|  |
| --- |
| SBM |
| AlMajdouie Naming and Deployment Standards |
|  |

|  |
| --- |
| Marwan Zakhia  4/12/2015 |

Table of Contents

[1 Document Control 3](#_Toc418493626)

[1.1 Change Record 3](#_Toc418493627)

[1.2 Approval 3](#_Toc418493628)

[1.3 Distribution 3](#_Toc418493629)

[2 Introduction 4](#_Toc418493630)

[2.1 Scope 4](#_Toc418493631)

[2.2 Intended Audience 5](#_Toc418493632)

[2.3 Document Summary 5](#_Toc418493633)

[3 Artifact Naming Standards 6](#_Toc418493634)

[3.1 Overview 6](#_Toc418493635)

[3.2 Naming Principles 7](#_Toc418493636)

[3.3 Domain Partitioning 9](#_Toc418493637)

[3.4 Component Naming Standards 12](#_Toc418493638)

[3.5 File Naming Principles 14](#_Toc418493639)

[4 XML Artifact Names 15](#_Toc418493640)

[4.1 Namespaces 15](#_Toc418493641)

[4.2 Enterprise Business Object (EBO) 16](#_Toc418493642)

[4.3 Service WSDL 17](#_Toc418493643)

[4.4 Event EDL 19](#_Toc418493644)

[5 Capability Naming Standards 21](#_Toc418493645)

[5.1 Source Code Management 21](#_Toc418493646)

[5.2 Core MDS Project Structure 22](#_Toc418493647)

[5.3 Capability Project Structure 24](#_Toc418493648)

[5.4 Capability MDS Structure 25](#_Toc418493649)

[5.5 OSB Capability Structure 28](#_Toc418493650)

[5.6 SCA Capability Structure 33](#_Toc418493651)

[5.7 SCA Composite 35](#_Toc418493652)

[6 Fault Standards 38](#_Toc418493653)

[6.1 Fault Codes 38](#_Toc418493654)

[7 Deployment Standards 39](#_Toc418493655)

[7.1 SCA Capability Deployment 39](#_Toc418493656)

[7.2 OSB Deployment 40](#_Toc418493657)

Table of Figures

[Figure 3.1 – Service Paritioning Model 10](#_Toc354466979)

[Figure 3.2 – Illustrated Service Partitioning Model 11](#_Toc354466980)

# Document Control

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
|  | Marwan Zakhia | 0.1 | Initial revision |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Approval

|  |  |
| --- | --- |
| Name | Position |
|  |  |
|  |  |
|  |  |

## Distribution

|  |  |
| --- | --- |
| Name | Position |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Introduction

## Scope

A SOA solution implemented on the Oracle SOA Suite will be made up of a number of components, each composed of a number of artifacts, such as XML Schemas, WSDL Definitions, BPEL Processes, XSLT or XQuery Transformations, OSB Proxy Service and OSB Business Services.

The AlMajdouie SOA Platform for the gateway payment (SADAD) project is primarily an OSB platform which will host multiple SOA integration solutions that will need to co-exist with one another.

In such an environment naming and deployment standards are critically important, for a number of reasons, including:

* Avoiding duplicate component names, resulting in one component overwriting another component.
* Provides a consistent approach to service versioning
* Simplifies automation of the build and deployment process
* Simplifies the monitoring and administration of components

Clear, concise and consistent naming standards allow names to convey meaning. This simplifies and reduces the overhead of maintaining components. In addition, within a SOA environment, the consistent naming of artifacts makes it easier to identify potential re-use opportunities.

This section of the SOA Reference Architecture defines the naming and deployment standards for artifacts developed for and deployed to the Oracle Service Bus 11gR1.

### Out of Scope

The bottom layer of the AlMajdouie SOA Application Architecture, namely the Application Services that represent applications or business logic hosted externally to the AlMajdouie SOA Infrastructure, are typically provided by existing applications. These applications include:

* Commercial off the shelf (COTS) application such as Oracle e-Business Suite, Siebel, PeopleSoft, or SAP.
* Be-Spoke Applications developed in-house using technologies such as Java, C#, Oracle Forms, PL/SQL, etc.
* External Web Services provided by other 3rd parties, e.g. services provided by partners, suppliers, customers, etc… as well as software as a service (SaaS).

It should be noted that this document only covers artifacts that are implemented on the Oracle SOA platform itself and thus the naming of Application Services is outside the scope of this document.

This document does not provide guidance on the construction of names used in WSDL or XML Schema, or on best practices pertaining to these.

## Intended Audience

The intended audience for this document are:

* System Architects - Provides standards, guidelines, best practice and patterns for ensuring the consistent architecture and implementation of SOA solutions deployed to the AlMajdouie SOA Platform
* Developers - Provides standards, guidelines, best practice and patterns for ensuring the consistent architecture and implementation of SOA solutions deployed to the AlMajdouie SOA Platform

## Document Summary

This document covers the following areas:

* Artifact Naming Standards
* XML Artifact Names
* Capability Naming Standards
* Deployment Standards

# Artifact Naming Standards

## Overview

The clear, concise and consistent naming of artifacts results in more readable code, as a good name conveys meaning as to the functional purpose of an artifact, its role in a collection of related artifacts (that address the same functional purpose) and any patterns that has been implemented.

This leads to code that is more “readable”; resulting in higher quality code (as its easier to visually spot defects) and code that is easier to maintain.

In addition, within a SOA environment, consistent naming and categorization of a service, simplifies the identification of similar service requirements and potential re-use opportunities.

Another consideration for AlMajdouie, is that the Oracle FMW SOA Platform is a shared platform, which will host multiple SOA Solutions, each of which may have been developed independently with possibly minimal (if any) knowledge of the other solutions to deployed to the platform. Each of these SOA Solutions needs to be implemented and deployed in such a fashion that it can co-exist with other current and future solutions.

In such an environment, standards which govern the naming and deployment of SOA artifacts are critically important, for a variety of reasons, including:

* Firstly, it is important to ensure each component deployed to the SOA platform has a unique name, to avoid the situation where two projects independently create a component with the same name.  
  For example Project A creates a component called Foo and deploys it to the AlMajdouie SOA Platform; Project B then creates a different component also called Foo and deploys it the AlMajdouie SOA Platform and in the process over writes Project A’s component.
* Secondly, it is important to ensure that a component is only deployed once[[1]](#footnote-1) to the AlMajdouie SOA Platform. This is particularly the case for EBO’s which could define part of the AlMajdouie Canonical Model.  
    
  For example, if each component deployed its own copy of an EBO, then apart from the maintenance overhead that this would create, it is quite feasible for the different copies to get out of sync, a common cause of run-time issues on an Oracle SOA Platform.

Another important consideration is **configuration** information which is specific to the environment in which a component is deployed. For example, the end point of an invoked web services, fault policies for a composite and notification emails.

Due to the nature of the shared platform, any SOA solution will typically need to be promoted through a variety of platforms, such as SIT, UAT, Pre-Prod before being allowed into production.

If we have to modify the code, in order to “configure” it for the new environment, we introduce the possibility of error as well as invalidating any previously conducted test.

So it is important that the naming and deployment standards clearly separate out configuration information from the code.

A SOA solution implemented on the Oracle SOA Suite will be made up of a number of SOA components, each composed of a number of artifacts, such as XML Schemas, WSDL Definitions, BPEL Processes, XSLT or XQuery Transformations, OSB Proxy Service, OSB Business Services, etc.

Each artifact, consists of public parts that are visible to other components, for example WSDL files, SCA Composite and OSB Proxy, as well as private parts, that are only visible within the artifact; for example the names of a variable, an activity within a BPEL process or an Action within an OSB Proxy. These standards are only concerned with the public parts of an artifact.

## Naming Principles

Any name must be somewhat comprehensible by people unaware of the naming standard. That is, at first glance, the name should give some idea about what it actually represents.

The use of abbreviations (as detailed in this document) should be used if it makes the meaning clearer and more succinct. The primary rule here is it needs to be understandable more than it does short (although too long would decrease readability quickly also).

Some examples of good names are:

* **CustomerParty**  
  This is clearly about describing customers. We could also have used CustParty but this abbreviation is not really required since the full form is clearer and not too long
* **getCustomerParty**  
  This is clearly also about customers. Just by looking it one can see from the name it’s an operation because it starts with a verb, and for some input it will get us a customer party.
* **PIN**  
  This is a good example of a compound name that is abbreviated (see rules below). PIN is a well know abbreviation within banking so the meaning is clear and the alternative of PersonalIdentificationNumber would convey no additional information nor be any clearer than the abbreviated form.
* **getCustomerPIN**Just as detailed above, can tell this operation will return a customer personal identification number for a given customer.

If you are ever in doubt about a name and in the absence of clear guidance from the reference architecture you should ask colleagues about the name. If they can clearly state the meaning with nothing but the name it’s likely to be a good name.

* + 1. **Nouns and Verbs**

The use of nouns and verbs in the naming of elements and artifacts is very important so for completeness their definition and examples are included here.

* **Noun**  
  A noun is a naming word; they’re used to name things. The type of noun used in naming standards is a common noun, which names classes of objects. Examples of common nouns are Customer, Book, Car, and Plane etc.
* **Verb**A verb describes an action or state. Examples of a verb are the CRUD operations, create, read, update, and delete.
  + 1. **Context in Names**

The use of inferred context when naming is *NOT* allowed. An example of the use of inferred context might be a web service with the name CustomerParty and providing an operation called get.

Whilst the meaning can be inferred as getCustomerParty the primary principle of the naming standards is to make meaning easy, clear and unambiguous.

The *ONLY* exception to this is the naming of XML elements as detailed in the Canonical Data Model (CDM) Standards; here the context can be directly inferred by an element’s lineage. This is in line with best practice and leads to well structured, easily parsed and queried XML documents without losing readability.

* + 1. **Abbreviations**

Abbreviations should be used in compound names in order to minimize name length, as long as this doesn’t reduce readability. When used on its own the full name should be used.

For example we may use address by itself, but when combined with another word (e.g. shipping), we use its abbreviation to get shippingAddr.

#### Note

The following guidelines are defined for the cases where no common abbreviation is defined; in these cases abbreviate to the first syllable of each word.

For example reference can be abbreviated to ref, aggregation to agg.

Where the abbreviation could be confused for a common word, then we should include the first letter after the first syllable. For example instead of abbreviating address to add, we abbreviate it to addr.

## Domain Partitioning

The service domain is partitioned to align part of the service landscape with appropriate enterprise model. This alignment provides an integral part of the service and artifact naming standards, as detailed below.

The following diagram *illustrates* the Service Partitioning Model for AlMajdouie; however the reader should consult both the AlMajdouie SOA Architecture Blueprint and the relevant enterprise model for a full and current version.

Figure 3.1 – Service Paritioning Model

From this we can see the service partitioning model has at least four tiers, namely:

* **Domain**; defines the realm of administration over the resource
* **Reference Model**; defines the reference model that the service aligns with.
* **Sub-domain**; a logical sub-division of the reference model or the parent sub-domain with core business ability, for example *Deliver Services*.

**Capability**; collection of components that provide related re-usable services, for example *Manage Customer Relations*. Capabilities can reside at any level of the model below the *Reference Model* as described below.

The following is an example of the service partitioning model, with some details completed for illustrative purposes.

Figure 3.2 – Illustrated Service Partitioning Model

These are detailed further below.

* + 1. **Domain**

The domain is a Uniform Resource Locator (URL) based on AlMajdouie’s domain name to make it unique and managed. It acts as an identification label that defines the realm of administration over the resource; i.e. it ensures that no other resource with the same qualified name will be created.

Any service created by AlMajdouie must be created under the domain **http://www.almajdouie.sa**.

* + 1. **Reference Model**

Defines the reference model that the service aligns with, this can be one of:

* Application Reference Model (ARM)
* Business Reference Model (BRM)
* Information Reference Model (IRM)
* Product Reference Model (PRM)
* Technical Reference Model (TRM)

See the AlMajdouie SOA Architecture Blueprint for details if you’re unsure of this concept.

* + 1. **Sub-Domain**

The sub-domain represents a logical sub-division of the reference model or the parent sub-domain. The reference model concerned defines the exact name of the sub-domain. Some abbreviations might be used.

Note: The number of sub-domains depends on the depth of the reference model. The use of the term sub-domain here is a grouping term; the actual term depends, also, on the reference model.

* + 1. **Capability**

A capability is a collection of components that provide one or more related re-usable services; made available as a collection of External Virtual Services. A key design driver for a capability is that it is designed for re-use (rather than agility), so that it can be easily leverage by multiple consumers (including other capabilities and DBAs)

From a lifecycle management perspective, the unit of deployment is a capability.

* + 1. **Capability Name**

The reference model, sub-domains, and capability names form key components in the name of any artifact, or part of any artifact which is visible any other artifact; as defined in the remainder of this document.

To provide both clarity and consistency, a standard short name is defined for each sub-domain in the reference models and the AlMajdouie XML Dictionary should be consulted and complied with for these. Within the naming standards the capability name is represented as:

<reference model>.<sub-domain>+.<capability>

The <sub-domain>+ represents the naming of the parent sub-domains, as there may (and often will) be more than one. For the sake of brevity, within the remainder of this document, we will use <capability> to represent the details above.

For example, the naming standard for the abstract WSDL of an EBS is:

<capability>.<compType>.<compName>\_<version>.wsdl

For example, a Financial Transaction EBS is part of the AlMajdouie reference model, and *acc* and *Transaction* sub-domain of this model, it would be named as follows:

pp.acc.Transaction.ebs.FinancialTransaction\_1.0.wsdl

## Component Naming Standards

The consistent naming of an artifact will ensure that purpose of that artifact is easier to comprehend, as well as reducing the likelihood of duplication (i.e. similar versions of a service being created independently). Within the AlMajdouie SOA Architecture the name of a component (e.g. an EBO, Service or Event), is made up of three segments, these are:

* Component Type
* Component Name

Component Version

Each of these is defined below.

* + 1. **Component Type**

To provide both clarity and consistency, a standard short name (or abbreviation) is defined for each component type as defined in the following table:

| Component Type | Abbreviation |
| --- | --- |
| Enterprise Business Object | ebo |
| External Business Service | ebs |
| External Virtual Event | eve |
| Internal Virtual Service | ivs |
| Internal Virtual Event | ive |
| Client Virtual Service | cvs |
| Business Process | bp |
| Task Service | task |
| Entity Service | ent |
| Functional Service | func |
| Internal Event Channel | iec |
| Event Publisher | Ep |
| Application business connector service | abcs |

Within the remainder of this document, we will use <compType> to represent the standard short name of a component type. For example, the naming standard for the abstract WSDL of an EBS is:

<capability>.<compType>.<compName>\_<version>.wsdl

Following this, we can see the Financial Transaction EBS, as above, would be named as follows:

pp.acc.Transaction.ebs.FinancialTransaction\_1.0.wsdl

* + 1. **Component Name**

The component name must be somewhat comprehensible by people unaware of the naming standard. That is, at first glance, the name should give some idea about the functional purpose of the component.

Within the remainder of this document, we will use <compName> to represent the name of a component. For example, the naming standard for the abstract WSDL of an EBS is:

<capability>.<compType>.<compName>\_<version>.wsdl

Following this, we can see the Financial Transaction EBS, as above, would be named as follows:

pp.acc.Transaction.ebs.FinancialTransaction\_1.0.wsdl

* + 1. **Component Version**

When we modify a service, e.g. from version 1 to version 2, then from the perspective of the consumer there are two possible outcomes. Either the version 1 consumer can continue to successfully use version 2 of the service, in which case the service is said to be backward compatible or the change will break the existing contract.

To be explicit, a service is said to be backwards compatible if ALL requests that would have been successfully processed by version 1 of the service, will be successfully processed by version 2 of the service.

The SOA Reference Architecture defines three levels of versioning, these are:

* Major; signifies a change that is incompatible with previous deployment of the service. Major changes typically indicate significant new features or major revisions to the existing services.
* Minor; signifies a change that is backward compatible with previous versions of the service that share the same major number. These types of changes typically extend the existing component. For example, the addition of a new operation to a service.
* Patch; signifies a change that is intended to fix one or more bugs in a previous minor release. It may also include small new features which don’t modify the service contract.

Major and minor version numbers MUST be included in the artifact name as:

<major>.<minor>

Note: The patch version is not included in the artifact name

For the sake of brevity, within the remainder of this document, we will use <version> to represent <major>.<minor>

For example, the naming standard for the abstract WSDL of an EBS is:

<capability>.<compType>.<compName>\_<version>.wsdl

Following this, we can see the WSDL for version 1.0 of the Notification services which is part of the communication capability in the utility competency would be name as follows:

util.comm.ebs.Notification\_1.0.wsdl

## File Naming Principles

File names are generally named in a flat structure, encoding the service type and business capability partitioning directly into the name of the file. For example naming a file:

util.comm.ivs.Notification\_1.0.wsdl

Instead of:

util/comm/ivs/Notification\_1.0.wsdl

This provides a number of advantages, including:

* All references within xml artifacts are relative, so they don't have to explicitly reference them via oramds, this means the references will work in MDS and OSB as well as other tools, such as XMLSpy.
* Simplifies deploying the artifacts to MDS and OSB in a consistent fashion, as the structure is mainly in the file name
* By having the structure in the file name, it’s simpler to identify an artifact when moved out of context (e.g. you attach an ebo schema in an email).

For example aaa.bbb.foo is clearly different from xxx.yyy.foo.

* We use svn:externals in quite a few places to ensure we only have a single version of a schema shared across mds and osb, by having a flat file structure simplifies this, as well as improves svn performance.

# XML Artifact Names

This section details the naming standards for the following XML Artifacts:

* Enterprise Business Objects (EBO)
* Service WSDL’s
* Event EDL’s

## Namespaces

AlMajdouie has made the choice to use Uniform Resource Locators (URLs) as namespaces for XML resources constructed by AlMajdouie.

For AlMajdouie’s use the general format of the URL MUST be as follows:

http://www.almajdouie.com/<use-specific>

The use-specific section is detailed in the following sections.

The **WHOLE** URL **MUST** be unique within AlMajdouie, and to facilitate this the URL must be registered with the Integration Competency Centre.

* + 1. **Artifact Namespaces**

***ALL*** AlMajdouie authored artifacts must be in the namespace as defined by the following definition:

http://www.almajdouie.com/<CompType>/<CompName>

The <CompType*>* must be one of the approved abbreviations from section 3.4.1, but must be in lower case. The <CompName> must as described in section 3.4.2 to get the partitioning combined with the artifact short name.

The following table shows some examples of the namespaces:

| Artifact Type | Namespace Definition |
| --- | --- |
| External BusinessService | http://www.almajdouie.com/ebs/pp.acc.Transaction.FinancialTransaction |
| Internal Virtual Service | http://www.almajdouie.com/ivs/pp.acc.Transaction.FinancialTransaction |

Table 3.1 – Example Artifact Namespaces

* + 1. **Namespace Prefix**

One might argue the definition of a naming scheme for name space prefixes is moot, given that as long as it maps to the correct namespace the prefix could be anything. Keeping the name space prefix consistent across schemas (and where possible – XML documents, but this is not possible most of the time) increases readability.

If I see a prefix *xsd* there is a very good chance this maps to the XML schema namespace.

A namespace prefix should confirm to the following:

* Be no more than 3-6 letters in length
* As representative as possible to the schema within the above constraint
* Not just for example, ba1, ba2, ba3, etc…

Must NOT start with *xml*

## Enterprise Business Object (EBO)

Enterprise Business Objects are partitioned in accordance to the AlMajdouie Information Reference Model (IRM).

The full name of an EBO should be constructed as follows:

<domain>.<sub-domain>.<eboShortName>

Where the domain and sub-domain are the classifications of this business object within the IRM. The short name MUST conform to the common naming standards defined earlier.

For example, the full name for the Customer EBO which is part of the party module in the customer sub-domain would be named as follows:

pty.cust.ebo.Customer

* + 1. **EBO Schema Name**

The name of the EBO schema file should be constructed as follows:

<domain>.<sub-domain>.<eboShortName>\_<version>.xsd

For example, the EBO for version 1.0 of the Customer EBO in the *Party Customer* capability would be named as follows:

pty.cust.Customer\_1.0.xsd

* + 1. **EBO Namespace**

The target namespace of the EBO should be constructed as follows:

http://www.almajdouie.com/ebo/<domain>.<sub-domain>.<eboShortName>

For version 1.0 of the Customer EBO in the *Party Customer* capability would be named as follows:

http://www.almajdouie.com/ebo/pty.cust.Customer

**NOTE:** The version of the EBO MUST NOT be included in the namespace of the EBO. This introduces brittle schemas where a version change can ripple through dependencies.

## Service WSDL

* + 1. **Service Name**

The name of a service must follow the general naming conventions detailed above. It must also be descriptive enough to describe the grouping of operations it contains. If the service is a Business Service type, the service name should be a simple composite name, generally with the business entity and a verb attached. Compound names must be in camel case[[2]](#footnote-2) with the first letter in lower case (e.g. propertySet). Examples of service names couple be

* orderFulfilment
* salesOrderProcessing

customerEntity

#### Service Names for Enterprise Business Services

Enterprise Business Services (EBS) are the external (to the capability) course-grained reuse points (as detailed in the SOA Architecture Blueprint). As such we impose an additional restriction on the EBS naming scheme.

The service name of the EBS has the additional restriction that it must align to the AlMajdouie Capability and Services Model. The lowest level of this model is the services provided by the capability. Using this naming will align the exposed services of the capability in the SOA infrastructure to the business model for reporting and measurement of SOA success.

If alignment of the service name doesn’t naturally fit to this model, approval and governance is REQUIRED from the Integration Competency Centre at first for validation and possible remediation.

#### Operation Naming

The name of an operation must be descriptive enough to give a reader an idea of its purpose. The name must be in the following format:

* A verb, for example, get, update, delete, process, calculate, followed by,
* A noun describing what’s being operated on. This noun need not be the name of an EBO; it might be an element on an EBO or metadata associated with it.

As detailed above (section 3.2.2), the operation name MUST NOT assume the context of the service name.

Some good examples of operation names are:

* getSalesOrder  
  This operation will return (*get*) a *SalesOrder* object.
* calulateGST  
  This operation will *calculate* the *GST.*
* setShippingInstructions  
  This operation will *set* the shipping instructions element or metadata.

enrolStudentLoan  
This is a good operation for an External Virtual Service named *Enrol*.

* + 1. **WSDL Filename**

The WSDL of a service is made up of three separate files:

* EBM  
  Defines the schema of each Enterprise Business Message (EBM) referenced by the WSDL
* WSDL  
  Defines the abstract WSDL plus the binding
* End Point (EP) WSDL  
  Imports the WSDL file, and includes the service end point definition

The name of the each of these artifacts should be constructed as follows:

|  |  |
| --- | --- |
| Artifact | Artifact Name |
| EBM | <capability>.<compType>.<compName>\_<version>.xsd |
| WSDL | <capability>.<compType>.<compName>\_<version>.wsdl |
| EP WSDL | <capability>.<compType>.<compName>\_<version>.ep.wsdl |

Table 3.2 – WSDL Naming Standards

For example, the WSDL for version 1.0 of the Notification external virtual service which is part of the communication capability in the utility competency would consist of the following artifacts:

util.comm.ebs.Notification\_1.0.xsd

util.comm.ebs.Notification\_1.0.wsdl

util.comm.ebs.Notification\_1.0.ep.wsdl

* + 1. **WSDL Namespace**

The target namespace of the WSDL should be constructed as follows:

http://www.almajdouie.com/<compType>/<capability>.<compName>

For example, the namespace for version 1.0 of the Financial Transaction external virtual service capability would be as follows:

http://www.almajdouie.com/ebs/pp.acc.Transaction.FinancialTransaction

Note:

* The same target namespace is used for all three artifacts which make up the service WSDL.
* The version of the WSDL MUST NOT be included in the namespace of the WSDL. This introduces brittle contracts where a version change can ripple through dependencies.
  + 1. **WSDL Content**

The Enterprise Business Message (EBM) naming standards are details in the Web Service Description Language (WSDL) Standards document. It is there the construct of the EBM is explained and how the EBM name is derived from the operation name (naming standards described below).

## Event EDL

The EDL for an event is made up of two separate files:

* EBM  
  Defines the schema of each Enterprise Business Message (EBM) referenced by the EDL
* EDL  
  Defines the abstract event definition

The name of the each of these artifacts should be constructed as follows:

|  |  |
| --- | --- |
| Artifact | Artifact Name |
| EBM | <capability>.<compType>.<compName>\_<version>.xsd |
| EDL | <capability>.<compType>.<compName>\_<version>.edl |

Table 3.3 – EDL Naming Standards

For example, the EDL for version 1.0 of the Notification events which is part of the communication capability in the utility competency would consist of the following artifacts:

util.comm.eve.Notification\_1.0.xsd

util.comm.eve.Notification\_1.0.edl

* + 1. **EDL Namespace**

The target namespace of the EDL should be constructed as follows:

http://www.almajdouie.com/<compType>/<capability>.<compName>

For example, the namespace for version 1.0 of the Notification external virtual event within the communication capability would be as follows:

http://www.almajdouie.com/eve/util.comm.Notification

NOTE:

* The same target namespace is used for both artifacts which make up the event EDL.
* The version of the EDL MUST NOT be included in the namespace of the EDL. This introduces brittle events where a version change can ripple through dependencies.

# Capability Naming Standards

This section details the directory structure and naming standards for all source artifacts which make up a capability. This structure has been designed to meet a number of goals, including:

* Maintain a single version of each shared artifact across capabilities and OSB and SCA components
* Remove any deployment dependencies between capabilities
* Remove any deployment dependencies between OSB and SCA
* Remove any deployment dependencies between SCA Components within the same capability
* Separate out from the core source code any environment specific configuration, such as fault polices and service end points.
* Leverage MDS to share resources (EBO XSDs, EBS WSDLs etc.) between SCA Capabilities.

## Source Code Management

This assumes that Subversion[[3]](#footnote-3) is used as the source code management system (SCM) for all artifacts.

This allows us to control what versions of each service are included in each build, as well as support patch releases (see 2.4.3) .

* + 1. **Subversion Repositories**

Each capability will be located in a different location within the same Subversion Repository. This allows each capability to be managed separately, and if required, has different levels of access.

The repository name, containing each capability, will be named as follows:

SOA/cap/<capability>

* + 1. **Subversion Externals**

The directory structure makes use of subversion externals which allows us to map the content of a directory within a different (or the same) repository to a local directory within our repository.

This allows us to maintain a single version of each shared artifact; for example each OSB Workspace requires a copy of the MDS project (which contains artifacts such as EBO’s and EBS WSDL’s) in order to be able to build and deploy the project.

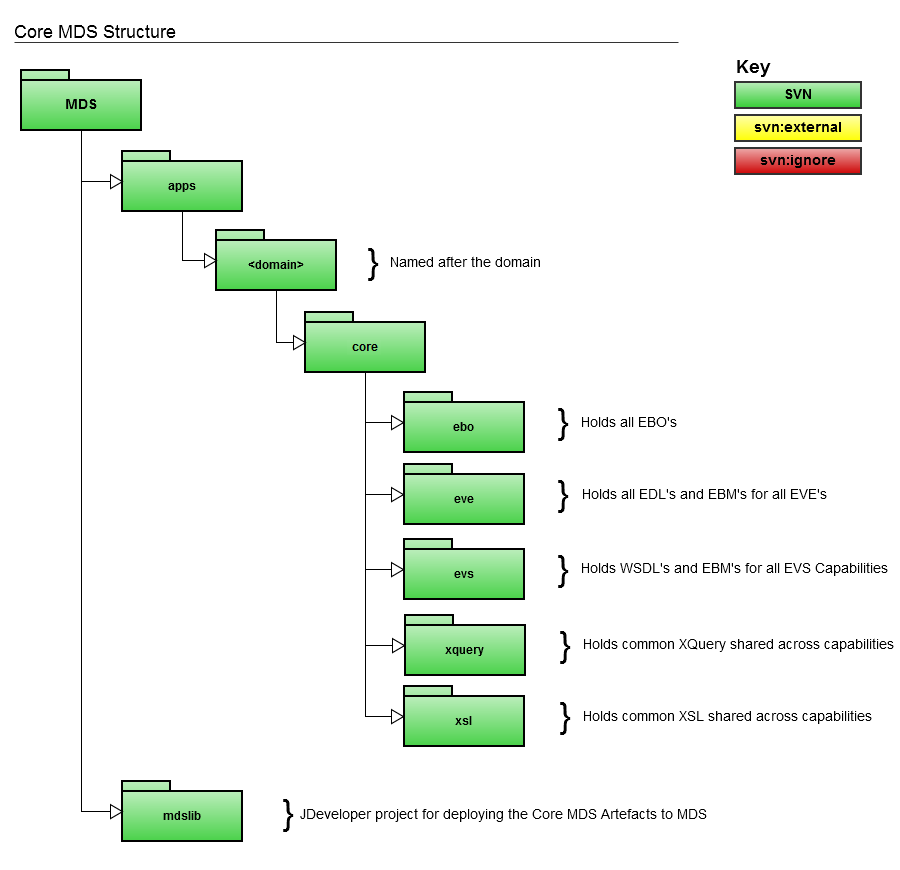
Without externals, we would be required to create a copy of the MDS project in each OSB Workspace and synchronize it with the “master” copy each time it is changed. This approach commonly leads to various copies getting out of sync and the issues that entails.

## Core MDS Project Structure

Within the SOA Reference Architecture the unit of deployment is a capability; however there are domain-wide resources, such as EBOs, WSDLs for an EBS, and common transformations that need to be shared between multiple capabilities.

This scenario is handled, by maintaining these shared resources within their own MDS project structure in a separate source code directory within the repository (from a versioning perspective).

This is referred to as the core MDS structure. The following diagram provides a high level overview of the directory structure for core MDS.



A copy of this structure is shared with each capability through the use of subversion externals. Allowing each capability to be built, tested and deployed independently of other capabilities.

When deploying a capability into a stage system or a production system, the Core MDS project should be deployed first to avoid any dependency errors that might cause the deployment to fail (see 5.1).

It should be noted that a benefit of this approach, is that it removes the need for complex dependency management, when managing the sequence in which components are deployed. Since components will only require the correct version of MDS to be deployed prior to their own deployment.

Note: Each version of a capability will be dependent on a particular version of the MDS project. It is important to ensure that the deployment mechanism for MDS and Capabilities is capable of managing these versioning dependencies.

* + 1. **apps/almajdouie/core**

This contains the components of the SCA Capability deployed to MDS. Each of these is detailed below.

#### ebo

Contains all the EBO's for the http://www.almajdouie.com domain. Schemas in here should be edited with EXTREME care as this will impact any capability that is dependent on these schemas.

#### eve

Contains the EDLs and EVEs for all External Virtual Events for the http://www.almajdouie.com domain.

The EDLs are owned by the capability responsible for publishing the event, so should be maintained and edited as part of that capability. However, as they are shared between multiple capabilities, they are included within the core MDS.

#### ebs

Contains the WSDLs and EBMs for all External Virtual Services for the http://www.almajdouie.com domain.

The EBS’s are owned by the capability responsible for providing the service, so should be maintained and edited as part of that capability. However, as they are shared between multiple capabilities, they are included within the core MDS.

#### xquery

Contains common xquery (e.g. for handling message header properties) that are shared across capabilities.

#### xsl

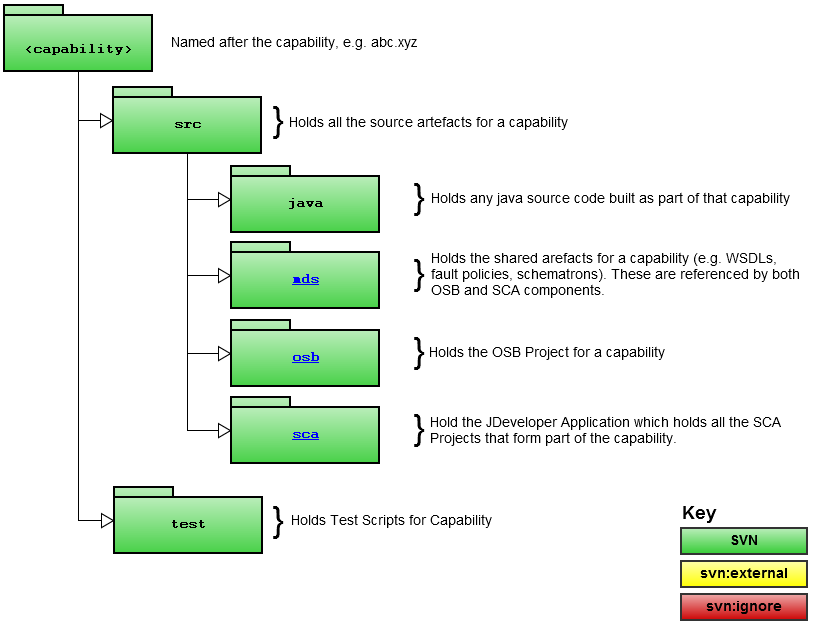
Contains common xsl (e.g. for handling message header properties) that are shared across capabilities.

* + 1. **mdslib**

This contains a JDeveloper project for deploying the Core MDS artifacts contained under the apps/almajdouie/mds directory to MDS.

## Capability Project Structure

The following diagram provides a high level overview of the directory structure for a capability.



* + 1. **src**

This contains the following sub folders:

* java  
  This is a place holder for any java source code built as part of the capability. This should follow the common AlMajdouie naming standards for Java.
* mds  
  Contains the artifacts for the capability which are deployed to MDS.
* osb  
  Contains the Eclipse Workspace for a capability, which contains all the OSB components that form part of the capability.
* sca  
  Contains a single JDeveloper application which contains all the SCA Projects that form part of the capability.

Each of these is detailed further below.

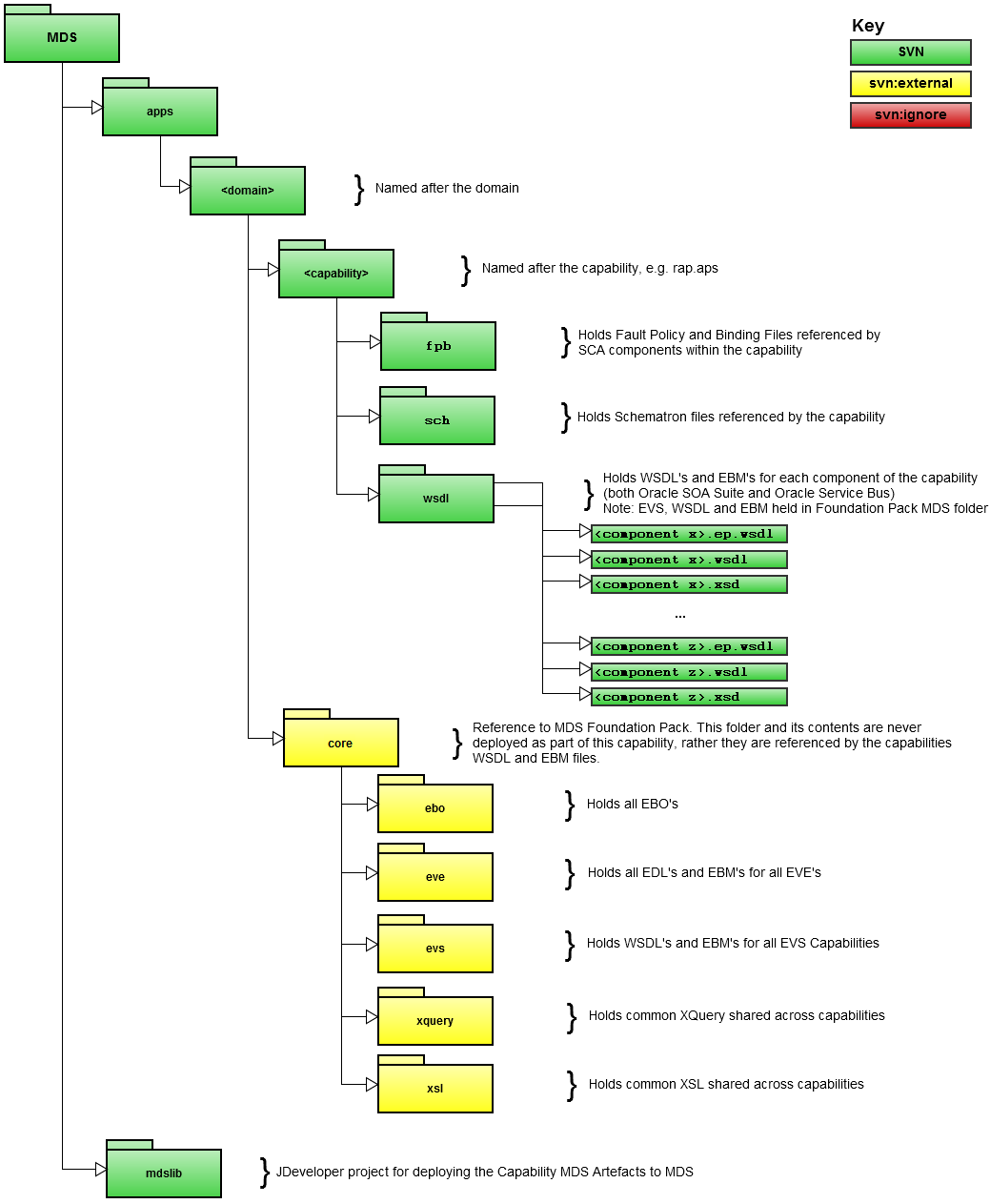
* + 1. **test**

This folder contains any test artifacts that are specific to this capability. Testing is outside the scope of this document, so is not defined any further.

## Capability MDS Structure

The Capability MDS Project structure is used to deploy artifacts that are ONLY referenced by this capability to MDS; this includes WSDL’s, Fault Policies and Schematrons.

The following diagram provides a high level overview of the directory structure for MDS Capability Structure.



Notice, the Capability MDS also includes a subversion external reference to the almajdouie Domain MDS structure. This is so that the MDS file system structure for a capability (once checked out), mirrors the structure in the SOA INFRA MDS after the almajdouie (after the Core MDS and the Capability MDS have been deployed).

This allows artifacts within the capability to reference artifacts within the Core MDS, via a relative path (as they will be when deployed to the SOA Infra MDS).

For example, the WSDLs and associated EBM’s for a capability, will typically reference one or more EBO’s within the Domain MDS. Having a subversion external reference, allows us to open those WSDL’s in a tool such as JDeveloper or XMLSpy and drill down to the EBO level.

* + 1. **apps/almajdouie/<capability>**

This contains the components of the SCA Capability deployed to MDS. Each of these is detailed below.

#### fpb

This contains the Fault Policies for the SCA Composites which make up the capability.

Note; these are deployed as part of MDS as this provides the flexibility to modify and re-deploy the fault policy for a composite without the need to re-deploy the actual composite.

#### sch

This contains the Schematrons for the SCA Composites which make up the capability.

Note; these are deployed as part of MDS as this provides the flexibility to modify and re-deploy the Schematrons for a composite, thus allowing us to change validation rules without the need to re-deploy the actual composite.

#### wsdl

Contains the Abstract WSDLs, End Point WSDLs and EBMs for each component of the capability.

These artifacts are also included (by externals) within the OSB Workspace for the capability.

Note: does not contain the WSDL’s for an EBS, as these are contained in the Core MDS Structure (see 4.2) as these are shared between capabilities.

* + 1. **apps/almajdouie/core**

This is a subversion external reference to the domain MDS project structure.

Note: This does NOT get deployed as part of the capability MDS, so the Domain MDS needs to be deployed prior to deploying the Capability MDS.

* + 1. **mdslib**

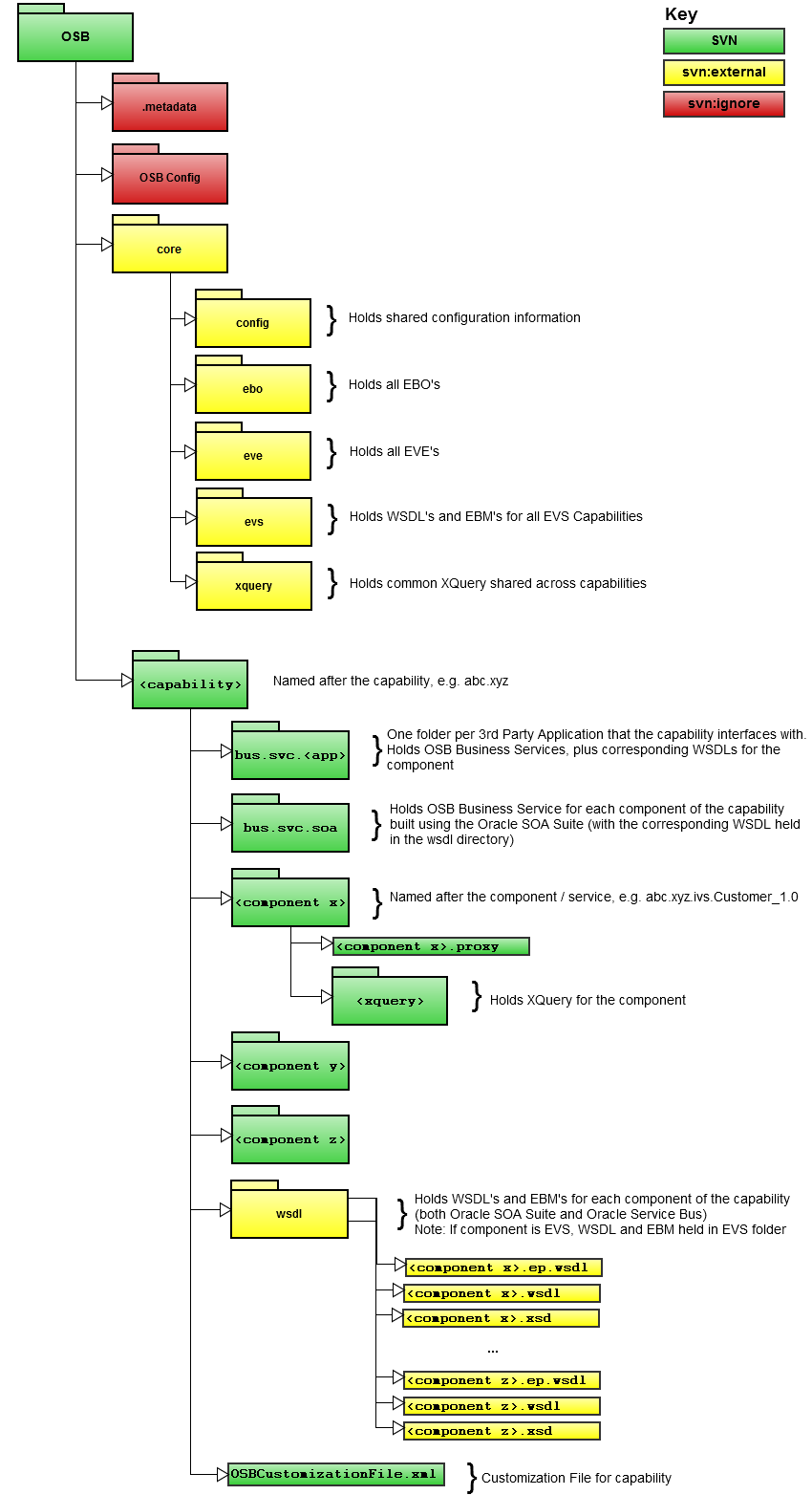
This contains a JDeveloper project for deploying the Capability MDS artifacts contained under the apps/almajdouie/<capability> directory to MDS.

Note: This should not include the artifacts in apps/almajdouie/mds.

## OSB Capability Structure

A single OSB Eclipse Workspace should be created to contain all OSB Proxies and related artifacts within a capability.

The following diagram outlines the file structure for the components of a capability implemented in OSB.



* + 1. **Eclipse Workspace**

The OSB Eclipse Workspace, should be named osb, and should contain the three projects:

* OSB Config  
  The master project within an OSB Workspace; it references the OSB Projects which make up the capability.
* core  
  Project containing resources which are shared between capabilities, for example EBO's for a domain.
* <capability>  
  Project containing all the components that make up a capability

#### .metadata

This folder contains Eclipse metadata about the project and the workspace, and is specific to an individuals working environment, so should not be placed under version control. This is achieved by setting the property

svn:ignore .metadata

on the osb directory.

#### OSB Config Project

The OSB Config project is the master project within an OSB Workspace; it references the OSB Projects which make up the capability as well as defining connections to an OSB Instance for the deployment of the OSB Projects within the workspace.

Creating this project also creates the .metadata project (which is required to work on an OSB Project). Again this folder must NOT be put under version control. This is achieved by setting the property svn:ignore as described above.

* + 1. **core**

This is a project containing resources which are shared between capabilities, for example EBO's for a domain (e.g. http://www.almajdouie.com). As these resources are shared we need to ensure that we only have a single instance of this project under subversion, this is achieved by first creating a separate MDS workspace (see OSB MDS Project Structure for details), and then using subversion to reference this an external project (using the svn:external command).

This will tell subversion that whenever you perform an svn update, to bring changes from the referenced external project into your working copy. Likewise, if you perform any updates on the shared project, then when committing your changes, any changes to the externally referenced project will also be committed to that project within subversion.

Folders contained in mds include:

* config  
  Holds shared configuration information, for example alert destinations
* **ebo**  
  Contains all the EBO's for a domain. Schema's in here should be edited with EXTREME care as this will impact any other capability that is dependent on these schema's.
* eve  
  Contains the EDLs and EVEs for all External Virtual Events for a domain.  
    
  As these are shared between multiple capabilities, only the EDLs and EBMs specific to the capability in the workspace should be edited here.
* ebs  
  Contains the WSDLs and EBMs for all External Business Services for a domain.

As these are shared between multiple capabilities, only the WSDLs and EBMs specific to the capability in the workspace should be edited here.

* xquery  
  Contains common xquery (e.g. for handling message header properties) that are shared across capabilities.
  + 1. **<capability name>**

This is a project containing all the components that make up a capability (and should have the full name of the capability, for example abc.xyz), this project will contain the following sub-folders:

* bus.abcs.<app>  
  Contains the WSDL's, plus supporting schema's for any application service invoked by the capability.
* bus.abcs.sca  
  Contains OSB Business Services (.biz) for each SCA component of the capability, with corresponding WSDL held in the wsdl directory.
* <service name>  
  One folder per service that makes up the capability.
* wsdl   
  Holds WSDL's and EBM's for each component of the capability (both Oracle SOA Suite and Oracle Service Bus)

Each of these is detailed further below.

#### bus.svc.<app>

This folder contains WSDL’s and supporting schemas for any application service invoked by the capability.

It will also contain a corresponding OSB Business Service (.biz) for each WSDL (so they can be invoked via the capability); this should take the same name as the WSDL with a .biz prefix in place of the .wsdl prefix.

For example, if we have the WSDL 3rdPartyService.wsdl, then the corresponding business service should be named 3rdPartyService.biz.

In the scenario where a WSDL includes multiple other WSDL’s and Schemas, it is recommend that one subfolder is created for each WSDL and its included files. The sub folder should have the same name as the wsdl, minus the .wsdl prefix; e.g. 3rdPartyService in the above example.

#### bus.abcs.soa

This folder contains OSB Business Services (.biz) for each SCA component of the capability, with corresponding WSDL held in the wsdl directory.

This should be named as follows:

<capability>.<compType>.<compName>\_<version>.biz

#### <component >

One folder per service (for example EBS, ABCS, Entity Service, Task Service) that makes up the capability.

This should be named as follows:

<capability>.<compType>.<compName>\_<version>

#### wsdl

This is a subversion external reference to the wsdl directory

apps/almajdouie/<capability>/wsdl

Under the capability MDS project structure.

This holds the WSDL's and EBM's for each component of the capability (both Oracle SOA Suite and Oracle Service Bus).

#### Service End Point

The endpoint URI for a Proxy Service should be defined as follows:

***EBS***

For a proxy service which is an EBS, its endpoint should be constructed as follows:

/ebs/<capability>.ebs.<compName>\_<version>

In the case where there are different EBS end points (e.g. IEBS, TEBS) the end point will be constructued as follows:

/iebs/<capability>.ebs.<compName>\_<version>

/tebs/<capability>.ebs.<compName>\_<version>

***IEC***

For a proxy service which is an IEC, its endpoint should be constructed as follows:

/iec/<capability>.iec.<compName>\_<version>

***Intra Capability Components***

For a proxy service, which will only be invoked by other components within the same capability, for example an IVS, Task, Func or Entity Services; then its endpoint should be constructed as follows:

/cap/<capability>.<compType>.<compName>\_<version>

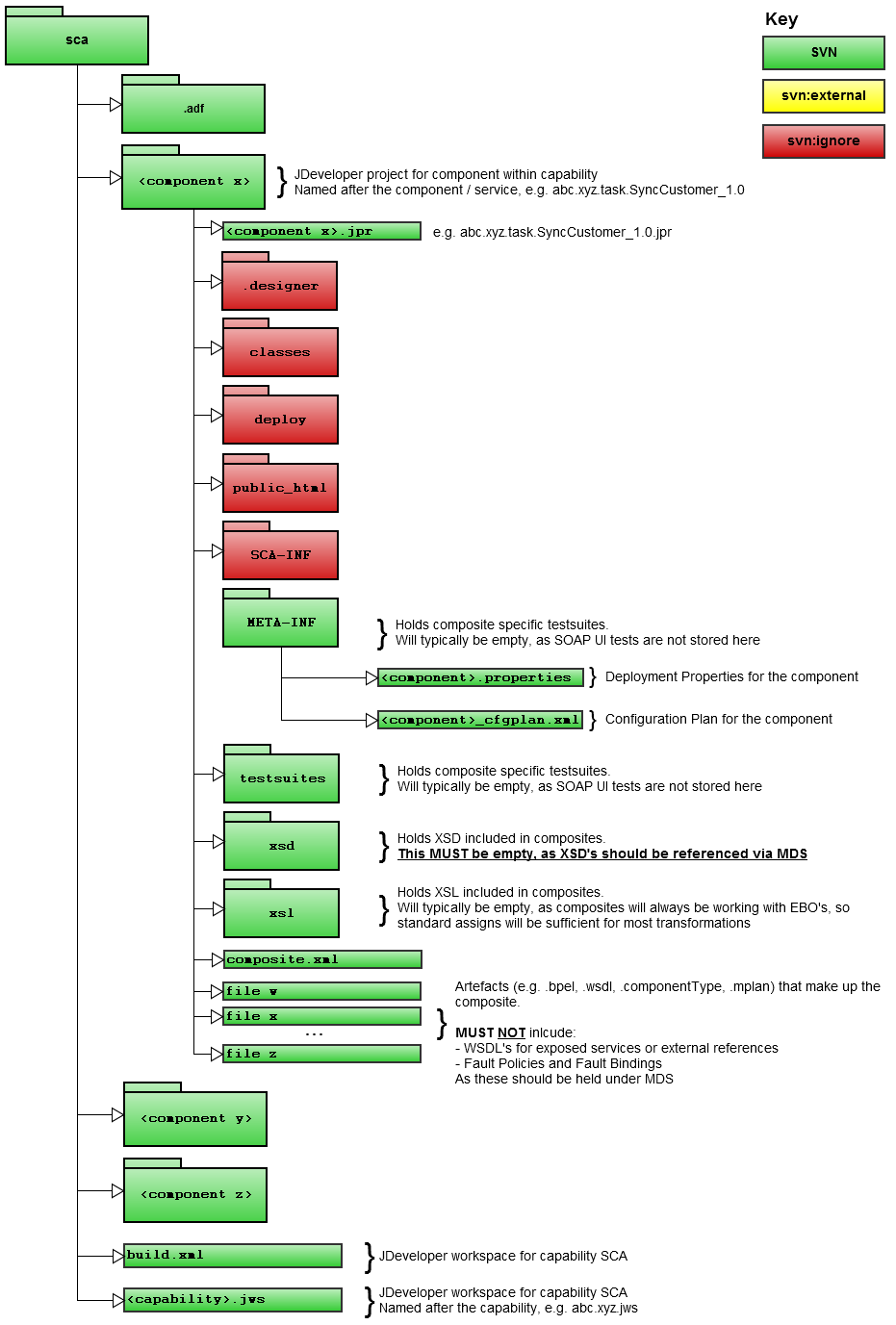
***CVS***

For a proxy service which is a CVS, its endpoint should be constructed as follows:

/cvs/<capability>.<app name>.cvs.<compName>\_<version>

## SCA Capability Structure

The following diagram outlines the file structure for the components of a capability implemented as a SCA composite.



## SCA Composite

The following naming standards should be followed for a services implemented as a SCA Composite.

All the artifacts that make up a composite should be contained in a folder named sca.

Note: Where multiple operations are provided by a service, then these should all be implemented as components within the same SCA Composite.

* + 1. **JDeveloper Application Name**

A single JDeveloper application should be created to contain all SCA Composites within a capability. The JDeveloper application should be named after the capability and constructed as follows:

<capability>.jws

* + 1. **JDeveloper Project Name**

The JDeveloper project name for a service implemented as a SCA Composite should be constructed as follows:

<capability>.<compType>.<compName>\_<version>

* + 1. **Exposed Services**

A SCA composite should have at most a single exposed service; this should be named as follows:

<capability>.<compType>.<compName>\_<version>

In cases where a composite is just processing EDL events, then it will not have any exposed services.

The exposed service MUST reference a WSDL deployed to MDS; in other words the WSDL for the exposed service must not be included in the JDeveloper project for the application.

* + 1. **External References**

External references within a SCA Composite, should ALWAYS fall under one of the following categories:

* SCA Composite that is part of the same capability
* OSB Proxy that is part of the same capability
* EBS implemented as an OSB Proxy

In each case, the External Reference should be named after the service it is referencing, and should be constructed as follows:

<capability>.<compType>.<compName>\_<version>

The external reference MUST reference the End Point (EP) WSDL for the service that is deployed to MDS; in other words the WSDL for the external reference must NOT be included in the JDeveloper project for the SCA Composite.

* + 1. **SCA Composite Fault Policies**

The Fault Policy for a SCA Composite is made up of two separate files:

* **Policy**   
  Defines the fault policies for a SCA Composite.
* **Binding**  
  Defines to which components within a composite that a fault policy is to be applied, this is known as binding.

The name of the each of these artifacts should be constructed as follows:

|  |  |
| --- | --- |
| **Artifact** | **Artifact Name** |
| Binding | <capability>.<compType>.<compName>\_<version>.bindings.xml |
| Policy | <capability>.<compType>.<compName>\_<version>.policies.xml |

For example, the fault policy for version 1.0 of the Notification external virtual service which is part of the communication capability in the utility competency would consist of the following artifacts:

util.comm.ebs.Notification\_1.0.binding.xml

util.comm.ebs.Notification\_1.0.policies.xml

These files should be included in the Capability MDS Structure (see 4.4) under the path:

apps/almajdouie/<capability>/fpb

To reference policies deployed on MDS, the follow properties need to be added to the composite.xml file for the component:

oracle.composite.faultPolicyFile

oracle.composite.faultBindingFile

These should be added directly following the service element and reference the location of the policy and binding files in MDS, as shown in the following code fragment:

<property name="oracle.composite.faultPolicyFile">

oramds:/apps/almajdouie/<capability>/fpb/*<policy>*

</property>

<property name="oracle.composite.faultBindingFile">

oramds:/apps/almajdouie/<capability>/fpb/*<binding>*

</property>

* + 1. **Schematron**

A SCA composite may have zero or more schematron files; this should be named as follows:

<capability>.<compType>.<compName>.<schematronName>\_<version>.sch

In cases where a schematron is shared between multiple composites within a capability, it should be named as follows:

<capability>.<compType>.<schematronName>\_<version>.sch

Where <schematronName> is a meaningful name which describes the purpose of the validation.

All Schematrons referenced by a SCA Composites must be deployed as part of the Capability MDS (see 4.4) under the path:

apps/almajdouie/<capability>/sch

Where a schematron is referenced within a composite, it must be reference via MDS; in other words Schematrons must NOT be included in the JDeveloper project for the SCA Composite.

# Fault Standards

This section of the naming and deployment standards covers standards related to the creation and throwing of faults.

## Fault Codes

Fault codes will be constructed using the following format:

<capability>:<code>

where the code is derived from the code range defined in the table below.

|  |  |
| --- | --- |
| **Code Range** | **Fault Types** |
| 000-099 | Unexpected errors not covered by any other error range defined below or not defined to be handled by a fault handler |
| 100-199 | Operation related errors e.g. operation not supported |
| 200-299 | Network/communication errors |
| 300-399 | Policy related errors e.g. security failures |
| 400-499 | Errors internal to SOA i.e. OSB or SOA Suite composites |
| 500-599 | Validation errors |
| 600-699 | *Not Defined* |
| 700-799 | *Not Defined* |
| 800-899 | *Not Defined* |
| 900-999 | Provider system errors ie errors in a system the SOA middleware communicates with |

# Deployment Standards

This section of the naming and deployment standards covers where artifacts should be deployed on the Oracle SOA Suite.

## SCA Capability Deployment

The following diagram illustrates the sequence in which JDeveloper Applications must be deployed for the SCA components of a capability.

It should be noted that when multiple capabilities are deployed, the SCA Core MDS only needs to be deployed once (assuming all capabilities are dependent on the same version of the Core MDS).

* + 1. **Deployment Partition**

When deploying a SCA Capability it should be deployed to a partition named after the capability replacing any “.” Characters with a “\_” character due to partition names not supporting the ”.” character. The partition name is constructed as follows:

<capability>

For example the component util.comm.task.Notification\_1.0 would be deployed to the partition util\_comm.

* + 1. **Patch Number**

The Deployment revision number is used to capture the patch number of the SCA Composite being deployed (see 2.4.3).

For example, the first deployment of util.comm.task.Notification\_1.0, would be revision 1.0, with a subsequent patch being revision 2.0. If we were to deploy version 1.1 of the service, then the revision would be set to 1.0.

This allows patches to be deployed without impacting running or completed instances of the composite.

* + 1. **Configuration Plan**

The configuration plan for a SCA composite should be named as follows:

<capability>.<compType>.<compName>.<version>\_cfgplan.xml

Environment specific configuration plans should NOT be required, as the only difference between environments will be the end points for external references. These are defined in the ep.wsdl files (see 3.3) which are deployed separately to MDS.

## OSB Deployment

The following diagram illustrates the sequence in which OSB Projects must be deployed for the OSB components of a capability.

It should be noted that when multiple capabilities are deployed, the OSB Domain MDS only needs to be deployed once (assuming all capabilities are dependent on the same version of the Core MDS).

* + 1. **Deployment Project**

When deploying an OSB Capability it will be deployed to a project named after the capability, the project name is constructed as follows:

<capability>

For example the component util.comm.ebs.Notification\_1.0 would be deployed to the project util.comm.

* + 1. **Patch Number**

Unlike SCA Composites, the Oracle Service Bus does not provide a means for side by side deployment of multiple revisions of the same (version of a) Proxy Service.

As OSB Proxy Services are stateless, this typically does not prove an issue.

* + 1. **OSB Customization File**

The customization file for an OSB Capability should be named as follows:

OSBCustomizationFile.xml

Aspects of the customization file which vary between environments, for example end points of external web services should be defined in separate property files and substituted at deployment. These property files should be maintained as part of the AlMajdouie build and deployment solution.

1. In reality some components, such as EBOs, may need to be deployed twice, once to the Oracle Service Bus and once to the Oracle SOA Suite if both products are in use. [↑](#footnote-ref-1)
2. http://en.wikipedia.org/wiki/CamelCase [↑](#footnote-ref-2)
3. http://subversion.tigris.org [↑](#footnote-ref-3)