately Protected Environments, unclassified high value information in Highly Protected Environments, and discretionary access control of classified information in Highly Protected Environments. This assurance level is appropriate for applications that require identification of an entity as a legal person, rather than merely as a member of an organization. ***Note:*** *This definition is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*
DoD PKI Class 4 Assurance Level		Applications that handle high value unclassified information (mission critical) in minimally protected environments will require Class 4 certificates. ***Note:*** *This definition is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*
Domain		A group of related items within a certain area of interest. In **DDS**, a domain is the basic construct used to bind individual **publications** and **subscriptions** together for communication. A distributed application can elect to use single or multiple domains for its **data-centric** communications. Domains isolate communication, promote scalability and segregate different classifications of data. (See **Global Data Space**)

Domain Analysis		The process of identifying the types of information that the data model uses. A good data model captures descriptive information about each of the types.
Domain Name System	DNS	The Domain Name System stores information about hostnames and domain names in a type of distributed database on networks, such as the Internet. Of the many types of information that can be stored, most importantly it provides a physical location (IP address) for each domain name, and lists the mail exchange servers accepting email for each domain. The DNS provides a vital service on the Internet as it allows the transmission of technical information in a user-friendly way. While computers and network hardware work with IP addresses to perform tasks such as addressing and routing, humans generally find it easier to work with hostnames and domain names (such as ***www.example.com***) in URLs and email addresses. The DNS therefore mediates between the needs and preferences of humans and of software.
Dynamic HTML	DHTML	Designates a technique of creating interactive web sites by using a combination of the static markup language HTML, a client-side scripting language such as JavaScript, and the style definition language Cascading Style Sheets. (Source: http://en.wikipedia.org/wiki/Dynamic_web_page)
Dynamic Web Page		See **DHTML**.
Electronic Business Using eXtensible Markup Language	ebXML	ebXML is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes. (Source: http://www.ebxml.org/geninfo.htm)
Electronic Data Interchange	EDI	Standard formats for exchanging business data and documents.
Encryption		Encryption is the process of obscuring information to make it unreadable without special knowledge. While encryption has been used to protect communications for centuries, only organizations and individuals with an extraordinary need for secrecy have made use of it. In the mid-1970s, strong encryption emerged from the sole preserve of secretive government agencies into the public domain, and is now employed in protecting widely-used systems, such as Internet e-commerce, mobile telephone networks and bank automatic teller machines. (Source: http://en.wikipedia.org/wiki/Encryption)
Endpoint		The URL or location of the Web service on the internet.
End User		A human user of information. This is distinct from those who develop or support the automated systems that provide the information. -OR- A person who uses a device-specific user agent to access a Web site. (Source: http://www.oasis-

		open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Enterprise		An organization considered as an entity or system that includes interdependent resources (e.g., people, organizations, and technology) that must coordinate functions and share information in support of a common mission or a set of related missions. In the computer industry, the term is often used to describe any large organization that utilizes computers. An intranet, for example, is a good example of an enterprise computing system. (Source: http://www.webopedia.com/TERM/e/enterprise.html)
Enterprise Application Archive	EAR	A JAR archive that contains a J2EE application. It contains all the JAR, WAR, and RAR archives for an enterprise application, plus an XML descriptor. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Enterprise Application Integration	EAI	Software to effect interface between enterprise software systems. Provides interface at the application layer.
Enterprise Java Bean	EJB	A server-side component architecture for the development and deployment of object-oriented, distributed, enterprise-level applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and secure. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Enterprise Service		A service that provides capabilities to the enterprise. See also **Core Enterprise Service** and **Community of Interest Service**.
Environment Variable		Environment variables are a set of dynamic values that can affect the way running processes will behave. (Source: http://en.wikipedia.org/wiki/Environment_variable)
eXtensible Access Control Markup Language	XACML	XACML is used to represent and evaluate access control policies. XACML is designed to standardize the use of declarative policy to control access to resources. Used with **SAML**.
eXtensible Markup Language	XML	A markup language defines tags (markup) to identify the content, data, and text in XML documents. It differs from **HTML**, the markup language most often used to present information on the Internet. HTML has fixed tags that deal mainly with style or presentation. An XML document must undergo a transformation into a language with style tags under the control of a style sheet before it can be presented by a browser or other presentation mechanism. Two types of style sheets used with XML are CSS and XSL. Typically, XML is transformed into HTML for presentation. Although tags can be defined as needed in the generation of an XML document, you can use a document type definition (DTD) to define the elements allowed in a particular type of document. A document can be compared by using the rules in the DTD to determine its validity and to locate particular elements in the document. A Web services application's J2EE deployment

		descriptors are expressed in XML with schemas defining allowed elements. Programs for processing XML documents use SAX or DOM APIs. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
eXtensible Style Language Transformations	XSLT	A language to express the transformation of XML documents into other XML documents. (Source: W3C Glossary)
eXtensible Stylesheet Language	XSL	A standard that lets you do the following: (1) Specify an addressing mechanism, so that you can identify the parts of an XML document that a transformation applies to (XPath). (2) Specify tag conversions, so that you can convert XML data into different formats (XSLT). (3) Specify display characteristics, such as page sizes, margins, font heights and widths, and the flow objects on each page. Information fills in one area of a page and then automatically flows to the next object when that area fills up. That allows you to wrap text around pictures, or continue a newsletter article on a different page (XSL-FO). (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Facade		Provides a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. This can simplify a number of complicated object interactions into a single interface.
Font Size		The font size refers to the size of the font from baseline to baseline, when set solid (in CSS terms, this is when the ***font-size*** and ***line-height*** properties have the same value). (Source: http://www.w3.org/TR/REC-CSS2/fonts.html)
FORCEnet	Fn	An operational construct and architectural framework that integrates the SEAPOWER21 concepts of Sea Strike, Sea Shield, and Sea Basing by connecting warriors; sensors, networks; command and control; platforms and weapons; providing accelerated speed and accuracy of decision; and integrating knowledge to dominate the battlespace. FORCEnet provides the following capabilities: expeditionary, multi-tiered, sensor and weapon grids; distributed, collaborative, command and control; dynamic, multi-path survivable networks; adaptive/automated decision aids; and human-centric integration.
Foreign Key	FK	An attribute in a relation of a database that serves as the primary key of another relation in the same database.



Global Information Grid	GIG	Globally interconnected, end-to-end set of information capabilities, associated processes, and personnel for collecting, processing, storing, disseminating, and managing information on demand to warfighters, policy makers, and support personnel. The GIG includes all owned and leased communications and computing systems and services, software (including applications), data, security services, and other associated services necessary to achieve Information Superiority. It also includes National Security Systems (NSS) as defined in section 5142 of the Clinger-Cohen Act of 1996. The GIG supports all DoD, National Security, and related Intelligence Community (IC) missions and functions (strategic, operational, tactical, and business) in war and in peace. The GIG provides capabilities from all operating locations (bases, posts, camps, stations, facilities, mobile platforms, and deployed sites). The GIG provides interfaces to coalition, allied, and non-DoD users and systems.
Hierarchical Database		A hierarchical database defines a set of parent-child relationships. Their use should be limited to integration of existing databases, such as IBM's Informational Management System (IMS). Hierarchical database systems require developers to predict all possible access patterns in advance and design the database accordingly. A database access pattern that is not included in the design becomes very difficult and inefficient.
High Availability		Data tier availability can be affected by hardware failure, power outages, data errors, user errors, programmer errors, OS errors, and RDBMS errors. Various hardware and software methods help mitigate availability issues. The more reliable a system needs to be, the more it costs. Consequently, defining availability to meet requirements is essential to controlling costs.
Hypertext Markup Language	HTML	A markup language for hypertext documents on the Internet. HTML supports embedding images, sounds, video streams,

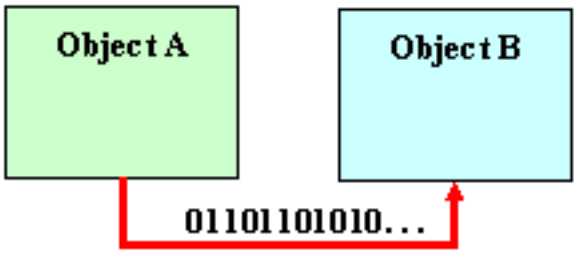
		form fields, references to other objects with URLs, and basic text formatting. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Hypertext Transfer Protocol	HTTP	The Internet protocol used to retrieve hypertext objects from remote hosts. HTTP messages consist of requests from client to server and responses from server to client. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Hypertext Transmission Protocol Over SSL	HTTPS	HTTPS is the secure version of **HTTP**, the communication protocol of the World Wide Web. It was invented by Netscape Communications Corporation to provide authentication and encrypted communication and is used in electronic commerce. Instead of using plain text socket communication, HTTPS encrypts the session data using either a version of the **SSL** (Secure Socket Layer) protocol or the **TLS** (Transport Layer Security) protocol, thus ensuring reasonable protection from eavesdroppers, and man in the middle attacks. The default TCP/IP port of HTTPS is 443. (Source: http://en.wikipedia.org/wiki/HTTPS)
Identity		Identity refers to the nature or attributes of the track: ***Friend***, ***Assumed Friend***, ***Neutral***, ***Unknown***, ***Pending***, ***Suspect***, or ***Hostile***.
Information		Data to which meaning is assigned, according to context and assumed conventions. Data that has been interpreted, translated, or transformed to reveal the underlying meaning.
Information Assurance	IA	Measures taken to protect and defend our information and information systems to ensure Confidentiality, Integrity, Availability, and Accountability, extended to restoration with protect, detect, monitor, and react capabilities.
Infrastructure		
Integrated Development Environment	IDE	
Integration		Integration is the action or process of combining elements so that they become a whole. Vertical integration acts within a system, whereas horizontal integration acts between or among systems. In the net-centric environment, integration creates links between computer systems, applications, services, or processes. The word is normally used in the context of computing, but can apply to business processes as much as to the underlying process automation. In the past, computer integration such as **enterprise application integration** (EAI) has typically been tightly coupled, or "hard wired," making it difficult to adapt to changing requirements. Thanks to the advent of Web services and the evolution of service-oriented architectures, more agile, loosely coupled forms of integration are starting to emerge.
Integrity		The property that data has not been modified (digital signature).

Interface		The functional and physical characteristics required to exist at a common boundary or connection between systems or items. (Source: DoD 4120.214-M)
Interface Definition Language	IDL	A language used to define interfaces to remote **CORBA** objects. The interfaces are independent of operating systems and programming languages. (Source: http://java.sun.com/javaee/reference/glossary/index.jsp#120354)
Internet		The Internet, or simply the Net, is the publicly available worldwide system of interconnected computer networks that transmit data by packet switching using a standardized Internet Protocol (IP) and many other protocols. It is made up of thousands of smaller commercial, academic, and government networks. It carries various information and services, such as electronic mail, online chat and the interlinked web pages and other documents of the World Wide Web. Because this is by far the largest, most extensive internet (with a lower case i) in the world, it is simply called the Internet (with a capital I). (Source: http://en.wikipedia.org/wiki/Internet)
Internet Engineering Task Force	IETF	The Internet Engineering Task Force (IETF) is a large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interested individual. (Source: http://www.ietf.org/overview.html)
Internet Information Services	IIS	A set of Internet-based services for Windows machines. Originally supplied as part of the Option Pack for Windows NT, they were subsequently integrated with Windows 2000 and Windows Server 2003. The current (Windows 2003) version is IIS 6.0 and includes servers for FTP, SMTP, NNTP and HTTP/HTTPS. Earlier versions also included a Gopher server.
Internet Inter-ORB Protocol	IIOP	A protocol used for communication between CORBA object request brokers. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Internet Protocol	IP	Data packets routed across network, not switched via dedicated circuits.
Interoperability		The ability of systems, units, or forces to provide data, information, materiel, and services to and accept the same from other systems, units, or forces, and to use the data, information, materiel, and services so exchanged to enable them to operate effectively together. DoD Information Technology (IT) and National Security Systems (NSS) interoperability includes both the technical exchange of information and the end-to-end operational effectiveness of that exchanged information as required for mission accomplishment. Interoperability is more than just information exchange. It includes systems, processes, procedures, organizations, and missions over the life cycle and must be balanced with information assurance. (Source: adapted from CJCSI 6212.01D, 8 March 2006 and Committee on National

		Security Systems Instruction 4009, *National Information Assurance (IA) Glossary*, revised May 2003)
Intranet		An intranet is a local area network (LAN) used internally in an organization to facilitate communication and access to information that is sometimes access-restricted. Sometimes the term refers only to the most visible service, the internal web site. The same concepts and technologies of the Internet such as clients and servers running on the Internet protocol suite are used to build an intranet. HTTP and other internet protocols are commonly used as well, especially FTP and email. There is often an attempt to use internet technologies to provide new interfaces with corporate "legacy" data and information systems. (Source: http://en.wikipedia.org/wiki/Intranet)
ISO/IEC 11179		See **ISO-11170**.
ISO-11170		ISO-11179 (formally known as the ISO/IEC 11179 Metadata Registry Standard) is a standard for representing for an organization in a **Metadata Registry**.
J2EE Application		Any deployable unit of J2EE functionality. This can be a single J2EE module or a group of modules packaged into an EAR file along with a J2EE application deployment descriptor. J2EE applications are typically engineered to be distributed across multiple computing tiers. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
J2EE Component		A self-contained functional software unit that is supported by a container and is configurable at deployment time. The **J2EE** specification defines the following J2EE components. (1) Application clients and **applets** are components that run on the client. (2) Java servlet and JavaServer Pages (JSP) technology components, Web components that run on the server. (3) Enterprise JavaBeans (EJB) components (enterprise beans), business components that run on the server. J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server or client container. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
J2EE Server		The runtime portion of a J2EE product. A J2EE server provides EJB or Web containers or both. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Java		Java is a reflective, object-oriented programming language developed initially by at Sun Microsystems. It was intended to replace C++, although the feature set better resembles that of Objective-C. Java should not be confused with JavaScript, which shares only the name and a similar C-like syntax. Sun Microsystems currently maintains and updates Java regularly.

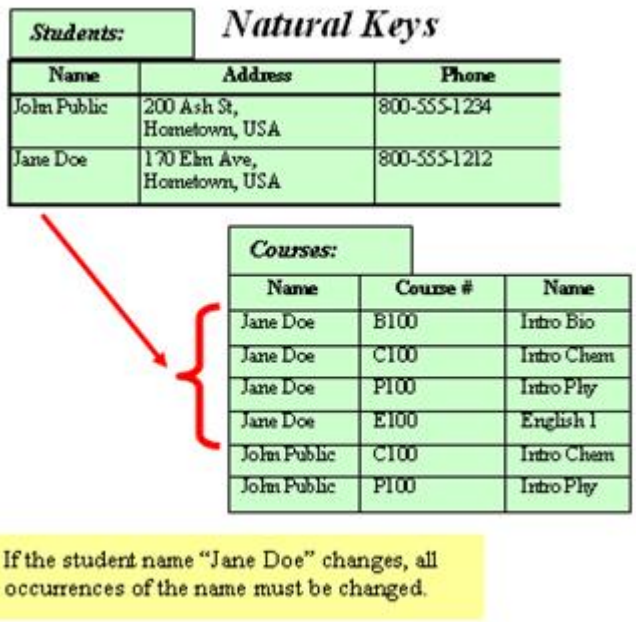
		Specifications of the Java language, the Java Virtual Machine (JVM) and the Java API are community-maintained through the Sun-managed Java Community Process.
Java 2 Platform, Enterprise Edition	J2EE	The J2EE environment is the standard for developing component-based multi-tier enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications. Features include Web services support and development tools. Sun Microsystems has simplified the name of the Java platform for the enterprise; the "2" is dropped from the name, as well as the dot number so the next version of the Java platform for the enterprise is **Java Platform, Enterprise Edition** 5 or Java EE 5.(Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Java Archive	JAR	A platform-independent file format that enables you to bundle multiple files into a single archive file. JAR files are packaged with the ZIP file format, so you can use them for ZIP-like tasks such as lossless data compression, archiving, decompression, and archive unpacking. Typically JAR files contain the class files and auxiliary resources associated with applets and applications. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Java Class Files		Class files contain bytecodes for the Java Virtual Machine. They are normally produced by a compiler for the Java programming language. A Java interpreter can then read these files and execute the code contained within.
Java Connector Architecture	J2A	The J2EE Connector architecture defines a standard architecture for connecting the J2EE platform to heterogeneous EISs [enterprise information systems]. Examples of EISs include ERP, mainframe transaction processing, database systems, and legacy applications not written in the Java programming language. By defining a a set of scalable, secure, and transactional mechanisms, the J2EE Connector architecture enables the integration of EISs with application servers and enterprise applications. (Source: http://java.sun.com/j2ee/connector/reference/industrysupport/index.html)
Java Database Connection	JDBC	An API that supports database and data-source access from Java applications.
Java Development Kit	JDK	
Java Message Service	JMS	An API for invoking operations on enterprise messaging systems. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Java Naming and Directory Interface	JNDI	An API that provides naming and directory functionality. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Java Platform, Enterprise Edition	Java EE	Java Platform, Enterprise Edition (Java EE) is the industry standard for developing portable, robust, scalable and secure

		server-side Java applications. Building on the solid foundation of the Java Platform, Standard Edition (Java SE), Java EE provides Web services, component model, management, and communications APIs that make it the industry standard for implementing enterprise-class service-oriented architecture (SOA) and next-generation Web applications. Sun Microsystems has simplified the name of the Java platform for the enterprise. Formerly, the platform was known as Java 2 Platform, Enterprise Edition (**J2EE**), and specific versions had "dot numbers" such as J2EE 1.4. The "2" is dropped from the name, as well as the dot number so the next version of the Java platform for the enterprise is Java Platform, Enterprise Edition 5 or Java EE 5. (Source: http://java.sun.com/javaee/)
JavaScript		The Netscape-developed object scripting language used in millions of web pages and server applications worldwide. Contrary to popular misconception, JavaScript is not "Interpretive Java." Rather, it is a dynamic scripting language that supports prototype-based object construction.
JavaServer Page	JSP	An extensible Web technology that uses static data, JSP elements, and server-side Java objects to generate dynamic content for a client. Typically the static data is HTML or XML elements, and in many cases the client is a Web browser. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Joint Interoperability Test Command	JITC	Independent operational test and evaluation/assessor of DISA and other DoD Command, Control, Communications, Computers and Intelligence (C4I) acquisitions. (Source: http://jitc.fhu.disa.mil/mission.htm)
JScript		Microsoft's extended implementation of ECMAScript (ECMA262), an international standard based on Netscape's JavaScript and Microsoft's JScript languages. JScript is implemented as a Windows Script engine. This means that you can plug it in to any application that supports Windows Script, such as Internet Explorer, Active Server Pages, and Windows Script Host. It also means that any application supporting Windows Script can use multiple languages: JScript, VBScript, Perl, and others.
Just-In-Time Compilation	JIT	This is the primary method by which **.NET** executes **MSIL**. As the MSIL is executed, the code is compiled and optimized for the executing environment. JIT compilation provides environment optimization, runtime type safety, and assembly verification. To accomplish this, the JIT compiler examines the assembly metadata for any illegal accesses and handles violations appropriately.
Key Recovery Manager	KRM	A service of the DOD PKI where copies of key pairs used for encryption are stored and can be recovered for law enforcement purposes. ***Note:*** *This definition is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*

Knowledge		(Unlike information or data) Requires the presence of context, semantics, and purpose.
Land C2 Information Exchange Data Model	LC2IEDM	
Light Directory Access Protocol	LDAP	A set of protocols for accessing information directories. LDAP is a simpler version of the X.500 standard. Unlike X.500, LD Web Services for Interactive Applications AP supports TCP/IP, which is necessary for Internet access. Because it's a simpler version of X.500, LDAP is sometimes called X.500-lite. LDAP is a protocol for accessing on-line directory services. (Source: http://en.wikipedia.org/wiki/LDAP)
Local Area Network	LAN	A group of interconnected computer and support devices. (Source: http://www.sun.com/products-n-solutions/hardware/docs/html/817-6210-10/glossary.html)
Look and Feel		Look and feel refers to design aspects of a graphical user interface in terms of colors, shapes, layout, typefaces, etc. (the "look"); and, the behavior of dynamic elements such as buttons, boxes, and menus (the "feel"). It is used in reference to both software and **Web sites**. (Source: http://en.wikipedia.org/wiki/Look_and_feel)
Loosely Coupled		A computing model where application elements require a simple level of coordination and allow for flexible reconfiguration. Interconnection is often asynchronous and message-based.
Marshalling		The process of transferring data using **serialization** and **deserialization** is called marshalling. 
Mediation		A set of negotiated agreements for interacting between components that enable those components to work together to perform a task. These agreements are defined through standard interfaces and data interchange specifications. Mediation services provide multiple methods for integrating data sources and services: • Transformation • Aggregation

		• Adaptation • Orchestration • Choreography
Message		A complete unit of data available to be sent or received by services. It is a self-contained unit of information exchange. A message always contains a **SOAP** envelope, and may include additional **MIME** parts as specified in MTOM, and/or transport.
Message-Oriented Middleware	MOM	Message-oriented middleware acts as an arbitrator between incoming and outgoing messages to insulate producers and consumers from other producers and consumers.
Metadata		Data about the data, that is, the description of the data resources, its characteristics, location, usage, and so on. Metadata is used to identify, describe, and define user data.
Metadata Registry		A Metadata Registry is a central place where metadata definitions are stored and maintained. A metadata registry typically has the following characteristics: • It is a protected area where only approved individuals may make changes • It stores data elements that include both semantics and representations • The semantic areas of a metadata registry contain the meaning of a Data Element with precise definitions • The representational areas define how the data is represented in a specific format such as within a database or a structure file format such as XML Metadata Registries often are stored in an international format called **ISO-11170**.
Microsoft Intermediate Language	MSIL	An intermediate instruction set into which all **.NET** languages compile. You can execute MSIL code on any environment that supports the .NET framework. MSIL-compiled code is verified for safety during runtime, providing better security and reliability than natively compiled binaries. During compilation, .NET code is translated into Microsoft Intermediate Language (MSIL) rather than machine-specific binary code. MSIL is a machine- and platform-independent instruction set that can be executed in any environment within the .NET framework. .NET uses **just-in-time (JIT) compilation** as its primary means of executing MSIL. You can generate native binary images using Microsoft's Native Image Generator (**NGEN**).
Microsoft Message Queue	MSMQ	Messaging in .NET uses Microsoft Message Queue (MSMQ). MSMQ is responsible for reliably delivering messages

		between applications inside and outside the enterprise. MSMQ ensures reliable delivery by placing messages that fail to reach their intended destination in a queue and then resending them once the destination is reachable. MSMQ also supports transactions. It permits multiple operations on multiple queues, with all of the operations wrapped in a single transaction, thus ensuring that either all or none of the operations will take effect. Microsoft Distributed Transaction Coordinator (MSDTC) supports transactional access to MSMQ and other resources.
Mission		The task, together with the purpose, that clearly indicates the action to be taken and the reason for that action.
Model-Driven Architecture	MDA	Model-driven architecture is a trademarked term denoting a specific approach to the development of software using models as the basis. The MDA specifies system functionality separately from the implementation of that functionality on a specific technology platform. To accomplish this goal, the MDA defines an architecture that provides a set of guidelines for structuring specifications expressed as models. The MDA model architecture relates multiple standards, including Unified Modeling Language (UML), the Meta Object Facility (MOF), the XML Metadata interchange (XMI), and the Common Warehouse Metamodel (CWM). Note that the term "architecture" in MM does not refer to the architecture of the system being modeled, but rather to the architecture of the various standards and model forms that serve as the technology basis for MDA .
Multi-Purpose Internet Mail Extensions	MIME	
Namespace		A standard that lets you specify a unique label for the set of element names defined by a **DTD** or **XSD**. A document using that DTD or XSD can be included in any other document without causing a conflict between element names. The elements defined in a particular DTD are uniquely identified so that, for example, the parser can tell when an element **`name`** should be interpreted according to the particular DTD rather than using the definition for an element **`name`** in a different DTD. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Native Image Generator	NGEN	NGEN compilation enables you to production of a native binary image of **MSIL** code for the current environment. This improves the performance of the **.NET** application by eliminating the **JIT** overhead associated with the execution. Running NGEN against an assembly, the resulting native

		image is placed in the Global Assembly Cache for use by all other .NET assemblies. NGEN is a good tool for improving performance of .NET applications as long as the executing environment remains static. If executing an NGEN-generated image in an incompatible environment, .NET automatically reverts to using JIT. To mitigate this, run NGEN during deployment against the installed assemblies.
Native XML Database		Defines a logical model for an XML document (as opposed to the data in that document) and stores and retrieves documents according to that model. These databases are accessed via programming interfaces such as SAX, DOM, or JDOM. There is a trend away from pure XML storage because all the leading relational database vendors are introducing advanced XML capabilities.
Natural Key		A Natural Key is a primary keys that is made up completely or in part from naturally occurring data in the tables.  See **Surrogate Key** and **Primary Key**.
Niche Databases		Various vendors create niche databases in response to shortcomings in relational databases. Market domination by large vendors has made it hard for small vendors to break into the market, so niche database vendors mainly provide supporting tools.
Nonce		A unique random string.
Normalization		Normalization avoids duplication of data, insert anomalies, delete anomalies, and update anomalies. A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only. A relation is in second normal form (2NF) if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key. A relation is in third normal form (3NF) if and only if it is in 2NF and every

		non-key attribute is non-transitively dependent on the primary key. Data models should follow the three forms unless there is overriding justification not to. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
North Atlantic Treaty Organization	NATO	NATO is an international organization for defense collaboration established in 1949, in support of the North Atlantic Treaty signed in Washington, D.C., on April 4, 1949. Its other official name is the French equivalent, l'Organisation du Trait de l'Atlantique du Nord (OTAN).
Object Management Group	OMG	OMGTM is an international, open membership, not-for-profit computer industry consortium. OMG Task Forces develop enterprise integration standards for a wide range of technologies, and an even wider range of industries. OMG's modeling standards enable powerful visual design, execution and maintenance of software and other processes. OMG's middleware standards and profiles are based on the **Common Object Request Broker Architecture** (CORBA) and support a wide variety of industries. (Source: http://www.omg.org/)
Object-Oriented Analysis	OOA	OOA (Object Oriented Analysis) constitutes the development of software engineering requirements and specifications for a system. These are expressed as an object model (object oriented design) which is composed of a population of interacting objects.
Object-Oriented Databases	OODBMS	Object-oriented databases are based on the object model, and use the same conceptual models as **object-oriented analysis** and **design**.
Object-Oriented Design		Any design that incorporates objects, classes, and inheritance. Contrast with object-based design and class-based design.
Object-Oriented Programming Language		A programming language that enables programmers to define and use objects, classes, and inheritance; for example, C++, Ada 95.
Object Request Broker	ORB	A library that enables **CORBA** objects to locate and communicate with one another. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Online Certificate Status Protocol	OCSP	Online Certificate Status Protocol is a method for determining the revocation status of an X.509 digital certificate using means other than **CRLs**. It is described in RFC 2560 and is on the Internet standards track. OCSP messages are encoded in ASN.1 and usually communicated over **HTTP**. OCSP's request/response nature leads to OCSP servers being termed as OCSP responders.
Online Status Check	OSC	OSC is service that may be provided by the **Certificate Authority** (CA). A relying party sends a request to the OSC service with a certificate, the OSC service responds with a digitally signed response that includes the date and time, certificate identification, and the status of the certificate about whose validity the relying party inquired. The possible

		responses include "unknown" which may be the response to a query regarding an expired certificate. ***Note:*** *This definition is derived from the DoD Class 3 PKI Public Key-Enabled Application Requirements Document, Version 1.0, 13 July 2000.*
Online Status Check Responder	OSCR	OSCR is the server that responds to a relying party's OSC request.
Ontology		An explicit specification of how to represent the objects and concepts that exist in some area of interest and of the relationships that pertain among them. (Source: DoD 8320.02-G, 12 April 2006, Guidance for Implementing Net-Centric Data Sharing)
Open Database Connectivity	ODBC	In computing, Open Database Connectivity (ODBC) provides a standard software API method for using database management systems (DBMS). The designers of ODBC aimed to make it independent of programming languages, database systems, and operating systems. (Source: http://en.wikipedia.org/wiki/Odbc; 30 March 2007)
Open Standard		Open standards are publicly available specifications for achieving a specific task. By allowing anyone to obtain and implement the standard, they can increase compatibility between various hardware and software components, since anyone with the necessary technical know-how and resources can build products that work together with those of the other vendors that base their designs on the standard (although patent holders may impose "reasonable and non-discriminatory" royalty fees and other licensing terms on implementers of the standard). Source: http://en.wikipedia.org/wiki/Open_standard) ***Note:*** *NESI restricts the use of the term "standard" to technologies approved by formalized committees that are open to participation by all interested parties and operate on a consensus basis.*
Organization for the Advancement of Structured Information Standards	OASIS	A not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. (Source: http://www.oasis-open.org/who/)
OS File Systems		A file system that stores and retrieves data, acting as a data tier. Advocates cite performance and simplicity, but the loss of DBMS-inherent capabilities such as ad-hoc queries and the ability to upgrade to faster machines is a deterrent. File-system-based data tiers often result in proprietary solutions that are hard to maintain and port.
Parser		A module that reads in XML data from an input source and breaks it into chunks so that your program knows when it is working with a tag, an attribute, or element data. A non-validating parser ensures that the XML data is well formed but does not verify that it is valid. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)

Personalization		The ability for portal members to subscribe to specific types of content and services. Users can customize the look and feel of their environment.
Personal Web Server	PWS	A **Web server** program for personal computer users who want to share **Web pages** and other files from their hard drive. PWS is a scaled-down version of Microsoft's more robust Web server, Internet Information Server (**IIS**). PWS can be used with a full-time **Internet** connection to serve Web pages for a **Web site** with limited traffic. It can also be used for testing a Web site offline or from a "staging" site before putting it on a main Web site that is exposed to more traffic.
Physical Model		Translates the conceptual model to a particular RDBMS implementation.
Portability		The ease with which a system or component can be transferred from hardware or software environment to another. (Source: IEEE Std 610.12-1990) The level of software portability of any specific product depends on two factors: the design of the product itself, and the characteristics of the source and target execution environments. Software products are rarely if ever 100% portable. Generally, the level of portability depends on the target platform. Software that is highly portable to one class of platform might be not portable to other classes.
Portable Object Adapter	POA	A CORBA standard for building server-side applications that are portable across heterogeneous ORBs. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Portable Operating System Interface for Computing Environments	POSIX	
Portal		A Web portal is a **Web site** that provides a starting point, gateway, or portal to other resources on the **Internet** or an intranet. Intranet portals are also known as "enterprise information portals" (EIP). Examples of existing portals are Yahoo, Excite, Lycos, Altavista, Infoseek, and Hotbot. (Source: http://en.wikipedia.org/wiki/web_portal)
Portal Page		A complete document rendered by a portal. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Portlet		A reusable Web component that displays relevant information to portal users. Examples for portlets include email, weather, discussion forums, and news. The purpose of the **Web Services for Remote Portlets** (WSRP) interface is to provide a **Web services** standard that allows for the "plug-n-play" of **portals**, other intermediary **Web applications** that aggregate content, and applications from disparate sources. The portlet specification enables interoperability between portlets and portals. This specification defines a set of **APIs** for portal computing that addresses the areas of aggregation, personalization, presentation, and security. (Source: http://en.wikipedia.org/wiki/Portlets)

Portlet Container		A portlet container provides a runtime environment for **portlets** implemented according to the portlet **API**. In this environment portlets can be instantiated, used, and finally destroyed. The portlet container is not a standalone container like the **servlet** container; instead it is implemented as a thin layer on top of the servlet container and reuses the functionality provided by the servlet container. (Source: http://portals.apache.org/pluto/)
Portlet Specification	JSR 168	To enable interoperability between **portlets** and **portals**, this specification defines a set of **APIs** for portal computing that address the areas of aggregation, personalization, presentation, and security. (Source: http://www.jcp.org/en/jsr/detail?id=168)
Primary Key	PK	An object that uniquely identifies a row within a table.
Private Key		The private key is one of a pair of keys that are generated as part of asymmetric key cryptography. The private key is kept secret and the public key is public and can be shared openly with others.
Producer		A **Web service** conforming to the **WSRP** specification. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Protocol		An agreed-upon format for transmitting data between two devices. The protocol determines the type of error checking to be used, data compression method, if any, how the sending device will indicate that it has finished sending a message, and how the receiving device will indicate that it has received a message. (Source: http://www.webopedia.com/TERM/p/protocol.html)
Proxy Pattern		Provides a surrogate or placeholder for another object to control access to it.
Public Key	PK	See **Public Key Cryptography**.
Public Key Certificate		Used in client-certificate authentication to enable the server, and optionally the client, to authenticate each other. The public key certificate is the digital equivalent of a passport. It is issued by a trusted organization, called a certificate authority, and provides identification for the bearer. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Public Key Cryptography		Public key cryptography, also known as asymmetric cryptography, is a form of cryptography in which a user has a pair of cryptographic keys - a public key and a private key. The private key is kept secret, while the public key may be widely distributed. The keys are related mathematically, but the private key cannot be practically derived from the public key. A message encrypted with the public key can be decrypted only with the corresponding private key. (Source: http://en.wikipedia.org/wiki/Public_key)

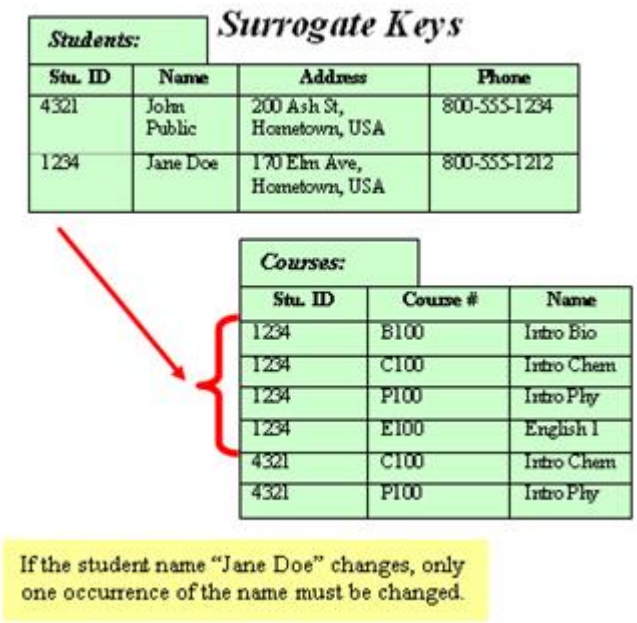
Public Key Enabling	PK-Enabling	The incorporation of the use of certificates for security services such as authentication, confidentiality, data integrity, and nonrepudiation. PK-Enabling involves replacing existing or creating new user authentication systems using certificates instead of other technologies, such as userid and password or **Internet Protocol** filtering; implementing public key technology to digitally sign, in a legally enforceable manner, transactions and documents; or using public key technology, generally in conjunction with standard symmetric encryption technology, to encrypt information at rest and/or in transit. (Source: DoDI 8520.2, *Public Key Infrastructure (PKI) and Public Key (PK) Enabling*, 1 April 2004 [R1206])
Public Key Infrastructure	PKI	Framework established to issue, maintain, and revoke public key certificates accommodating a variety of security technologies, including the use of software. (Source: CNSS Instruction No. 4009, Revised May 2003, *National Information Assurance (IA) Glossary*)
Publish/Subscribe Messaging System		A messaging system in which clients address messages to a specific node in a content hierarchy, called a topic. Publishers and subscribers are generally anonymous and can dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a node's multiple publishers to its multiple subscribers. Messages are generally not persistent and will only be received by subscribers who are listening at the time the message is sent. A special case known as a "durable subscription" allows subscribers to receive messages sent while the subscribers are not active. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Quality of Service	QoS	Data timeliness, accuracy, completeness, integrity, and ease of use. Refers to the probability of the network meeting a given traffic contract. In many cases is used informally to refer to the probability of a packet passing between two points in the network. (Source: http://en.wikipedia.org/wiki/Quality_of_service) -OR- A defined level of performance that adapts to the environment in which it is operating. QoS may be requested by the user of the information. The level of QoS provided is based on the request, the available capabilities of the provider, and the priority of the user.
Real-Time		An operation within a larger dynamic system is called a real-time operation if the combined reaction- and operation-time of a task is shorter than the maximum delay that is allowed, in view of circumstances outside the operation. The task must also occur before the system to be controlled becomes unstable. A real-time operation is not necessarily fast, as slow systems can allow slow real-time operations. This applies for all types of dynamically changing systems. The polar opposite of a real-time operation is a batch job with interactive timesharing falling somewhere in-between the two extremes. (Source: http://en.wikipedia.org/wiki/Real_time)
Refactoring		Refactoring is often used to describe modifying source code without changing its external behavior, and is sometimes informally referred to as "cleaning it up." Refactoring

		is often practiced as part of the software development cycle: developers alternate between adding new tests and functionality and refactoring the code to improve its internal consistency and clarity. Testing ensures that refactoring does not change the behavior of the code.
Reference Data Set		The Reference Data Set Gallery [of the **DoD Metadata Registry and Clearinghouse**] provides collections of related data that represent a defined entity within a community of interest. Examples of reference data sets include country codes, U.S. state codes, and marital status codes. (Soure: http://www.disa.mil/nces/development/developer_doc_overview.html)
Referential Integrity		A feature provided by RDBMSs that prevents users or applications from entering inconsistent data. Most RDBMSs have various referential integrity rules that you can apply when you create a relationship between two tables.
Registered Namespace		A namespace that has been registered and approved with a **namespace** registration services. For the DoD, use the **DoD Metadata Registry**.
Relational Database	RDB	A collection of data items organized as a set of formally-described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables.
Relational Database Management System	RDBMS	A database management system (DBMS) that is based on the relational model or that presents the data to the user as relations. A collection of tables, each table consisting of a set of rows and columns, can satisfy this property. RDBMSs also provide relational operators to manipulate the data in tabular form. (Source: http://en.wikipedia.org/wiki/RDBMS)
Relative Font Size		Fonts that display according to the size of the surrounding text. Some designers call them scalable fonts. Instead of displaying a fixed pixel size, a relative font size displays as a percentage of the surrounding elements. http://www.netmechanic.com/news/vol5/design_no13.htm
Remote Method Invocation	RMI	A technology that allows an object running in one Java virtual machine to invoke methods on an object running in a different Java virtual machine. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Remote Procedure Call	RPC	An alternative to sockets that abstracts the communication interface to the level of a procedure call. The programmer has the illusion of calling a local procedure, but in fact the arguments of the call are packaged and sent to the remove target of the cell. RPC systems encode arguments and return values using an external data representation such as XDR. RPC does not translate well into distributed object systems, which require communication between program-level objects in different address spaces. To match the semantics of object invocation, distributed object systems require RMI. A local surrogate (stub) object manages the invocation on a remote object.

Resource Adaptor Archive	RAR	A **J2EE component** that implements the **J2EE Connector Architecture** for a specific Enterprise Information System (EIS). **J2EE applications** communicate with an EIS through the resource adapter. You can deploy RARs on any **J2EE server**. A RAR file may be independent or contained in an **EAR** file.
Resource Definition Framework	RDF	
Rollback		The point in a transaction when all updates to any resources involved in the transaction are reversed. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Sans Serif Font		A sans serif font is a font that has no serifs. Examples are ***Arial***, ***Century Gothic***, and ***Helvetica***. (Source: http://web.mit.edu/abiword_v2.0.10/Tutorials/klw/glossary.html)
Schema		A diagrammatic representation, an outline, or a model. In relation to data management, a schema can represent any generic model or structure that deals with the organization, format, structure, or relationship of data. Some examples of schemas are (1) a database table and relational structure, (2) a **document type definition** (DTD), (3) a data structure used to pass information between systems, and (4) an **XML schema document** (XSD) that represents a data structure and related information encoded as XML. Schemas typically do not contain information specific to a particular instance of data (Source: DoD 8320.02-G, 12 April 2006, *Guidance for Implementing Net-Centric Data Sharing*)
Secret Key		The asymmetric key cryptography approach generates two keys, a public key and a private key. The **private key** is often referred to as the secret key.
Secure Hash Algorithm	SHA	The SHA (Secure Hash Algorithm) family is a set of related cryptographic hash functions. In cryptography, a cryptographic hash function is a hash function with certain additional security properties to make it suitable for use as a primitive in various information security applications, such as authentication and message integrity. A hash function takes a long string (or message) of any length as input and produces a fixed length string as output, sometimes termed a message digest or a digital fingerprint. (Source: http://en.wikipedia.org/wiki/SHA#SHA-0_and_SHA-1)
Secure Sockets Layer	SSL	A protocol for transmitting private documents via the Internet. SSL uses a cryptographic system employing two keys to encrypt data: a public key known to everyone and a private or secret key known only to the recipient of the message. (Source:http://www.webopedia.com/TERM/S/SSL.html)
Security Assertion Markup Language	SAML	An XML standard for exchanging authentication and authorization data between security domains; that is, between an identity provider and a service provider. SAML is a product of the **OASIS** Security Services Technical Committee. (Source:http://en.wikipedia.org/wiki/SAML)

Semantics		The implied meaning of data, the study or words and their meanings.
Serialization		Serialization is the process of writing a complex object into a serial stream of data. When the data is successfully transferred, the data can be **deserialized** back into a complex object. ***Note:*** *The process of transferring data using serialization and* ***deserialization*** *is called* ***marshalling****.*
Serif Font		A serif is a feature of the letters in a given typeset. They appear at the end of lines within the letters. An example would be the letter T in Times New Roman - at the end of each horizontal line is a tick that hangs down (that is the serif). Serif fonts include ***Times New Roman***, ***Bookman Oldstyle***, and ***Courier***.
Server		A computer software application that carries out some task (i.e., provides a service) on behalf of yet another piece of software called a **client**.
Service		A service is any function that has a clearly defined interface accessed through well-defined public access points.
Service-Oriented Architecture	SOA	Services enable access to data and application functionality through public interfaces exposed to the enterprise.
Service Provider		The person, organization, or automated asset that implements and operates a service.
Service Registry		Provides descriptive information about a service, enabling the lookup and discovery of services.
Servlet		A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web applications using a request-response paradigm. (Source:http://java.sun.com/j2ee/1.4/docs/glossary.html)
Session Key		Synonymous with the **secret** or **private key**.
Simple Mail Transfer Protocol	SMTP	
Simple Object Access Protocol	SOAP	SOAP is a lightweight XML-based messaging protocol used to encode the information in **Web service** request-and-

		response messages before sending them over a network. SOAP messages are independent of any operating system or protocol and may be transported using a variety of Internet protocols, including **SMTP**, **MIME**, and **HTTP**. (Source: http://www.webopedia.com/TERM/S/SOAP.html)
Simple Structured Data		Simple Structured Data has an uncomplicated data structure. All requisite **metadata** is provided and simple data types only are used (e.g., integers, long integers, strings, and simple lists.
Simple Unstructured Data		Simple Unstructured Data has uncomplicated data structure but not all requisite **metadata** is provided.
Single Sign-On	SSO	
Single Touch Point		The portal becomes the delivery mechanism for all business information services.
Software Communications Architecture	SCA	An implementation-independent framework for the development of software for an established hardware platform, such as software defined radios.
Software Component		A software component is a software system element offering a predefined service and able to communicate with other components. It is a unit of independent deployment and versioning, encapsulated, multiple-use, non-context-specific and composeable with other components. Source: http://en.wikipedia.org/wiki/Software_component#Software_component
Stored Procedure		A unit or module of code that executes in a database and implement some bit of application logic or business rule. Often written in proprietary language such as Oracle's PL/SQL or Sybase's Transact-SQL.
Stovepipe System		A stovepipe system is a legacy system that is an assemblage of inter-related elements that are so tightly bound together that the individual elements cannot be differentiated, upgraded or refactored. The stovepipe system must be maintained until it can be entirely replaced by a new system. Examples of stovepipe systems: • Systems for which new hardware is no longer available • Systems whose original source code has been lost • Systems that were built using old or ad hoc engineering methodologies for which support can no longer be found The term is also used to describe a system that does not interoperate with other systems, presuming instead that it is the only extant system. A stovepipe system is an example of an anti-pattern legacy system and demonstrates software brittleness. (Source: http://en.wikipedia.org/wiki/Stovepipe_system)

Structured Query Language	SQL	The standardized relational database language for defining database objects and manipulating data. (Source:http://java.sun.com/j2ee/1.4/docs/glossary.html)
Structured Query Language 1992	SQL-92	The SQL-92 and SQL:1999 standards are very detailed and specific. At the current time, no **RDBMS** vendors fully support the entire standard. Vendors that claim they are SQL-92-compliant or SQL:1999-compliant are actually only compliant to a certain level. The SQL-92 standard defines the following levels, which also apply to SQL:1999: (1) Notational; (2) Transitional level SQL92; (3) Intermediate level SQL92; (4) .Full SQL92. (Source:http://dbs.uni-leipzig.de/en/lokal/standards.pdf; http://developer.mimer.com/documentation/html_82/Mimer_SQL_Reference_Manual/Intro_SQL_Stds3.html)
Structured Query Language 1999	SQL-99	See **SQL-92**.
Style Sheet		Style sheets describe how documents are presented on screens, in print, or perhaps how they are pronounced. (Source: http://www.w3.org/Style)
Surrogate Key		A surrogate key is a primary key that has been explicitly created and has no relationship with the naturally occurring data found within a table.  See **Natural Key** and **Primary Key**.
Symmetric Key Algorithm		Encryption algorithm where the same key is used for both encrypting and decrypting a message.
System		Two or more interrelated pieces of equipment (or sets) arranged in a package to perform an operational function or to

		satisfy a requirement. (Source: *Defense Acquisition Glossary of Terms*, Jan 2001)
System Component		A basic part of a system. System components may be personnel, hardware, software, facilities, data, material, services, and/or techniques that satisfy one or more requirements in the lowest levels of the functional architecture. System components may be subsystems and/or configuration items. ***Note:*** *See* ***component****.*
Taxonomy		The science of categorization, or classification, of things based on a predetermined system. In reference to Web sites and portals, a site's taxonomy is the way it organizes its data into categories and subcategories, sometimes displayed in a site map. (Source: http://www.webopedia.com/TERM/t/taxonomy.html)
Taxonomy Gallery		The Taxonomy Gallery [of the **DoD Metadata Registry and Clearinghouse**] provides XML-based **taxonomy** files that describe one or more nodes in a hierarchical classification of items, and their relationships to other nodes. The taxonomy files registered with the Taxonomy Gallery are organized by governance namespace. (Source: http://www.disa.mil/nces/development/developer_doc_overview.html)
Tenet		Net-centric design precept.
Topic		Topics are used to manage content flow between publishers and subscribers. Topics must be known in such a way that subscribers can refer to them unambiguously. In **DDS**, Topics conceptually fits between **publications** and **subscriptions** and associate a name (unique in the **domain**), a data-type, and **QoS** parameters related to the data.
Transaction		A set of input data that triggers execution of a specific processor job. Usually manipulates data that may need to be rolled back to the original values if any part of the transaction fails. Transactions enable multiple users to access the same data concurrently. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Transmission Control Protocol	TCP	One of the core protocols of the Internet protocol suite. Using TCP, programs on networked computers can create connections to one another, over which they can send data. The protocol guarantees that data sent by one endpoint will be received in the same order by the other, without any pieces missing. It also distinguishes data for different applications (such as a Web server and an email server) on the same computer. (Source: http://en.wikipedia.org/wiki/Transmission_Control_Protocol)
Transmission Control Protocol/Internet Protocol	TCP/IP	A suite of communications protocols used to connect hosts on the Internet. TCP/IP uses several protocols, the two main ones being TCP and IP. TCP/IP is built into the UNIX operating system and is used by the Internet, making it the

		de facto standard for transmitting data over networks. Even network operating systems that have their own protocols, such as Netware, also support TCP/IP.
Transport Layer Security	TLS	A protocol that guarantees privacy and data integrity between client/server applications communicating over the Internet. The TLS protocol is made up of two layers: • The TLS Record Protocol -- layered on top of a reliable transport protocol, such as TCP, it ensures that the connection is private by using symmetric data encryption and it ensures that the connection is reliable. The TLS Record Protocol also is used for encapsulation of higher-level protocols, such as the TLS Handshake Protocol. • The TLS Handshake Protocol -- allows authentication between the server and client and the negotiation of an encryption algorithm and cryptographic keys before the application protocol transmits or receives any data. (Source: http://www.webopedia.com/TERM/T/TLS.html)
Trigger		In a DBMS, a trigger is a SQL procedure that initiates (fires) an action when an event (INSERT, DELETE, or UPDATE) occurs. Since triggers are event-driven specialized procedures, the DBMS stores and manages them. A trigger cannot be called or executed; the DBMS automatically fires the trigger as a result of a data modification to the associated table. Triggers maintain the referential integrity of data by changing the data in a systematic fashion.
Triple Data Encryption Algorithm	TDEA	An encryption algorithm whose key consists of three DES (Data Encryption Standard) keys, which is also referred to as a key bundle. A DES key consists of 64 binary digits ("0"s or "1"s) of which 56 bits are randomly generated and used directly by the algorithm. (The other 8 bits, which are not used by the algorithm, may be used for error detection.) Each TDEA encryption/decryption operation (as specified in ANSI X9.52) is a compound operation of DES encryption and decryption operations. Let EK(I) and DK(I) represent the DES encryption and decryption of ***I*** using DES key ***K*** respectively. (Source: http://www.atis.org/tg2k/_triple_data_encryption_algorithm.html)
Trust Point		A trust point is a **Certificate Authority** (**CA**) that is the root of all trust for all CAs in a CA hierarchy.
Tunneling		Transporting IPv6 traffic through IPv4 networks by encapsulating IPv6 packet in IPv4 and vice-versa.
Unicode		A standard defined by the Unicode Consortium. Unicode uses a 16-bit code page that maps digits to characters in languages around the world. Because 16 bits covers 32,768 codes, Unicode is large enough to include all the world's languages, with the exception of ideographic languages that have a different character for every concept, such as Chinese.

		For more information, see http://www.unicode.org/. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Unified Class Library		With the introduction of **.NET**, Microsoft redesigned the access to common system components and services such as XML Web services, Enterprise Services, ADO.NET, and XML by creating a single object-oriented library. All the Microsoft Visual .NET languages (Visual Basic, C++, J#, C#, etc.) have access to this library. To make access to these objects available within the various languages, Microsoft provided infrastructure such as hierarchical namespaces, structures, types, and common objects like collections.
Unified Modeling Language	UML	In the field of software engineering, the Unified Modeling Language (UML) is a standardized specification language for object modeling. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model. UML is officially defined at the **Object Management Group** (OMG) by the UML metamodel, a Meta-Object Facility metamodel (MOF). (Source: http://en.wikipedia.org/wiki/Unified_Modeling_Language; 30 March 2007)
Uniform Resource Identifier	URI	An encoded address that represents any Web resource, such as an **HTML** document, image, video clip, or program. As opposed to a **URL** or a **URN**, which are concrete entities, a URI is an abstract superclass. (Source: http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.wsinted.glossary.doc/topics/glossary.html)
Uniform Resource Locator	URL	A sequence of characters that represents information resources on a computer or in a network such as the Internet. This sequence of characters includes (1) the abbreviated name of the protocol used to access the information resource and (2) the information used by the protocol to locate the information resource.(Source: http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.wsinted.glossary.doc/topics/glossary.html)
Uniform Resource Name	URN	A name that uniquely identifies a **Web service** to a **client**. (Source: http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.wsinted.glossary.doc/topics/glossary.html)
UNIQUE Key Integrity Constraint		A **`UNIQUE`** key integrity constraint requires that every value in a column or set of columns (key) be unique; that is, no two rows of a table have duplicate values in a specified column or set of columns. (Source: http://www.lc.leidenuniv.nl/awcourse/oracle/server.920/a96524/c22integ.htm)
Universal Description, Discovery, and Integration	UDDI	An industry initiative to create a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet, as well as a registry. It is being developed by a vendor consortium. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Use-Case		A sequence of actions, performed by a system, that yields a result of value to a user. A set of actions, including variants,

		that a system performs that yields an observable result of value to a particular actor.
User Datagram Protocol	UDP	A connectionless protocol that, like **TCP**, runs on top of **Internet Protocol** (IP) networks. Unlike **Transmission Control Protocol/Internet Protocol** (TCP/IP), UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagrams over an IP network. It's used primarily for broadcasting messages over a network. (Source: http://www.webopedia.com/TERM/U/User_Datagram_Protocol.html)
VBScript		A programming language developed by Microsoft that is similar to **JavaScript**. It is used to embed code into **HTML** pages. It is actually a subset of Microsoft's Visual Basic.
Vendor		Any person, organization, or automated asset that interfaces with the information environment as a service consumer or service provider.
Very High Speed Integrated Circuit	VHSIC	Specific type of digital logic circuit.
VHDL Component		Special piece of conventional code that allows the construction of hierarchical circuit designs.
VHSIC Hardware Description Language	VHDL	Commonly used design-entry language in the electronic design automation of digital circuits.
VoiceXML	VXML	VoiceXML (VXML) is the **W3C** standard **XML** format for specifying interactive voice dialogues between a human and a computer. It is fully analogous to **HTML**, and brings the same advantages of **Web application** development and deployment to voice applications that HTML brings to visual applications. Just as HTML documents are interpreted by a visual web browser, VoiceXML documents are interpreted by a voice browser. A common architecture is to deploy banks of voice browsers attached to the public switched telephone network (PSTN) so that users can simply pick up a phone to interact with voice applications. VoiceXML has tags that instruct the voice browser to provide speech synthesis, automatic speech recognition, dialog management, and soundfile playback.
Web Application		A collection of components that can be bundled together and run in multiple containers from multiple vendors. -OR- An application written for the Internet, including those built with Java technologies such as Java Server Pages and servlets, and those built with non-Java technologies such as CGI and Perl. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Web Application Archive	WAR	A **JAR** archive that contains a **Web module**. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Web Browser		A client program that initiates requests to a **Web server** and displays the information that the server returns. (Source: http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.wsinted.glossary.doc/topics/glossary.html)

Web Container		A container that implements the Web-component contract of the **J2EE** architecture. This contract specifies a runtime environment for Web components that includes security, concurrency, life-cycle management, transaction, deployment, and other services. A Web container provides the same services as a **JSP** container as well as a federated view of the J2EE platform **APIs**. A Web container is provided by a Web or J2EE server. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Web Module		A deployable unit that consists of one or more Web components, other resources, and a Web application deployment descriptor. The Web module is contained in a hierarchy of directories and files in a standard Web application format. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Web Ontology Language	OWL	A markup language for publishing and sharing data using ontologies on the Internet. (Source: http://en.wikipedia.org/wiki/Web_Ontology_Language)
Web Page		A document created with **HTML** (HyperText Markup Language) that is part of a group of hypertext documents or resources available on the World Wide Web. Collectively, these documents and resources form what is known as a **Web site**. You can read HTML documents that reside somewhere on the Internet or on your local hard drive with software called a **Web browser**. Web pages can contain hypertext links to other places within the same document, to other documents at the same Web site, or to documents at other Web sites.
Web Server		Software that provides services to access the Internet, an intranet, or an extranet. A Web server hosts **Web sites**, provides support for HTTP and other protocols, and executes server-side programs (such as **CGI** scripts or servlets) that perform certain functions. In the **J2EE** architecture, a Web server provides services to a **Web container**. For example, a Web container typically relies on a Web server to provide **HTTP** message handling. The J2EE architecture assumes that a Web container is hosted by a Web server from the same vendor, so it does not specify the contract between these two entities. A Web server can host one or more Web containers. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
Web Service		A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically **WSDL**). Other systems interact with the Web service in a manner prescribed by its description using **SOAP** messages, typically conveyed using **HTTP** with an **XML** serialization in conjunction with other Web-related standards. (Source: http://www.w3.org/TR/ws-gloss/)
Web Services Description Language	WSDL	An XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The

		operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint.
Web Services for Interactive Applications	WSIA	
Web Services for Remote Portlets	WSRP	The WSRP specification defines a **Web service** interface for interacting with interactive presentation-oriented Web services. It has been produced through the joint efforts of the Web Services for Interactive Applications (**WSIA**) and Web Services for Remote Portals (WSRP) OASIS Technical Committees. Scenarios that motivate WSRP/ WSIA functionality include (1) **portal** servers providing **portlets** as presentation-oriented Web services that can be used by aggregation engines; (2) portal servers consuming presentation-oriented Web services provided by portal or non-portal content providers and integrating them into a portal framework. (Source: http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf)
Web Services Interoperability Organization	WS-I	WS-I is an open industry organization chartered to promote Web services interoperability across platforms, operating systems and programming languages. The organization's diverse community of Web services leaders helps customers to develop interoperable Web services by providing guidance, recommended practices and supporting resources. (Source: http://www.ws-i.org/about/Default.aspx)
Web Site		A Web site, website, or WWW site (often shortened to just "site") is a collection of Web pages (i.e., HTML/XHTML documents accessible via **HTTP** on the Internet). All publicly accessible Web sites in existence comprise the World Wide Web. The pages of a Web site are accessed from a common root URL, the homepage, and usually reside on the same physical server. The URLs of the pages organize them into a hierarchy, although the hyperlinks between them control how the reader perceives the overall structure and how the traffic flows between the different parts of the site. (Source: http://en.wikipedia.org/wiki/web_site)
Well-Formed		A textual object is a well-formed **XML document** if: 1. Taken as a whole, it matches the production labeled document. 2. It meets all the well-formedness constraints given in this specification. 3. Each of the parsed entities which is referenced directly or indirectly within the document is well-formed. (source: http://www.w3.org/TR/REC-xml/#dt-wellformed)
Wireless Application Protocol	WAP	WAP is an open international standard for applications that use wireless communication, such as Internet access from a mobile phone. WAP provides services equivalent to a web browser with some mobile-specific additions. It is specifically designed to address the limitations of very small portable

		devices. During its first years of existence WAP suffered from considerable negative media attention and has been criticised heavily for its design choices and limitations. (Source: http://en.wikipedia.org/wiki/WAP)
Wireless Markup Language	WML	WML is the primary content format for devices that implement the **WAP** (Wireless Application Protocol) specification based on XML, such as mobile phones. (Source: http://en.wikipedia.org/wiki/Wireless_Markup_Language)
Wire Protocol		In a network, it is the mechanism for transmitting data from point a. to point b. It often refers to a distributed object protocol such as , or **RMI**, which is software only and which invokes the running of programs on remote servers. (Source: http://www.techweb.com/encyclopedia/defineterm.jhtml?term=wire+protocol)
Wisdom		Knowledge with information so thoroughly assimilated as to have produced sagacity, judgment, and insight. The ability to use knowledge for a purpose.
World Wide Web	WWW	The World Wide Web ("WWW," or simply "Web") is an information space in which items of interest, referred to as resources, are identified by global identifiers called Uniform Resource Identifiers (**URI**). The term is often mistakenly used as a synonym for the **Internet**, but the web is actually a service that operates over the Internet. (Source: http://en.wikipedia.org/wiki/World_Wide_web)
World Wide Web Consortium	W3C	The World Wide Web Consortium (W3C) is an international consortium where Member organizations, a full-time staff, and the public work together to develop Web standards. W3C's mission is to lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web. (Source: http://www.w3.org/Consortium/)
XML Document		A document object that is **well-formed**, according to the XML recommendation, and that might (or might not) be valid. The XML document has a logical structure (composed of declarations, elements, comments, character references, and processing instructions) and a physical structure (composed of entities, starting with the root, or document entity). (Source: http://msdn2.microsoft.com/en-us/library/ms256452.aspx)
XML Gallery		The XML Gallery [of the **DoD Metadata Registry and Clearinghouse**] contains information resources such as submission packages, elements, attributes, and schemas that have been registered by DOD software developers. These information resources use XML, a platform and vendor independent format for exchanging data, to handle data, data structures, and data descriptions (metadata). (Source: http://www.disa.mil/nces/development/developer_doc_overview.html)
XML Information Resources		Document Type Definition (**DTD**) or XML Schema Documents (**XSD**) files.
XML Instance Document		An XML document defined by an XML Schema but is populated with the data, not the definition of the data.

XML Path Language	XPath	The result of an effort to provide a common syntax and semantics for functionality shared between XSL Transformations (XSLT) and XPointer. The primary purpose of XPath is to address parts of an XML document. It also provides basic facilities for manipulation of strings, numbers, and Booleans. XPath uses a compact, non-XML syntax to facilitate use of XPath within URIs and XML attribute values. XPath gets its name from its use of a path notation as used in URLs for navigating through the hierarchical structure of an XML document. (Source: http://msdn2.microsoft.com/en-us/library/ms256452.aspx)
XML Process Definition Language	XPDL	Is the language proposed by the Workflow Management Coalition (WfMC) to interchange process definitions between different workflow products. To goal of XPDL is to provide a Lingua Franca for the workflow domain allowing for the import and export process definitions between a variety of tools ranging from workflow management systems to modeling and simulation tools.
XML Schema		A database-inspired method for specifying constraints on documents using an XML-based language. Schemas address deficiencies in **DTDs**, such as the inability to constrain the kinds of data that can occur in a particular field. Because schemas are founded on XML, they are hierarchical. Thus it is easier to create an unambiguous specification, and it is possible to determine the scope over which a comment is meant to apply. (Source: http://java.sun.com/j2ee/1.4/docs/glossary.html)
XML Schema Definition	XSD	A language proposed by the **W3C** XML Schema Working Group for use in defining schemas. Schemas are useful for enforcing structure and/or constraining the types of data that can be used validly within other XML documents. XML Schema Definition refers to the fully specified and currently recommended standard for use in authoring XML schemas. Because the XSD specification was only recently finalized, support for it was only made available with the release of MSXML 4.0. It carries out the same basic tasks as DTD, but with more power and flexibility. Unlike DTD, which requires its own language and syntax, XSD uses XML syntax for its language. XSD closely resembles and extends the capabilities of XDR. Unlike XDR, which was implemented and made available by Microsoft in MSXML 2.0 and later releases, the W3C now recommends the use of XSD as a standard for defining XML schemas. (Source: http://msdn2.microsoft.com/en-us/library/ms256452.aspx)

References

R1001	DOM – a Whatis.com definition - http://whatis.techtarget.com/definition/0,,sid9_gci213910,00.html.
R1002	**SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
R1003	Web Service Definition Language (WSDL) - http://www.w3.org/TR/wsdl
R1004	XSD - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0%2c%2csid26_gci831325%2c00.html
R1007	XSL - definition of XSL in Encyclopedia - http://encyclopedia.laborlawtalk.com/XSL
R1008	Web Services Security Specification, March 2004, (http://www.oasis-open.org/specs/index.php)
R1009	An Introduction to the Web Services Architecture and Its Specifications, October 2004, (http://msdn.microsoft.com/webservices)
R1010	XML (http://en.wikipedia.org/wiki/Xml)
R1011	ActiveX Control definition - http://isp.webopedia.com/TERM/A/ActiveX_control.html
R1012	Component Object Model definition - http://isp.webopedia.com/TERM/C/Component_Object_Model.html
R1013	JScript definition - http://isp.webopedia.com/TERM/J/JScript.html
R1014	IIS - definition of IIS in Encyclopedia - http://encyclopedia.laborlawtalk.com/IIS
R1015	Personal Web Server - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci296469,00.html
R1016	HTML - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212286,00.html
R1017	VBScript definition from The Glossary of Internet Terms - http://www.strategicwebventures.com/definitions/Glossary/VBScript/
R1018	HTML - a Whatis.com definition - http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212286,00.html
R1019	What is JSP? - http://www.webopedia.com/TERM/J/JSP.html
R1020	Net-Centric Checklist, July 30, 2004, Version 2.1.4
R1021	ANT - http://ant.apache.org/
R1022	Java EE - http://java.sun.com/javaee
R1023	For answers to frequently asked questions about **cascading style sheets**, see http://www.blooberry.com/indexdot/css/topics/stylefaq.htm

R1024	Don't Make Me Think by Steve Krug (ISBN 0-7897-2310-7)
R1025	Creating Killer Interactive Web Sites by Adjacency (ISBN 1-56830-373-4)
R1026	Designing Web Usability by Jakob Nielsen (ISBN 1-56205-810-X)
R1027	OMG - http://www.omg.org/gettingstarted/gettingstartedindex.htm
R1029	**SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
R1030	Web Service Definition Language (WSDL) - http://www.w3.org/TR/wsdl
R1031	Adapter pattern - http://c2.com/cgi/wiki?AdapterPattern
R1032	Design patterns: Proxy - http://www.dofactory.com/Patterns/PatternProxy.aspx
R1033	Façade pattern - http://c2.com/cgi/wiki?FacadePattern
R1034	Java EE - http://java.sun.com/javaee/
R1035	EJB - http://java.sun.com/products/ejb/
R1036	.jar - http://java.sun.com/developer/Books/javaprogramming/JAR/
R1037	.war - http://access1.sun.com/techarticles/simple.WAR.html
R1038	.ear - http://java.sun.com/J2EE/tutorial/1_3-fcs/doc/Overview4.html
R1039	.rar - http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Connector2.html
R1040	JavaBeans definition - http://isp.webopedia.com/TERM/J/JavaBeans.html
R1041	Model-view-controller - a Whatis.com definition - http://whatis.techtarget.com/definition/0,,sid9_gci214607,00.html
R1042	Java Servlets definition - http://www.fromallangles.com/glossary/web-hosting/terms/java-servlets.htm
R1043	Java Naming and Directory Interface - http://java.sun.com/products/jndi/
R1044	Erdogan, Levent. "Java Message Service (JMS) for J2EE." New Riders Publishing, August 14, 2002.
R1045	See the Microsoft MSDN document .NET Compact Framework for detailed comparisons between the **.NET Compact Framework** and the **.NET** Framework.
R1046	"The Semantic Web," Michael C, Daconata, Leo J. Obrst, Kevin T. Smith; Wiley Publishing Inc., 2003
R1047	http://members.optusnet.com.au/~webindexing/Webbook2Ed/glossary.htm
R1048	http://www.w3.org/TR/2004/REC-owl-features-20040210/
R1049	http://www.w3.org/TR/2002/NOTE-wscl10-20020314/
R1050	Wikipedia

R1051	DoD Meta Data Registry for XSLTsamples.[http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal]
R1052	XSLT 1.0 (Extensible Stylesheet Language Transformations), for transforming XML documents [http://www.w3.org/TR/xslt]
R1053	XSLT 2.0 (W3C Working Draft, 5 November 2004) [http://www.w3.org/TR/xslt20]
R1054	XSL Transformations (XSLT) Version 2.0
R1055	XSL (Extensible Stylesheet Language) 1.0, presentation rules to transform a document. [See http://www.w3.org/Style/XSL and http://www.w3.org/TR/xsl]
R1056	The Extensible Stylesheet Language Family http://www.w3.org/Style/XSL
R1058	CSS (Cascading Style Sheets) versions 1 (CSS1) and 2 (CSS2) [See http://www.w3.org/Style/CSS, http://www.w3.org/TR/REC-CSS1, http://www.w3.org/TR/REC-CSS2]
R1062	Won Kim. Introduction to Object-Oriented Databases. Computer Systems. MIT Press, Cambridge, MA, 1990.
R1069	Native XML database vendors: http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native
R1070	**C2IEDM data model** specifications: http://www.mip-site.org/
R1071	Various methodologies, such as **refactoring**, support the determination of **interfaces**. **Refactoring** generally applies to the entire software implementation, but is especially helpful in properly flushing out **interfaces**.
R1072	There are other approaches defined through the pattern community: http://hillside.net/patterns/.
R1073	See chapter 3 of Java Design: Building Better Apps &
R1074	For information about tags that might appear in future releases of Java, see http://java.sun.com/products/jdk/javadoc/proposed-tags.html.
R1075	For Microsoft Visual Studio 6 and earlier, use the open-source tools described in C/C++ Javadoc guidelines.
R1076	For Microsoft Visual Studio .NET 2002 and above, use the XML documentation tools that come with the .NET IDE Integrated Development Environment.
R1077	AWT - http://java.sun.com/products/jdk/awt/
R1078	Swing - http://java.sun.com/products/jfc/
R1079	Thick client - http://www.jargonsoft.com/m2/tech/JargonWhitePaper.html
R1080	OASIS – a Whatis.com definition - http://searchvb.techtarget.com/gDefinition/0%2c294236%2csid8_gci527425%2c00.html
R1081	What is **UDDI**? - http://www.webopedia.com/TERM/U/UDDI.html
R1091	"Web Portal Design Guide 1.1 July 2006," Space and Naval Warfare Systems Center San Diego

R1092	For information on WSRP access specifications, see http://www-106.ibm.com/developerworks/webservices/library/ws-wsrp/
R1093	For information on JSR-168 access specifications, see http://www.jcp.org/aboutJava/communityprocess/final/jsr168/
R1094	For information on portlets, JSR-168, and WSRP access, see http://www.portletsfactory.com/resources/portlets-jsr168-wsrp-portals-books.html
R1095	The Navy Enterprise Application Development Guide (NEADG) provides developer's guidance for the Task Force Web (TFW) Navy Enterprise Portal (NEP).
R1097	The Air Force Portal is built upon the BroadVision portal (http://www.broadvision.com/). For specifics on this portal, refer to http://www.gcss-af.com/.
R1098	Other Air Force portal initiatives are based on the Plumtree portal (http://www.plumtree.com/).
R1099	**SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
R1100	Web Service Definition Language (**WSDL**) - http://www.w3.org/TR/wsdl
R1101	Adapter pattern - http://c2.com/cgi/wiki?AdapterPattern
R1102	Design patterns: Proxy - http://www.dofactory.com/Patterns/PatternProxy.aspx
R1103	Façade pattern - http://c2.com/cgi/wiki?FacadePattern
R1104	For information on .NET vs. J2EE Web services, see http://www.webservicesarchitect.com/content/articles/hanson01.asp.
R1105	**SOAP** definition - http://sbc.webopedia.com/TERM/S/SOAP.html
R1108	Software Communications Architecture
R1109	Minimum CORBA v1.0
R1110	OMG Lightweight Log Service
R1111	Software-based Communications DTF
R1112	Software Communications Architecture (Wikipedia)
R1113	"Circuit Design with VHDL" by Volnei A. Pedroni, MIT Press, 2004. ISBN: 0-262-16224-5
R1114	"VHDL Coding Styles and Methodologies" (2nd Edition) by Ben Cohen, Kluwer Academic Publishers, 1999. ISBN: 0-7923-8474-1
R1115	"JTRS JPEO Software Standards (Version 1.0)," SPAWAR Systems Center San Diego, 2006.
R1116	W3C Extensible Markup Language (XML)
R1117	http://xfront.com/BestPracticesHomepage.html
R1118	Microsoft Standards Reference - http://msdn2.microsoft.com/en-us/library/ms256177.aspx

R1119	Component Organization and Registration Environment - https://www.collab.core.gov/CommunityBrowser.aspx?id=2234
R1120	Federal XML Naming and Design Rules - http://xml.coverpages.org/Federal-NDR-20050609.pdf
R1121	W3C Extensible Markup Language (XML) 1.0 (Fourth Edition) - http://www.w3.org/TR/2006/REC-xml-20060816/
R1122	W3 Schools XML Syntax Rules Tutorial - http://www.w3schools.com/xml/xml_syntax.asp
R1123	W3 Schools XML Validator Tutorial - http://www.w3schools.com/xml/xml_validator.asp
R1124	http://www.w3.org/2001/XMLSchema.xsd
R1125	http://www.xfront.com/xml-schema.html
R1126	http://www.xfront.com
R1127	http://www.xfront.com/ZeroOneOrManyNamespaces.pdf
R1128	http://www.xfront.com/DefaultNamespace.pdf
R1129	Extensible Content Models - http://www.xfront.com/ExtensibleContentModels.pdf
R1130	Element versus Type - http://www.xfront.com/ElementVersusType.pdf
R1131	XML Schema Part 2: Datatypes Second Edition - http://www.w3.org/TR/xmlschema-2/#built-in-datatypes
R1132	Composition versus Subclassing - http://www.xfront.com/composition-versus-subclassing.html
R1133	http://www.w3.org/TR/xpath
R1134	http://www.w3schools.com/xpath/default.asp
R1135	XPath (Wikipedia) - http://en.wikipedia.org/wiki/XPath
R1136	HOWTO: Write Namespace-Agnostic XPath and XSLT - http://jcooney.net/archive/2005/08/09/6517.aspx
R1137	http://www.w3.org/DOM/
R1138	http://www.saxproject.org/
R1139	Department of Defense Instruction 5000.2, "Operation of the Defense Acquisition System," Section 3.7 and Enclosure 7
R1140	Human Engineering MIL-STD 1472F, section 5.14 "User Computer Interface"
R1141	"Guide for developing usable and useful web sites" [http://usability.gov]
R1142	"Microsoft Windows User Experience: Official Guidelines for User Interface Developers and Designers," Redmond, WA: Microsoft Press, 1999
R1143	"Apple Human Interface Guidelines," Apple Computer, Inc., 2004

	[http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/index.html]
R1144	"Seven Common Mistakes in Designing a Usable Portal," Iozzo, N., 2002 [http://www.tandemseven.com/pdf/T7_Seven_Common_Mistakes.pdf]
R1145	"The GNOME Usability Project. GNOME Human Interface Guidelines (1.0)," [http://developer.gnome.org/projects/gup/hig/1.0/]
R1146	"Java Look and Feel Design Guidelines: Advanced Topics," Sun Microsystems, Inc., 2001
R1147	"Java Look and Feel Design Guidelines. Second Edition," Sun Microsystems, Inc., 2001
R1148	"Designing Interfaces: Patterns for Effective Interaction Design," Tidwell, J., O'Reilly Media, Inc., 2006
R1149	"Common Presentation Layer Guide Standard 03-01," NAVSEA, September 2006
R1150	"C++ Coding Standards, 101 Rules, Guidelines and Best Practices," Herb Sutter and Andrei Alexandrescu, Addison-Wesley, 2004. ISBN: 0-321-11358-6
R1151	"Web-Based Portal Computer-Human Interface Guidelines," Ahlstrom, V. & Allendoerfer, K., 2004 [http://hf.tc.faa.gov/products/bibliographic/tn0423.htm]
R1152	"Web Application Design Handbook: Best Practices for Web-Based Software," Fowler, S. & Stanwick, V., San Francisco: Morgan Kaufmann Publishers, 2004.
R1154	"Federal IT Accessibility Initiative," [http://www.section508.gov/]
R1155	"Electronic and Information Technology Accessibility Standards," Federal Register, [http://www.access-board.gov/sec508/508standards.pdf]
R1156	"Web Content Accessibility Guidelines 1.0," W3C, [http://www.w3.org/TR/WAI-WEBCONTENT/]
R1157	"Guidelines for Keyboard User Interface Design," Microsoft Corporation [http://msdn.microsoft.com/library/?url=/library/en-us/dnacc/html/ATG_KeyboardShortcuts.asp]
R1159	"Internationalization Best Practices: Specifying Language in XHTML & HTML Content," W3C, [http://www.w3.org/TR/i18n-html-tech-lang/]
R1160	"Internationalization Quick Tips for the Web," W3C [http://www.w3.org/International/quicktips/]
R1161	"Developing and Localizing International Software," Madell, T., Parsons, C. & Abegg, J., Englewood Cliffs, NJ: Prentice Hall, 1994
R1162	"Programming for the World: A Guide to Internationalization," O'Donnell, S.M., Englewood Cliffs, NJ: Prentice Hall, 1994
R1163	"Software Internationalization and Localization: An Introduction," Uren, E., Howard, R. & Perinotti,T., New York: Van Nostrand Reinhold, 1993
R1164	DoD Directive 5000.1, *The Defense Acquisition System*, 12 May 2003 (certified current as of 24 November 2003); http://www.dtic.mil/whs/directives/corres/pdf/500001p.pdf.

R1165	DoD Instruction 5000.2, *Operation of the Defense Acquisition System*, 12 May 2003; http://www.dtic.mil/whs/directives/corres/pdf/500002p.pdf.
R1166	DoD Directive 8100.1, *Global Information Grid (GIG) Overarching Policy*, 19 September 2002 (certified current as of 21 November 2003); http://www.dtic.mil/whs/directives/corres/pdf/810001p.pdf.
R1167	DoD Directive 4630.05, *Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 05 May 2004 (certified current as of 23 April 2007); http://www.dtic.mil/whs/directives/corres/pdf/463005p.pdf.
R1168	DoD Instruction 4630.8, *Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS)*, 30 June 2004; http://www.dtic.mil/whs/directives/corres/pdf/463008p.pdf.
R1170	*DoD Global Information Grid (GIG) Architecture*, Version 2.0, August 2003.
R1171	*DoD Architecture Framework (DoDAF)*, Version 1.5, 23 April 2007; https://dars1.army.mil/IER/index.jsp
R1172	*DoD Net-Centric Data Strategy*, DoD Chief Information Officer, 9 May 2003, http://www.defenselink.mil/cio-nii/docs/Net-Centric-Data-Strategy-2003-05-092.pdf.
R1173	CJCSI 3170.01F, *Joint Capabilities Integration and Development System*, 01 May 2007; http://www.dtic.mil/cjcs_directives/cdata/unlimit/3170_01new.pdf.
R1174	CJCSM 3170.01C, *Operation of the Joint Capabilities Integration and Development System*, 01 May 2007; http://www.dtic.mil/cjcs_directives/cdata/unlimit/m317001.pdf.
R1175	CJCSI 6212.01D, *Interoperability and Supportability of Information Technology and National Security Systems*, 8 March 2006; http://www.dtic.mil/cjcs_directives/cdata/unlimit/6212_01.pdf.
R1176	*Net-Centric Operations and Warfare Reference Model (NCOW RM)*, v1.1, 17 November 2005.
R1177	*Net-Centric Checklist*, V2.1.3, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 12 May 2004; http://www.defenselink.mil/cio-nii/docs/NetCentric_Checklist_v2-1-3_.pdf.
R1178	*A Modular Open Systems Approach (MOSA) to Acquisition*, Version 2.0, September 2004; http://www.acq.osd.mil/osjtf/mosapart.html.
R1179	*DoD IT Standards Registry* (*DISR*); http://disronline.disa.mil.
R1180	*Net-Centric Attributes List*, Office of the Assistant Secretary of Defense for Networks and Information Integration/Department of Defense Chief Information Officer, 2 February 2007; http://www.defenselink.mil/cio-nii/docs/NetCentricAttributesOfficial.pdf .
R1181	*Global Information Grid (GIG) Key Interface Profiles (KIPs) Framework* (DRAFT), Version 0.95, 7 October 2005.
R1202	**OMG** Data Distribution Service for Real-time Systems Version 1.2
R1203	**OMG** Data Distribution Portal (http://portals.omg.org/dds)
R1206	DoD Instruction 8520.2; 1 April 2004; *Public Key Infrastructure (PKI) and Public Key (PK) Enabling*