Digital Imaging and Communications in Medicine (DICOM)

Supplement 224: Service Discovery and Control

Prepared by:

DICOM Standards Committee, Working Group 23: Artificial Intelligence/Application Hosting

1300 N. 17th Street, Suite 900

Rosslyn, Virginia 22209 USA

Status: Version 2, March 25, 2021

Developed pursuant to DICOM Work Item 2020-08-A

Table of Contents

Document History

2020/08/25	Version 1	BB	New Document – Initial Working Draft
2021/03/25	Version 2	BB	Moved Entrypoints to Traits to comply with new OAM spec
	Version 3	BB	Content for WG 6 first read

Open Issues

1.	WADO-WS was left out on purpose, should it be, are there any use cases for it?
2.	What would be the minimum supported traits for DICOM compliance? These have been proposed in the document, but are the sufficient?
3.	How to handle expired or leaked secrets, is this in scope or handled by the platform on a per inference request. When can a platform restart a container?
4.	Fold content into Part 19. Part 19 becomes more generalized to be a collection of services and interfaces, current DICOM API becomes a nested section 9.
5.	For WADO should we just deal with data transfer only, and items such as iccprofile be removed? What would be the minimum attributes required to be part of the supplement?
6.	Are REST API Provider and User both needed, or can Provider be sufficient to cover all use cases?

Closed Issues

1.	FHIR Endpoint being used as an Entrypoint trait is outside the scope of this supplement, it can be created, but should not be added to DICOM.

1. Scope and Field of Application

Supplement 224 adds mechanisms for discovering and managing processing services which will be referred to throughout the supplement as “Applications”.

These Applications are comprised of one or more Components. Components each perform a single task and must expose an Entrypoint to process data dispatched to it by the Platform in the form of jobs. Entrypoints are applied to Components as traits. The implementation mechanism, replicability, lifetime and Entrypoint of a given Component are defined by the Component's Workload type and characteristics. Manifests are used to parameterize Components as well as describe an Application to a Platform. Platforms are systems that register Applications using Manifests and dispatch work to an Application’s

Entrypoint based on the Application Scope. In the context of this document, each of an Application's Components carries out a specific action, such as detecting lesions or segmenting anatomical regions.

This supplement will also cover support mechanisms for data exchange with a focus on transaction security.

2. Normative References

The following standards contain provisions that, through reference in this text, constitute provisions of this Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Standard are encouraged to investigate the possibilities of applying the most recent editions of the standards indicated below.

Open Application Model OAM v0.2.0 27 Mar 2020 https://github.com/oam-dev/spec

Kubernetes v1,19 28 Aug 2020 https://kubernetes.io/docs/home/

3. Definitions

OAM - Open Application Mode, defines a number of standard but extensible abstractions to model micro-service applications by natural, with operation configurations as part of the application definition. OAM resource types can be defined as core, standard or extended. Core resource definitions must exist on every OAM platform.

Kubernetes - An open-source container orchestration engine for automating deployment, scaling, and management of containerized applications

YAML - (a recursive acronym for "YAML Ain't Markup Language") is a human-readable data-serialization language. It is commonly used for configuration files and in applications where data is being stored or transmitted.

Application Programming Interface (API) - A set of interface methods that Applications and Platforms can use to communicate with each other.

Component - An entity that maps characteristics to an implementation mechanism such as a container or network service. Where the Component is a static network service or URI, scaling is the responsibility of the Component. Where the Component is a container, virtual machine or other managed executable, scaling is the responsibility of the Platform. The implementation mechanism of a Component is defined the Component Workload type, while replicability, lifetime and some security aspects are defined by the Component Traits.

Workload - The core definition of a component. These can be a ContainerizedWorkload, which requires instantiation and scaling is the responsibility of the Platform, DicomServerWorkload, which provides network services or request URIs, are not controlled by the Platform and scaling is the responsibility of the Workload, and DicomTaskWorkload, which are parameterized executables where scaling is the responsibility of the Platform.

Entrypoint - Entrypoints describe how data is accepted and results are emitted. These can be thought of as SCU-SCP interface methods such as DIMSE storage services, web service APIs or mapped input/output directories. Entrypoints and their security are defined by applying them as traits to a Component in the Application Configuration.

Application – A combination of one or more Components described by a Manifest. These can be network services, APIs, executables or containers. The Application additionally may be hosted thus, utilizing services and resources offered by the Platform. The Application functions as the SCP to the Platform for the work to be done.

Trait - A trait defines a piece of add-on functionality, characteristics, that pertains to the operation of a Component and defined in the Application configuration. Traits may be limited to certain Workload types based on their definition or a Component's characteristic definition. Traits represent features of the system that are operational concerns, not developer concerns.

Scope - Scopes provide different ways to group components into applications. Scopes are applied to an Application Configuration. Each scope represents some associated behavior or functionality.

Parameter – A Parameter is an attribute in the specification that is made mutable, this can be required to be defined in the Application Configuration or can have a default value which will be used if a value is not specified.

Manifest – Used to describe an Application and its Components. It contains a description of the underlying implementation mechanism (service, container, etc.) as well as the Application's Scope and Trait details. The Manifest is the authoritative description of an Application.

Platform – The system used to register, manage and interact with Applications. A Platform employs Applications to perform work. Platforms register Applications using the Application's Manifest, thereby functioning as the SCU for each individual Application. A Platform can host, instantiate, parameterize executables of or activate via API or use of network service, an Application. When the Platform acts as a host, the Platform provides the infrastructure in which the Application runs and interacts with the external environment. This includes network access, database and security.

Resources - Resources are the building blocks of an Application Configuration Manifest. Resources need schemas defined for them before a unique resource instance can be defined.

A. Service Discovery and Control Overview

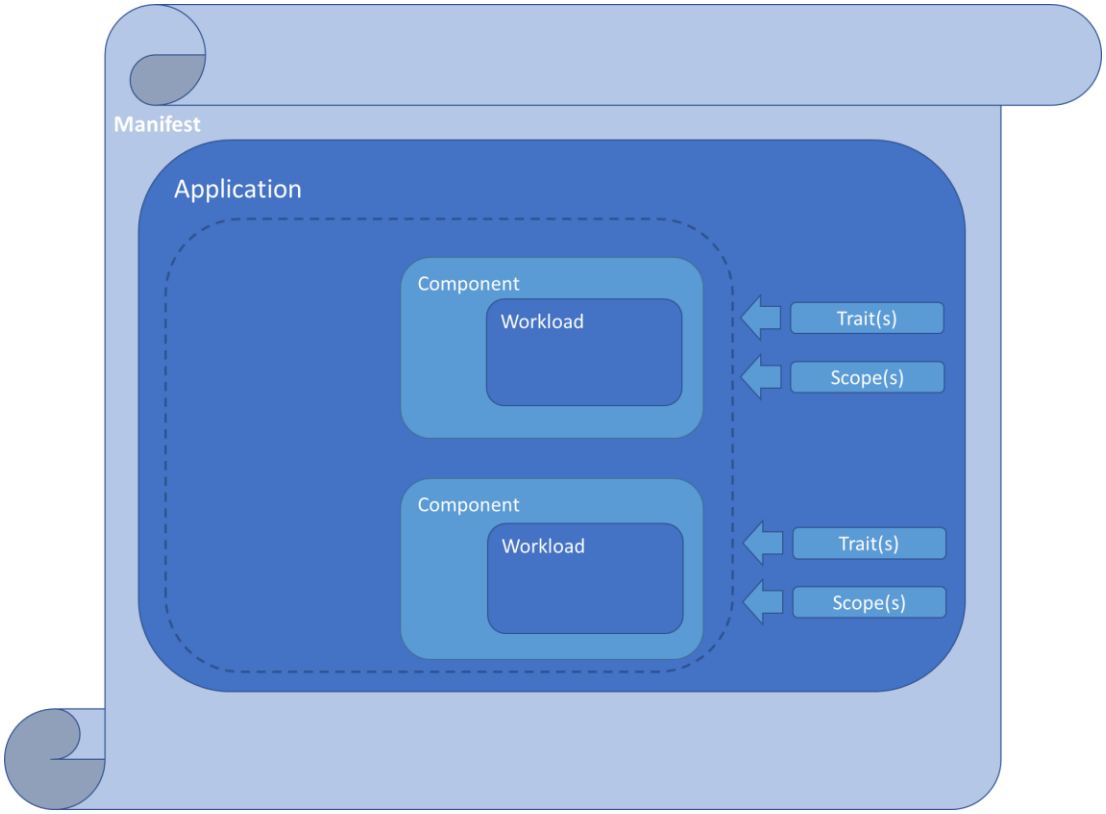
This supplement provides a mechanism for discovering and managing processing services. With the rise of artificial intelligence, containerized processing, service-oriented architecture, and microservices, there is a proliferation of processing services in the medical imaging space. Systems that use those services need to discover what services are available and to launch and control those services, for example, signaling a service to change its operational state ("start", "stop", "prepare for processing", etc.).

This supplement introduces a Manifest that describes the Application to a Platform. The Manifest is a unique object that describes what does the Application does, what is needed to run the Application and/or where it is located as well as what the input and output Entrypoints are. A Manifest is used to describe DICOM services or Applications in general, that is "I provide services X, Y, Z", and the ability to apply those services to specific instances, so that both discovery of what services are available can be found and registry of what services can be provided can be made.

A.1 Manifest Structure Overview – YAML

Figure A.1 is a visual representation of the all the components of a DICOM Application Manifest and how they are related to one another.

Figure A.1 Manifest Structure



The structure of a complete DICOM Application Manifest YAML file is as follows:

Scope Definition(s)

Component Definition (1)
- Characteristics
- Workload
  - Workload Specific Settings
- Parameter Declaration

Component Definition (2)
- Characteristics
- Workload
  - Workload Specific Settings
- Parameter Declaration

Application Configuration
- Components
  - Component 1
    - Workload Settings
    - Trait Properties
    - Scope Reference
  - Component 2
    - Workload Settings
    - Trait Properties
    - Scope Reference

The Manifest leverages the Open Application Model (https://oam.dev/) for DICOM service discovery and control, and specifically application manifests. Briefly, in OAM:

- A developer creates an application or service. To deliver it to users, the developer defines how to discover, instantiate and interact with it in a YAML manifest. This manifest encapsulates a workload and the information needed to run it.
- An application operator deploys instances of an application and configures it with “operational traits” (parameters that control things like instance replication).
- The application developer and application operator have so far described an application and its operational characteristics in platform-neutral terms. The power of the Open Application Model comes from the underlying platforms that implement the model.

The DICOM Application Manifest may contain only an Application Configuration if the Scope and Component definitions it references have previously been registered with the Platform. Also, any Trait schemas referenced need to be available on the Platform as well. See Table A.1

Table A.1 Manifest Sections

Manifest Section	Optionality
Scope Definitions	O
Component Definitions	O
Application Configuration	M

B. Resources

Resources are the building blocks of an Application Configuration Manifest. Resources need schemas defined for them before a unique resource instance can be defined. The standardized schemas for DICOM Scopes, Traits, and Component Workloads are defined as part of the DICOM Standard. Unique instances of these resources are defined by application developers within their Application Configuration Manifest. These can also be defined individually prior to posting an Application Configuration Manifest whereas an Application Configuration would reference previously defined resources that are already defined within the Platform.

When unique instances of a resource are created those values which the owner of the resource has made mutable, or requires a value be entered by the consumer of the resource, will be specified as a parameter in its definition. The actual values for each of these parameters are defined as part of the Application Configuration in which the resource is used.

The Application Configuration schema used for DICOM is the same as defined for OAM. The schema listed is valid for the version of OAM listed as a normative reference.

B.1 Resource Schemas

The resource definition schemas shall be specified as shown in Table B.1. All resources, Resource Definitions, Scopes, Traits, Component Workloads and Application Configurations, will use this base schema to define their contents.

Table B.1 Resource Schema Attributes

Attribute	Type	Description
apiVersion	string	A string that identifies the version of the schema the object should have. For example, the standard types use standard.oam.dev/v1alpha3
kind	string	A string defining the type of resource that is being defined. The types of schemas defined within this specification are Resource Definitions, Scopes, Traits, Components and Applications.

metadata	Metadata (see B.1.1)	Information about the resource. The name value in the metadata must be unique to a platform. This name is used as the kind when defining a unique resource instance using its schema.
spec	Spec (see the referenced resource schema)	The settings to be used to define resource. These setting are the standardized resource schema unless they are used in as a resource definition whereas to define the schema itself.

When resources are defined with parameters, configuration settings that are either mutable or require user values, the parameters attribute is added to the schema. Resources where parameters are valid are noted as part of their specification properties in section B.3.

B.1.1 Resource Metadata

Metadata provides information about the contents of the component. The name attribute is used to uniquely identify the resource. Table B.1.1 provides the schema for the metadata section of a resource.

Table B.1.1 Resource Metadata Schema Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		A name for the component
labels	Labels (see B.1.1.1)	N		Standard label requirements are based on Workload Type. Labels follow the Kubernetes specification for labeling.
annotations	Annotations (see B.1.1.2)	N		Annotations provide a mechanism for attaching arbitrary text within the metadata of an object. The annotations object follows the Kubernetes specification

B.1.1.1 Labels

Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system. Table B.1.1.1 provides the schema for the labels section of metadata.

Table B.1.1.1 Metadata Labels Schema Attributes

Attribute	Type	Required	Default Value	Description
app	map[string]string	N		A name for the schematic

B.1.1.1.1 Example Labels

An example of a label is shown with multiple entries to provide vendor and support information. This is how the labels section would appear in a resource.

```
labels:
        vendorName: applicationVendor
        phone: 8005551212
        supportEmail: support@applicationVendor.com
```

B.1.1.2 Annotations

Annotations provide a mechanism for attaching arbitrary text within the metadata of an object. Table B.1.1.2 provides the schema for the annotations section of metadata.

Table B.1.1.2 Metadata Annotations Schema Attributes

Attribute	Type	Required	Default Value	Description
app	map[string]string	N		A name for the schematic

Table B.1.1.2.1 shows annotation labels that are predefined and recommended.

Table B.1.1.2.1 Recommended Metadata Annotations

Attribute	Type	Required	Default Value	Description
description	string	N		A short description of the component.
version	string	N		A user provided string defining the semantic version of the component, e.g. the release version of this software

B.1.1.2.1 Example Annotations

An example of an annotation is shown with the recommended entry to provide information. This is how the annotation section would appear in a resource.

```
annotations:
        description: sample api frontend
        version: “1.2.1”
```

B.1.2 Example Resource Instance

An example of a DICOM Operation Scope resource instance is shown with the recommended metadata information provided. The spec section of the DICOM Operation Scope resource is defined in section C.1.1. This is how a properly formatted resource would appear.

```
apiVersion: standard.oam.dev/v1alpha3
kind: DicomOperationScope
metadata:
   name: example-operation-scope
   labels:
        vendorName: applicationVendor
        phone: 8005551212
        supportEmail: support@applicationVendor.com
   annotations:
        description: aiAppScope
        version: “1.2.1”
spec:
   type: workitem
   vendor: MyAIApp
   version: 1.2.1
   code: RDES128
   codeSystem: https://radelement.org
   sopClasses:
   - 1.2.840.10008.5.1.4.1.1.2
   - 1.2.840.10008.5.1.4.1.1.2.1
```

- 1.2.840.10008.5.1.4.1.1.4
- 1.2.840.10008.5.1.4.1.1.4.1
friendlyName: SubduralHematomaDetection

B.2 Resource Schema Definitions

When creating a resource schema definition, the type of schema being defined is listed in the attribute kind. The valid kinds for definitions are as follows:

- ScopeDefinition
- WorkloadDefinition
- TraitDefinition

Note the following DICOM resource schema definitions already exist as part of DICOM compliant OAM platform and can be used to define unique instances. The schemas themselves are defined in Annex A and the attributes defined as part of the DICOM standard for reference on defining the unique resource instances.

- Scopes – Definitions of each resource type are included in section C.1.
    - DicomOperationScope
- Component Workloads – Definitions of each resource type are included in section D.2.
    - ContainerizedWorkload
    - DicomServerWorload
    - DicomTaskWorkload
- Traits – Definitions of each resource type are included in section E.1.
    - JobTimeout
    - AuditTrail
    - TimeSync
    - AppCStoreProvider
    - AppCStoreUser
    - AppWadoUser
    - AppStowProvider
    - AppStowUser
    - OperatorInput
    - OperatorOutput
    - RestApiProvider
    - RestApiUser
    - UserIdentitySecurity
    - License

B.2.1 Example Resource Schema Definition

These schemas define the spec section of the resource along with each attribute's properties and dependencies. DICOM resource schemas are defined in Annex A. To create additional resource definitions, refer to the OAM specification and ensure the Platform implementation supports the schema. For reference this is how a properly formatted resource definition would appear.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AppCStoreProvider
spec:
    aet:
        description: This is the Called AET (Application Entity Title) of the
Entrypoint.  It is the intended acceptor of the service request.
        type: string
    port:
```

```
        description: This is the port which will be used to initiate the
DIMSE service request.
        type: integer
required:
- aet
- port
additionalProperties: false
```

B.3 Required Resources

Table B.3 lists the resource definition schemas of a DICOM Platform. To be minimally compliant a Platform must support and host all the mandatory DICOM resource schemas. A DICOM Platform must list all the resource schemas it supports in its conformance statement.

Table B.3 Required Resources

Resource Definition	**Optionality**	***To be removed late, but rationale***
DicomOperationScope	Mandatory	*Needed to provide linkage to job requests*
ContainerizedWorkload	Mandatory	*Core OAM component*
DicomServerWorkload	Mandatory	*Core DICOM feature*
DicomTaskWorkload	Optional	*May be most basic function, yet difficult to achieve. There can be countless environmental combinations that would need to be supported.*
JobTimeout	Mandatory	*To allow control of resource utilization*
AuditTrail	Mandatory	*Basic requirement, system must be able to audit PHI data interactions*
TimeSync	Mandatory	*To ensure audit messaging can be properly traced*
AppCStoreProvider	Mandatory	*Core DICOM feature*
AppCStoreUser	Mandatory	*Core DICOM feature*
AppWadoUser	Optional	*Providing core DICOM functionality may be enough. If a client has implemented DICOM Rest Services, it would be a reasonable assumption that they could implement or fall back to DIMSE.*
AppStowProvider	Optional	*Providing core DICOM functionality may be enough. If a client has implemented DICOM Rest Services, it would be a reasonable assumption that they could implement or fall back to DIMSE.*
AppStowUser	Optional	*Providing core DICOM functionality may be enough. If a client has implemented DICOM Rest Services, it would be a reasonable assumption that they could implement or fall back to DIMSE.*
OperatorInput	Mandatory	*Provides the lowest bar for entry, without the need to create additional services. This allows a core OAM function of mounts to be used in a way to specify an entrypoint.*
OperatorOutput	Mandatory	*Provides the lowest bar for entry, without the need to create additional services. This allows a core OAM function of mounts to be used in a way to specify an entrypoint.*
RestApiProvider	Optional	*Providing core DICOM functionality may be enough, adding the ability to host APIs and the proxy requirements can become complicated. Having alternatives such as DIMSE and Operators will allow for fallback options.*

RestApiUser	Optional	*Providing core DICOM functionality may be enough, adding the ability to host APIs and the proxy requirements can become complicated. Having alternatives such as DIMSE and Operators will allow for fallback options.*
UserIdentitySecurity	Optional	*Applications may not need this functionality. If the Platform does not provide this functionality, the clients could make the request without it.*
License	Optional	*Applications may not need this functionality. If the Platform does not support this requirement a client could implement this another way.*

B.4 Defining Unique Resources

When using a DICOM resource schema to define a unique resource instance, that is a resource whose settings follow a previously defined standard schema, these unique resource instances use the same basic schema template as shown in Table B.1.

B.4.1 Scope Specifications

Scope Definitions do not contain parameters. Unique definitions are applied at the Application Configuration level of a Manifest using Definition References, see section B.5. The settings for these resources are defined in their resource definition and all attribute values are provided. The specifically defined settings for DICOM standard Scopes are listed in section C. The types of Scope resource schemas defined within this specification are listed in section B.2.

B.4.2 Trait Specifications

Trait Definitions do not contain parameters as they are applied at the Application Configuration level of a Manifest. The properties of these resources are defined in their definitions and all values are provided as part of the manifest Application Configuration. Therefore, all settings are properties of the trait attributes. The specifically defined settings for DICOM standard Traits in section F. The types of Trait resource schemas defined within this specification are listed in section B.2.

B.4.3 Component Workload Specifications

The Workload Definitions may contain parameters, elements of the resource schema that are mutable or require a user definition. Workload Definitions permit component owners to declare, in infrastructure-neutral format, the runtime characteristics of a discrete unit of execution. The specifically defined settings for DICOM standard Component Workloads are listed in section E.2. Components specify Workloads, their characteristics, which Traits it can support, and attributes as well as declare parameters which can or must have values provided as part of the Application Configurations in which they are used. The types of Workload resource schemas defined within this specification are listed in section B.2.

B.5 Definition Reference

Once resources have been defined and loaded by a Platform, definition references are created which are later used by the Platform to validate the unique resource definitions that use them. For a Platform to claim it is DICOM complaint, all standard resource definitions in DICOM Part 19 Table B.3 defined as Mandatory, must be present. Definition references can be used for DICOM Operation Scopes that are predefined on a Platform in an Application's Manifest.

Platforms will create references that can be used in defining resources as part of a manifest as well as reading a Platform's capabilities. The reference specification is the resource kind, appended with 'Ref'. For example, scopeRef. Table B.5 provides the schema for reference to a defined Platform resource.

Table B.5 Definition Reference Attributes

Attribute	Type	Description
definitionRef	ResourceRef (see B.5.1)	The refence information to the resource definition.

B.5.1 Resource Reference

A resource reference indexes a reference by name. These names uniquely define a resource and must be unique to a Platform. Table B.5.1 provides the schema for the Definition Reference section.

Table B.5.1 Resource Reference Attributes

Attribute	Type	Description
name	sting	The name used to uniquely identify the resource as well as the group to which it belongs.

B.5.2 Example Definition Reference

An example of a Definition Reference instance is shown where Resource Definition is the resource kind. This is how a properly formatted resource would appear.

```
apiVersion: standard.oam.dev/v1alpha3
kind: ScopeDefinition
metadata:
  name: DicomOperationScope
spec:
  definitionRef:
    name: schema.dicomoperationscope.oam.dev
```

C. Scopes

Scopes group components into applications. Scopes are applied to Components in the Application Configuration. Each Scope represents some associated behavior or functionality. The DICOM Operation Scope is used to associate jobs to Applications, for example procedure codes to capabilities. There are no core Scopes defined by OAM, but OAM has Network and Health Scopes defined as standard Scopes.

C.1 DICOM Standardized Scope Definition Schemas

Application Scopes are used to group components together into logical applications by providing different forms of application boundaries with common group behaviors. These schemas are used to define the scope of a hosted application within DICOM. DicomOperationScope is a reserved name with a standardized schema defined within this standard. This is defined as a Standard Scope definition of OAM. Scopes do not contain the parameters. The settings for Scopes are defined in their resource definitions and are referenced as part of the manifest Application Configuration using a scopeRef. See Section B.5 for more information.

C.1.1 DICOM Operation Scope Schematic

The following is the definition reference for DICOM Operation Scope. The Resource Definition itself is described in Table C.1.1 and the YAML file is listed in Annex A. The DICOM Operation Scope is used to define an Application's capabilities. These capabilities are defined by a code system which the Platform

and Application use to process jobs. The DICOM standard definition reference for the DICOM Operation Scope is listed below. The fully defined resource definition is located in Annex A Section 1.1.2.

```
apiVersion: standard.oam.dev/v1alpha3
kind: ScopeDefinition
metadata:
  name: DicomOperationScope
spec:
  definitionRef:
    name: schema.dicomoperationscope.oam.dev
```

Table C.1.1 DICOM Operation Scope Attributes

Attribute	Type	Required	Default Value	Description
type	string	Y		Must be workitem, route, or invoked. Details the expected responsibility of the Platform as it relates to the Application.
vendor	string	N		The vendor of the Application
version	string	N		The version of the Application
code	string	N		A reference to a code defined by a terminology system. Required for workitem type.
codeSystem	string	N		The identification of the code system that defines the meaning of the symbol in the code. Required for workitem type.
sopClasses	array	N		SOP Classes as defined by DICOM Part 4. This is represented as an array of SOP Classes for which the Application capabilities are valid. When an SOP Class is not present the platform may assign work to this application scope at its discretion
friendlyName	string	N		Name given to the Work Item to understand its basic purpose.

C.1.1.1 DICOM Operation Types

Workitem – Operations defined by a shared code system between the Application and Platform. Work items are managed by the Platform and are usually in response to job requests which may be in the form of DICOM UPS messages.

Route – Data will be sent to the Application and once successfully transferred the platform can consider the request complete.

Invoked – An Application that is invoked by some explicit user action.

C.1.1.2 Example DICOM Operation Scope Definition

An example of a DICOM Operation Scope instance is shown. This is how a properly formatted resource would appear.

```
apiVersion: standard.oam.dev/v1alpha3
```

```
kind: DicomOperationScope
metadata:
   name: example-operation-scope
spec:
   type: workitem
   vendor: MyAIApp
   version: 1.2.1
   code: RDES128
   codeSystem: https://radelement.org
   sopClasses:
   - 1.2.840.10008.5.1.4.1.1.2
   - 1.2.840.10008.5.1.4.1.1.2.1
   - 1.2.840.10008.5.1.4.1.1.4
   - 1.2.840.10008.5.1.4.1.1.4.1
   friendlyName: SubduralHematomaDetection
```

This resource would be referenced in an Application Configuration as seen in this example.

```
apiVersion: core.oam.dev/v1alpha3
kind: Application
metadata:
   name: my-example-dicom-application
spec:
   components:
      - name: example-containerized-dicom-server
        type: ContainerizedWorkload
        settings:
          hostname: myapp.myhospital.org
          ipaddress: 10.5.175.110
        traits:
          - type: auditTrail
            properties:
              syslogUri: https://arr.myhospital.org/applogs/myapp
        scopes:
          - scopeRef:
              apiVersion: standard.oam.dev/v1alpha3
              kind: DicomOperationScope
              name: example-operation-scope
```

D. Components

These attributes provide information about a component definition. They follow the Kubernetes API convention. A component describes a set of container, service, or other executable entity that conforms to the OAM Component Model and implements a distinct operation. Components enable application developers to declare the characteristics of the code they deliver in infrastructure neutral terms. In practice, a simple containerized workload, a Helm chart, or a cloud database may all be modeled as a component.

D.1 Component Specification

Defines the workload that will be used for this Component. Each Component may only have one workload associated with it. Table D.1 provides the schema attributes for a Component.

Table D.1 Component Attributes

Attribute	Type	Required	Default Value	Description
workload	Workload (see Section D.1.1)	Y		Refers to a workload definition by name
schematic	Template (see Section 1.2)	Y		Refers to the workloads resource template

D.1.1 Workload

Defines the specific type of workload and which traits may or may not be applicable to the workload in this Component's implementation. Table D.1.1 provides the schema attributes for the Workload section of a Component.

Table D.1.1 Component Workload Attributes

Attribute	Type	Required	Default Value	Description
definition	Definition (see Section D.1.1.1)	Y		Refers to the type of workload and the version of the schematic for which it is defined
characteristics	[ ]ApplicableTraits (see Section D.1.1.2)	Y		Refer to traits that can or cannot be applied to the workload

D.1.1.1 Definition

Refers to the type of workload and the version of the schematic for which it is defined. Table D.1.1.1 provides the schema attributes for the Definition section of a Workload.

Table D.1.1.1 Workload Definition Attributes

Attribute	Type	Required	Default Value	Description
apiVersion	string	Y		A string that identifies the version of the schema the object should have. For example, the standard types use standard.oam.dev/v1alpha3
kind	string	Y		The type of workload, ContainerizedWorkload, DicomServerWorload or DicomTaskWorkload A declaration about what kind of capability this component relies upon.

D.1.1.2 Applicable Traits

Refer to traits that can or cannot be applied to the workload. For those traits not listed but may be part of the Platform configuration, it is implied that these traits are false. The default value for a listed trait is true if a value is not provided. Table D.1.1.2 provides the schema attributes for the Applicable Traits section of a Workload.

Table D.1.1.2 Workload Applicable Traits Attributes

Attribute	Type	Required	Default Value	Description

app	map[string]boolean	N	true	Defines whether a trait can be applied to the component's workload

D.1.2 Template

This section declares the schematic of a component that could be instantiated as part of an application in the later workflow.
The OAM specification has no opinion or enforcement on the schematic itself as long as a JSON schema equivalent parameter list is defined. Though in order to make the OAM specification implementable, several built-in approaches (cue, helm, kube) are defined as part of the specification itself. For the DICOM standard, kube is used by default.
Table D.1.2 provides the attributes for the schematic section of a Component.

Table D.1 2 Component Schematic Attributes

Attribute	Type	Required	Default Value	Description
schematic	string	Y	kube	Defines which workload definition template is being used. Must be kube, cue or helm.

D.1.3 Example Component Specification

An example of a DICOM Component instance with a Containerized Workload is shown. This is how a properly formatted resource would appear.

```
apiVersion: core.oam.dev/v1alpha3
kind: ComponentDefinition
metadata:
   name: example-containerized-dicom-server
   annotations:
      description: ExampleDicomServer
      version: 1.0.1
spec:
   workload:
      definition:
         apiVersion: core.oam.dev/v1alpha3
         kind: ContainerizedWorkload
      characteristics:
         auditTrail: true
         license: true
         operatorInput: true
         operatorOutput: true
         appCStoreProvider: true
         appCStoreUser: true
   schematic:
      kube:
         template:
            apiVersion: core.oam.dev/v1alpha3
            kind: ContainerizedWorkload
            metadata:
               name: example-containerized-dicom-server
            spec:
               osType: linux
               containers:
               - name: example-containerized-dicom-server
                 image: example/exampledicomserver:1.0.1@sha256:verytrustworthyhash
```

```
      resources:
        cpu:
           required: 1.0
        memory:
           required: 100MB
        ports:
        - name: liveness
          containerPort: 8080
        - name: readiness
          containerPort: 8088
        - name: https
          containerPort: 8443
        livenessProbe:
           httpGet:
              port: 8080
              path: /live
        readinessProbe:
           httpGet:
              port: 8088
              path: /ready
      env:
        - name: hostname
          value: my.server.com
        - name: ipaddress
          value: 192.168.1.2
parameters:
- name: image
  required: false
  fieldPaths:
  - "spec.containers[0].image"
- name: livenessPort
  required: false
  fieldPaths:
  - "spec.containers[0].resources.port[0].containerPort"
  - "spec.containers[0].resources.livenessProbe.httpGet.port"
- name: livenessPort
  required: false
  fieldPaths:
  - "spec.containers[0].resources.port[1].containerPort"
  - "spec.containers[0].resources.readinessProbe.httpGet.port"
- name: httpsPort
  required: false
  fieldPaths:
  - "spec.containers[0].resources.port[2].containerPort"
- name: hostname
  required: false
  fieldPaths:
  - "spec.containers[0].env.value[0]"
- name: ipaddress
  required: false
  fieldPaths:
  - "spec.containers[0].env.value[1]"
```

D.2 Component Workloads

Component workloads are defined in DICOM as Kubernetes objects. For container schematics these are noted as kube. OAM supports containers defined as cue templates and parameters or as a helm chart as well. The template used is noted in the schematics section of the component specification. (see D.1.2)

D.2.1 Containerized Workload Type Settings

Used to specify the settings of a container that needs instantiation by the Platform. This portion of the specification is used directly from OAM in DICOM.

The ContainerizedWorkload is a Serverless Container style workload definition that could be referenced as the schema for long-running containerized workload types for runtime platforms like Azure ACI, AWS Fargate or simple stateless workload of Kubernetes.

Note: it's by design that ContainerizedWorkload schema is NOT equivalent to Kubernetes Pod specification. As a schema for serverless style workload, ContainerizedWorkload intends to focus on developer facing primitives only and be self-contained so developers don't need to define objects like ConfigMap or Secret. Also, it exposes container ports by default.

Table D.2.1 provides the schematic specification for ContainerizedWorkload.

Table D.2.1 Containerized Workload Attributes

Attribute	Type	Required	Default Value	Description
osType	string	N		The OS required to host (all of) the component's containers (since containers share a kernel with the underlying host). Possible values include: • Linux • Windows Default can be none and let the runtime decide where to place the component.
arch	string	N		The CPU architecture required to host (all of) the component's containers (since containers share physical hardware with the underlying host). Possible values include: • i386 • amd64 • arm • arm64 Default can be none and let the runtime chose architecture.
containers	[ ]Container (see Section D.2.1.1)	Y		The OCI container(s) that implement the component. Runtime instance of an image.
parameters	Parameter (see Section D.3)	N		The component's configuration options. The parameters that can be adjusted during operation time.

D.2.1.1 Container

This section describes the runtime configuration necessary to run a containerized workload for this component. Table D.2.1.1 provides the attributes for the container section of a Containerized Workload.

Table D.2.1.1 Container Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The container's name. Must be unique per component.
image	string	Y		A path-like or URI-like representation of the location of an OCI image. Where applicable, this MAY be prefixed with a registry address, SHOULD be suffixed with a tag.
resources	Resources (see Section D.2.1.1.1)	Y		Resources required by the container.
cmd	[ ]string	N		Container run array
args	[ ]string	N		Used to pass arguments to the container. The container image's CMD is used if this is not provided.
env	[ ]Env (see Section D.2.1.1.2)	N		Environment variables for the container.
config	[ ]ConfigFile (see Section D.2.1.1.3)	N		Locations to write configuration as files accessible within the container
ports	[ ]Port (see Section 2.1.1.4)	N		Ports exposed by the container.
livenessProbe	HealthProbe (see Section 2.1.1.5)	N		Instructions for assessing whether the container is alive.
readinessProbe	HealthProbe (see Section 2.1.1.5)	N		Instructions for assessing whether the container is in a suitable state to serve traffic.
imagePullSecret	string	N		Key that can be used to retrieve the credentials for pulling this secret.

The details of the way that a runtime takes imagePullSecret and loads credentials is left to the OAM runtime implementation. For example, a Kubernetes implementation may treat this as a key that can be loaded from a secret. While it is not required, it is RECOMMENDED that image names be suffixed with a digest in OCI format. The digest may be used to compute the integrity of the image. example/foobar@sha256:72e996751fe42b2a0c1e6355730dc2751ccda50564fec929f76804a6365ef5ef.

The name field is required and must be 63 characters or less, beginning and ending with an alphanumeric character ([a-z0-9A-Z]) with dashes (-), underscores (_), dots (.), and alphanumerics between.

D.2.1.1.1 Resources

Resources describe compute resources attached to a runtime. Table D.2.1.1.1 provides the attributes for the resources section of a container.

Table D.2.1.1.1 Container Resources Attributes

Attribute	Type	Required	Default Value	Description
cpu	CPU (see Section D.2.1.1.1.1)	Y		Specifies the attributes of the cpu resource required for the container.
memory	Memory (see Section D.2.1.1.1.2)	Y		Specifies the attributes of the memory resource required for the container.
gpu	GPU (see Section D.2.1.1.1.3)	N		Specifies the attributes of the gpu resources required for the container.
volumes	[ ]Volume (see Section D.2.1.1.1.4)	N		Specifies the attributes of the volumes that the container uses.
extended	[ ]ExtendedResource (see Section D.2.1.1.1.5)	N		Implementation-specific extended resource requirements

For any resource that cannot be satisfied by the underlying platform, the platform MUST return an error and cease deployment. A resource is considered a requirement, and failure to meet that requirement means the runtime MUST NOT deploy the application. For example, if an application requests 1P of memory, and that amount of memory is not available, the application deployment must fail. Likewise, if an application requires 1 gpu, and the runtime does not provide gpus, the application deployment MUST fail.

D.2.1.1.1.1 CPU

Table D.2.1.1.1.1 provides the attributes for the CPU section of container resources.

Table 2.1.1.1.1 Container Resource CPU Attributes

Attribute	Type	Required	Default Value	Description
required	double	Y		The minimum number of logical cpus required for running this container.

D.2.1.1.1.2 Memory

Table D.2.1.1.1.2 provides the attributes for the memory section of container resources.

Table 2.1.1.1.2 Container Resource Memory Attributes

Attribute	Type	Required	Default Value	Description

required	string	Y		The minimum amount of memory required for running this container. The value should be a positive integer with/without unit suffix: P, T, G, M, K. If no unit is given, it defaults to 'bytes'.

D.2.1.1.1.3 GPU

Table D.2.1.1.1.3 provides the attributes for the GPU section of container resources.

Table 2.1.1.1.3 Container Resource GPU Attributes

Attribute	Type	Required	Default Value	Description
required	double	Y		The minimum number of logical gpus required for running this container.

D.2.1.1.1.4 Volume

Volume describes name, a location to mount the volume, along with access mode (such as read/write or read-only) and sharing policy for the mount. It also describes the underneath disk attributes needed by the volume. The format of the path is specific to the operating system of the consuming component, though implementations SHOULD provide support for UNIX-like path representations. Table D.2.1.1.1.4 provides the attributes for the volume section of container resources.

Table 2.1.1.1.4 Container Resource Volume Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		Specifies the name used to reference the path.
mountPath	string	Y		Specifies the actual mount path in the filesystem.
accessMode	string	N	RW	Specifies the access mode. Allowed values are RW (read/write) and RO (read-only).
sharingPolicy	string	N	Exclusive	The sharing policy for the mount, indicating if it is expected to be shared or not. Allowed values are Exclusive and Shared.
disk	Disk (see Section D.2.1.1.1.4.1)	N		Specifies the attributes of the underneath disk resources required by the volume.

Example:

```
name: "configuration"
mountPath: /etc/config
accessMode: RO
sharingPolicy: Shared
disk:
  required: "2G"
```

ephemeral: n

The above requires that a read-only volume be mounted at the path /etc/config, backed by a volume that provides at least 2G of non-ephemeral storage.

D.2.1.1.1.4.1 Disk

The disk specifies the attributes of the disk used by the volume. It describes information such as minimum disk size and the disk is ephemeral or not. Ephemeral disk indicates the component requires minimum disk size on the node to run it. For example, image processing component may require a larger cache on the node to run could use ephemeral disk. When ephemeral disk is set to false, it indicates external disk will be used. Table D.2.1.1.1.4.1 provides the attributes for the disk section of a volume.

Table 2.1.1.1.4.1 Volume Disk Attributes

Attribute	Type	Required	Default Value	Description
required	string	Y		The minimum disk size required for running this container. The value should be a positive value, greater than zero.
ephemeral	boolean	N		Specifies whether external disk needs to be mounted or not.

D.2.1.1.1.5 ExtendedResource

An extended resource is a declaration of a resource requirement for an implementation-specific resource. For example, OAM-compliant platforms may expose special hardware. This field allows containers to indicate that such special offerings are required in order for the containers to operate. Table D.2.1.1.1.5 provides the attributes for the extended resource section of container resources.

Table 2.1.1.1.5 Container Resource Extended Resource Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The name of the resource, as a Group/Version/Kind
required	string	Y		The required condition.

The name field MUST be a group/version/kind identifying the specific resource.

Example:

```
extended:
- name: ext.example.com/v1.MotionSensor
  required: "1"
- name: ext.example.com/v2beta4.ServoModel
  required: z141155-t100
```

If the named extended resource is not available for any reason, implementations MUST return an error when a component instance is created.

D.2.1.1.2 Env

Env describes an environment variable as a name/value pair of strings. Table D.2.1.1.2 provides the attributes for the environment variable.

Table D.2.1.1.2 Environmental Variable Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The environment variable name. Must be unique per container.
value	string	N		The environment variable value.

The name field must be composed of valid Unicode letter and number characters, as well as _ and -.

Example:

```
env:
  - name: "ADMIN_USER"
    value: "admin"  # This is a literal value
```

D.2.1.1.3 ConfigFile

ConfigFile describes a path to a file available within the container, as well as the data that will be written into that file. This provides a way to inject configuration files into a container. Table D.2.1.1.3 provides the attributes for the configuration file.

Table D.2.1.1.3 Configuration File Attributes

Attribute	Type	Required	Default Value	Description
path	string	Y		An absolute path within the container.
value	string	N		The data to be written into the file at the specified path. If this is not supplied, fromParam must be supplied
fromParam	string	N		The parameter whose value should be written into this file as a value

The path field must contain a path that abides by the pathing rules of the underlying operating system. If a relative path is given, implementations MUST assume the path is relative to the root directory of the container. Implementations MAY produce an error if using such a path would violate security measures or path layout requirements. If both fromParam and value are specified, fromParam MUST take precedence, even if the parameter value is an empty value. If neither is specified, the runtime MUST produce an error.

Example:

```
config:
  - path: "/etc/access/default_user.txt"
    value: "admin"  # This is a literal value
  - path: "/var/run/db-data"
    fromParam: "sourceData" # The value will be read from the parameter whose name is `sourceData`
```

D.2.1.1.4 Port

Table D.2.1.1.4 provides the attributes for the port section of a container.

Table D.2.1.1.4 Container Port Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		A descriptive name for the port. Must be unique per container.
containerPort	integer	Y		The port number. Must be unique per container.
Protocol	string	N	TCP	Indicates the transport layer protocol used by the server listening on the port. Valid values are TCP and UDP.

The name field must be lowercase alphabetical characters as present in the ASCII character set (0061-007A).

D.2.1.1.5 HealthProbe

Health Probe describes how a probing operation is to be executed as a way of determining the health of a component. Table D.2.1.1.5 provides the attributes for Health Probe.

Table D.2.1.1.5 Health Probe Attributes

Attribute	Type	Required	Default Value	Description
exec	Exec (see Section D.2.1.1.5.1)	N		Instructions for assessing container health by executing a command. Either this attribute or the httpGet attribute or the tcpSocket attribute MUST be specified. This attribute is mutually exclusive with both the httpGet attribute and the tcpSocket attribute.
httpGet	HTTPGet (see Section D.2.1.1.5.2)	N		Instructions for assessing container health by executing an HTTP GET request. Either this attribute or the exec attribute or the tcpSocket attribute MUST be specified. This attribute is mutually exclusive with both the exec attribute and the tcpSocket attribute.
tcpSocket	TCPSocket (see Section D.2.1.1.5.3)	N		Instructions for assessing container health by probing a TCP socket. Either this attribute or the exec attribute or the httpGet attribute MUST be specified. This attribute is mutually exclusive with both the exec attribute and the httpGet attribute.

initialDelaySeconds	integer	N	0	Number of seconds after the container is started before the first probe is initiated.
periodSeconds	integer	N	10	How often, in seconds, to execute the probe.
timeoutSeconds	integer	N	1	Number of seconds after which the probe times out.
successThreshold	integer	N	1	Minimum consecutive successes for the probe to be considered successful after having failed.
failureThreshold	integer	N	3	Number of consecutive failures required to determine the container is not alive (liveness probe) or not ready (readiness probe).

D.2.1.1.5.1 Exec

Table D.2.1.1.5.1 provides the attributes for the command execution section of Health Probe.

Table D.2.1.1.5.1 Health Probe Command Execution Attributes

Attribute	Type	Required	Default Value	Description
command	[ ]string	Y		A command to be executed inside the container to assess its health. Each space delimited token of the command is a separate array element. Commands exiting 0 are considered to be successful probes, whilst all other exit codes are considered failures.

D.2.1.1.5.2 HTTPGet

Table D.2.1.1.5.2 provides the attributes for the HTTP Get section of Health Probe.

Table D.2.1.1.5.2 Health Probe HTTP Get Attributes

Attribute	Type	Required	Default Value	Description
path	string	Y		The endpoint, relative to the port, to which the HTTP GET request should be directed.
port	integer	Y		The TCP socket within the container to which the HTTP GET request should be directed.
httpHeaders	[ ]HTTPHeader (see Section D.2.1.1.5.2.1)	N		Optional HTTP headers.

D.2.1.1.5.2.1 HTTPHeader

Table D.2.1.1.5.2.1 provides the attributes for the HTTP Header section of HTTP Get.

Table D.2.1.1.5.2.1 HTTP Get HTTP Header Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		An HTTP header name. This must be unique per HTTP GET-based probe.
value	string	Y		An HTTP header value.

Both name and value must abide by the HTTP/1.1 specification for valid header values

D.2.1.1.5.3 TCPSocket

Table D.2.1.1.5.3 provides the attributes for the TCP Socket section of Health Probe.

Table D.2.1.1.5.3 Health Probe TCP Socket Attributes

Attribute	Type	Required	Default Value	Description
port	integer	Y		The TCP socket within the container that should be probed to assess container health.

Port must be an integer value greater than 0

D.2.1.2 Containerized Workload Example

An example of a DICOM Component instance with a Containerized Workload is shown. This is how a properly formatted resource would appear.

```
apiVersion: core.oam.dev/v3
kind: Component
metadata:
  name: example-containerized-dicomweb-server
  annotations:
    description: ExampleDicomWebServer
    version: 1.0.1
spec:
  workload:
    type: ContainerizedWorkload
    characteristics:
      auditTrail: true
      license: true
      timeSync: true
      appWadoUser: true
      appStowProvider: true
      appStowUser: true
      userIdentitySecurity: true
  schematic:
    kube:
      template:
        apiVersion: core.oam.dev/v1alpha3
        kind: ContainerizedWorkload
        metadata:
          name: example-containerized-dicomweb-server
        spec:
          osType: linux
          containers:
```

```
  - name: example-containerized-dicomweb-server
    image: example/exampledicomwebserver:1.0.1@sha256:verytrustworthyhash
    resources:
      cpu:
        required: 1.0
      memory:
        required: 100MB
      ports:
      - name: http
        containerPort: 8080
      - name: https
        containerPort: 8443
      livenessProbe:
        httpGet:
          port: 8086
          path: /healthz
      readinessProbe:
        httpGet:
          port: 8088
          path: /healthz
      env:
      - name: hostname
        value: my.server.com
      - name: ipaddress
        value: 192.168.1.2
  parameters:
  - name: image
    required: false
    fieldPaths:
    - "spec.containers[0].image"
  - name: hostname
    required: false
    fieldPaths:
    - "spec.containers[0].env.value[0]"
  - name: ipaddress
    required: false
    fieldPaths:
    - "spec.containers[0].env.value[1]"
```

D.2.2 DICOM Server Workload Type Settings

Used to describe persistent network services or APIs that need not be instantiated by the Platform.

Table D.2.2 provides the schematic specification for DICOM Server Workload. The Definition Reference for DICOM Server Workload is also provided as reference. The Definition itself can be found in Annex A.

```
    apiVersion: standard.oam.dev/v1alpha3
    kind: WorkloadDefintion
    metadata:
      name: DicomServerWorkload
    spec:
      definitionRef:
        name: schema.dicomserverworkload.oam.dev
```

Table D.2.2 DICOM Server Workload Attributes

Attribute	Type	Required	Default Value	Description
host	string	Y		Host name or ip of the server. Note in workloads where the host name or ip may also be specified elsewhere in the settings and either is to be made mutable the parameter fieldPath for both locations needs to be specified
livenessProbe	HealthProbe (see Section 2.1.1.5)	N		Instructions for assessing whether the server is alive.
readinessProbe	HealthProbe (see Section 2.1.1.5)	N		Instructions for assessing whether the server is in a suitable state to serve traffic.

D.2.2.1 DICOM Server Workload Example

An example of a DICOM Component instance with a DICOM Server Workload is shown. This is how a properly formatted resource would appear.

```
apiVersion: standard.oam.dev/v3
kind: Component
metadata:
  name: example-dicom-server
  annotations:
    description: ExampleDicomServer
    version: 1.2.1
spec:
  workload:
    type: DicomServerWorkload
    characteristics:
      auditTrail: true
      timeSync: true
      jobTimeout: true
      appCStoreUser: true
      appCStoreProvider: true
      userIdentitySecurity: true
  schematic:
    kube:
      template:
        apiVersion: standard.oam.dev/v1alpha3
        kind: DicomServerWorkload
        metadata:
          name: example-dicom-server
        spec:
          host: mydicomserver.myhospital.org
          livenessProbe:
            httpGet:
              port: 8086
              path: /healthz
          readinessProbe:
            httpGet:
              port: 8088
              path: /healthz
```

D.2.3 DICOM Task Workload Type Settings

Used to describe parameterized executables which can be local or remote, if environment is hard to reproduce, it can be created on a remote server and called from there.

Table D.2.3 provides the schematic specification for DICOM Task Workload. The Definition Reference for DICOM Task Workload is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: WorkloadDefintion
metadata:
  name: DicomTaskWorkload
spec:
  definitionRef:
    name: schema.dicomtask workload.oam.dev
```

Table D.2.3 DICOM Task Workload Attributes

Attribute	Type	Required	Default Value	Description
exec	Command (see Section D.2.3.1)	Y		The path or uri to the executable.
env	[ ]Env (see Section D.2.1.1.2)	N		Environment variables. For Task Workload types environmental variables such as operating system or runtime component requirements should be specified here.

D.2.3.1 Command

Table D.2.3.1 provides the attributes for the command section of Task Workload.

Table D.2.3.1 Task Workload Command Attributes

Attribute	Type	Required	Default Value	Description
command	[ ]string	Y		A command to be executed. Each command will be executed sequentially. Commands exiting 0 are considered successful

D.2.3.2 Example DICOM Task Workload

An example of a DICOM Component instance with a DICOM Task Workload is shown. This is how a properly formatted resource would appear.

```
apiVersion: standard.oam.dev/v3
kind: Component
metadata:
  name: example-dicom-task
  annotations:
    description: ExampleDicomTask
    version: 2.2.1
spec:
```

```
  workload:
    type: DicomTaskWorkload
    characteristics:
      jobTimeout: true
      operatorInput: true
      operatorOutput: true
  schematic:
    kube:
      template:
        apiVersion: standard.oam.dev/v1alpha3
        kind: DicomTaskWorkload
        metadata:
          name: example-dicom-task
        spec:
          exec:
            command:
            - \\myshare.mynetwork.org\programs\exampletask\task.jar
          env:
          - name: jreVersion
            value: 1.8
```

D.3 Parameter

A Parameter is an attribute in the specification that is made mutable, this can be required to be defined in the Application Configuration or can have a default value which will be used if a value is not specified. Default values are placed in the field path value when the resource is defined. The use of fromParam as a value specifies the parameter whose value should be written into the resource fieldPath as a value. The parameter name is used in the Application Configuration to specify the settings of the parameter value. Table D.3 provides the schematic specification for Parameter.

Table D.3 Parameter Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The parameter's name. Must be unique per component.
description	string	N		A description of the parameter.
fieldPaths	string	Y		JSON field paths.
required	boolean	N	false	Whether a value must be provided when authoring an applicationConfiguration including this component.

Example:

```
parameters:
- name: image
  description: the uri of the image file
  fieldPaths:
  - "spec.containers[0].image"
  required: true
```

E. Traits

DICOM Trait definitions are broken down into 3 sections. These traits are for Control, Entrypoints and Security. These traits definition are Standard definitions of OAM for a DICOM compliant Platform.

E.1 Control Traits

The Control traits JobTimeout, AuditTrail, and TimeSync are reserved names with standardized schemas defined within this standard. This group of traits include those that control the overall actions of a component.

E.1.1 DICOM Job Timeout Trait Schematic

Used to set a timeout for a job. Once the timeout is exceeded the status will become failed with reason timeout exceeded. If the Platform is controlling the container or task the Platform may terminate the instance. Table E.1.1 provides the attributes for the DICOM Job Timeout Trait. The Definition Reference for Job Timeout is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: JobTimeout
spec:
  definitionRef:
    name: schema.jobtimeout.oam.dev
```

Table E.1.1 DICOM Job Timeout Trait Attributes

Attribute	Type	Required	Default Value	Description
seconds	integer	Y	30	Used to set a timeout for a job.

Example Job Timeout Trait Usage

```
traits:
  - name: jobTimeout
    properties:
      seconds: 30
```

E.1.2 DICOM Audit Trail Trait Schematic

For information on Audit Trial Message formats, schemas and coding refer to PS3.15 Annex A. The Trait specified in this section provides information on audit trail endpoints for connectivity only, not for content. Table E.1.2 provides the attributes for the DICOM Audit Trail Trait. The Definition Reference for Audit Trail is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AuditTrail
spec:
  definitionRef:
    name: schema.audittrail.oam.dev
```

Table E.1.2 DICOM Audit Trail Trait Attributes

Attribute	Type	Required	Default Value	Description
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
syslogUri	string	Y		Identifies the resource by name at the specified location or URL

Example Audit Trail Trait Usage

```
traits:
  - name: auditTrail
    properties:
      tlsVersion: 1.2
      tlsCertificate: server.app.com
      tlsPassword: myAppSecretPa$$w0rd
      tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      syslogUri: \\server.mysite.org\auditlog\myapp
```

E.1.3 DICOM Time Sync Trait Schematic

To ensure events are in synchronization, the use of a common time synchronization server is commonly used. Table E.1.3 provides the attributes for the DICOM Time Sync Trait. The Definition Reference for Time Sync is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: TimeSync
spec:
  definitionRef:
    name: schema.timesync.oam.dev
```

Table E.1.3 DICOM Time Sync Trait Attributes

Attribute	Type	Required	Default Value	Description
ntpTimeServer	string	Y		Server address used for time synchronization.

Example Time Sync Trait Usage

```
traits:
  - name: TimeSync
    properties:
      ntpTimeServer: time.nist.gov
```

E.2 Entrypoint Traits

The Entrypoint traits AppCStoreProvider, AppCStoreUser, AppWadoUser, AppStowProvider, AppStowUser, OperatorInput, OperatorOutput, RestApiProvider and RestApiProvider TimeSync are reserved names with standardized schemas defined within this standard and include optional security attributes. Entrypoint traits describe how data is accepted and results are emitted for a Component. These can be thought of as SCU-SCP interface methods such as DIMSE storage services, web service APIs or mapped input/output directories.

E.2.1 DICOM Application C-Store Provider Trait Schematic

This Entrypoint Trait is used to specify DICOM DIMSE endpoint information for an application acting as a C-Store SCP. Note the hostname will be provided as part of the Workload definition. Table E.2.1 provides the attributes for the DICOM Application C-Store Provider Trait. The Definition Reference for Application C-Store Provider is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppCStoreProvider
spec:
  definitionRef:
    name: schema.appcstoreprovider.oam.dev
```

Table E.2.1 DICOM Application C-Store Provider Trait Attributes

Attribute	Type	Required	Default Value	Description
aet	string	Y		AET the entrypoint uses for inbound data transfers, the entrypoint's storage SCP AET
port	integer	Y		Port the entrypoint uses for inbound data transfers. Note in workloads where the port may also be specified elsewhere in the settings and is to be made mutable the parameter fieldPath for both locations needs to be specified
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
scuAet	string	N		AET of SCU if required by SCP
scuHost	string	N		Host name or ip of SCU if required by SCP

Example Application C-Store Provider Trait Usage

```
traits:
  - name: AppCStoreProvider
    properties:
      aet: MYAPPAET
      port: 104
```

```
tlsVersion: 1.2
tlsCertificate: server.app.com
tlsPassword: myAppSecretPa$$w0rd
tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
scuAet: YOURAPPAET
scuHost: yourapp.yourdomain.org
```

E.2.2 DICOM Application C-Store User Trait Schematic

This Entrypoint Trait is used to specify DICOM DIMSE endpoint information for an application acting as a C-Store SCU. Table E.2.2 provides the attributes for the DICOM Application C-Store User Trait. The Definition Reference for Application C-Store User is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppCStoreUser
spec:
  definitionRef:
    name: schema.appcstoreuser.oam.dev
```

Table E.2.2 DICOM Application C-Store User Trait Attributes

Attribute	Type	Required	Default Value	Description
destAet	string	Y		AET the entrypoint uses for inbound data transfers, the entrypoint's storage SCP AET
destPort	integer	Y		Port the entrypoint uses for inbound data transfers. Note in workloads where the port may also be specified elsewhere in the settings and is to be made mutable the parameter fieldPath for both locations needs to be specified
destHost	string	Y		Host name or ip the entrypoint uses for inbound data transfers. Note in workloads where the host name or ip may also be specified elsewhere in the settings and either is to be made mutable the parameter fieldPath for both locations needs to be specified
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Example Application C-Store User Trait Usage

```
traits:
  - name: AppCStoreUser
    properties:
```

```
destAet: YOURAET
destPort: 104
destHost:  yourapp.yourdomain.org
tlsVersion: 1.2
tlsCertificate: server.app.com
tlsPassword: myAppSecretPa$$w0rd
tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

E.2.3 DICOM Application WADO User Trait Schematic

This Entrypoint Trait is used to specify DICOM WADO endpoint information for an application acting as a user. Table E.2.3 provides the attributes for the DICOM Application WADO User Trait. The Definition Reference for Application WADO User is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppWadoUser
spec:
  definitionRef:
    name: schema.appwadouser.oam.dev
```

Table E.2.3 DICOM Application WADO User Trait Attributes

Attribute	Type	Required	Default Value	Description
resourceUri	string	Y		Identifies a resource via a representation of its primary access mechanism.
acceptHeaders	string	Y		Section 9.1.2.2.1. The value of this parameter, if present, shall be either application/dicom, or one or more of the Rendered Media Types.
annotation	array	N		Section 8.3.5.1.2 May be patient and/or technique. Patient indicates that the rendered images shall be annotated with patient information. Technique indicates that the rendered images shall be annotated with information about the procedure that was performed.
quality	integer	N		Section 8.3.5.1.3. Is an unsigned integer between 1 and 100 inclusive, with 100 being the best quality.
viewport	string	N		Section 8.3.5.1.3 vw and vh are positive integers specifying the width and height, in pixels, of the rendered image or video. Both values are required. sx and sy are decimal numbers whose absolute values specify, in pixels, the top-left corner of the region of the source image(s) to

				be rendered. If either sx or sy is not specified, it defaults to 0. A value of 0,0 specifies the top-left corner of the source image(s). sw and sh are decimal numbers whose absolute values specify, in pixels, the width and height of the region of the source image(s) to be rendered. If sw is not specified, it defaults to the right edge of the source image. If sh is not specified, it defaults to the bottom edge of the source image. If sw is a negative value, the image is flipped horizontally. If sh is a negative value, the image is flipped vertically.
window	string	N		Section 8.3.5.1.4 Center, width, function – center is a decimal number containing the window-center value. Width is a decimal number containing the window-width value and function is one of the following 'linear', 'linear-exact' or 'sigmoid'
iccProfile	string	N		Section 8.3.5.1.5 Must be 'no', 'yes', 'srgb', 'adobergb', or 'rommrgb'.
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Example Application WADO User Trait Usage

```
traits:
  - name: AppWadoUser
    properties:
      resourceUri: https://mydicomserver.mysite.org/wado
      acceptHeaders: application/dicom
      tlsVersion: 1.2
      tlsCertificate: server.app.com
      tlsPassword: myAppSecretPa$$w0rd
      tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

E.2.4 DICOM Application STOW Provider Trait Schematic

This Entrypoint Trait is used to specify DICOM STOW endpoint information for an application acting as a provider. Note the hostname will be provided as part of the Workload definition which should correlate

with the resource URI. Table E.2.4 provides the attributes for the DICOM Application STOW Provider Trait. The Definition Reference for Application STOW Provider is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppStowProvider
spec:
  definitionRef:
    name: schema.appstowprovider.oam.dev
```

Table E.2.4 DICOM Application STOW Provider Trait Attributes

Attribute	Type	Required	Default Value	Description
resourceUri	string	Y		Identifies a resource via a representation of its primary access mechanism.
contentTypeHeaders	string	Y		Section 8.7.3.5 DICOM Media Type Syntax
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Example Application STOW Provider Trait Usage

```
traits:
  - name: AppStowProvider
    properties:
      resourceUri: https://mydicomserver.mysite.org/stow
      contentTypeHeaders: application/dicom
      tlsVersion: 1.2
      tlsCertificate: server.app.com
      tlsPassword: myAppSecretPa$$w0rd
      tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

E.2.5 DICOM Application STOW User Trait Schematic

This Entrypoint Trait is used to specify DICOM STOW endpoint information for an application acting as a user. Table E.2.5 provides the attributes for the DICOM Application STOW User Trait. The Definition Reference for Application STOW User is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppStowUser
spec:
  definitionRef:
    name: schema.appstowuser.oam.dev
```

Table E.2.5 DICOM Application STOW User Trait Attributes

Attribute	Type	Required	Default Value	Description
destResourceUri	string	Y		Identifies a resource via a representation of its primary access mechanism.
destContentTypeHeaders	string	Y		Section 8.7.3.5 DICOM Media Type Syntax
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Example Application STOW User Trait Usage

```
traits:
  - name: AppStowUser
    properties:
      destResourceUri: https://yourdicomserver.yoursite.org/stow
      destContentTypeHeaders: application/dicom
      tlsVersion: 1.2
      tlsCertificate: server.app.com
      tlsPassword: yourAppSecretPa$$w0rd
      tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
```

E.2.6 DICOM Operator Input Trait Schematic

When using the DICOM Operator Input Trait, data must be populated at the time of job start for task as opposed to a monitored folder. Data Types (SOP classes) are defined as part of the Application's Scope when DICOM files are used as the input type. When used for data types other than DICOM, data types can be listed here, although the content and coordination of formats are outside the scope of this specification and need to be coordinated between the Platform and Application. Applications and their components will have a minimum of read permission granted to an Operator Input. The Operator Input is the path to the data which is to be used to perform the requestion action. This differs from paths specified in a Container Workload as these as specifically used to pass data for job execution.

This Entrypoint Trait is used to specify DICOM Operator Input endpoint information to an application acting as a user. Table E.2.6 provides the attributes for the DICOM Operator Input Trait. The Definition Reference for Operator Input is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: OperatorInput
spec:
  definitionRef:
    name: schema.operatorinput.oam.dev
```

Table E.2.6 DICOM Operator Input Trait Attributes

Attribute	Type	Required	Default Value	Description
path	string	Y		To the folder level, everything in the folder will be ingested
userType	string	N		Must be basic, userIdPasscode, kerberos, or saml
username	string	N		Identification used to access resource if required
passcode	string	N		Passcode used to access resources if required
dataTypes	array	N		When not present MIME types are considered to be DICOM and are configured as part of the scope via SOP classes.
signatureType	string	N		Must be baseRsa, creatorRsa, authorizationRsa, or structuredReportRsa
macAlgorithm	string	N		Used for key-confirmation if required
publicKey	string	N		Used to decrypt data if required
efsAlgorithm	string	N		The symmetric encryption algorithm used
efsKey	string	N		Used to decrypt data if required

Example Operator Input Trait Usage

```
traits:
  - name: OperatorInput
    properties:
      path: \\mynetwork.org\appshare\data\in
```

E.2.7 DICOM Operator Output Trait Schematic

Applications and their components must have a minimum of read/write permission granted to an Operator Output. This allows for writing artifacts as well as verifying their existence and that they are not corrupt. The Operator Output is the path to where the data artifacts are to be placed after the Application performs the requestion action. This differs from paths specified in a Container Workload as these as specifically used to pass data artifacts resulting from job execution.

This Entrypoint Trait is used to specify DICOM Operator Output endpoint information to an application acting as a user. Table E.2.7 provides the attributes for the DICOM Operator Output Trait. The Definition Reference for Operator Output is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: OperatorOutput
spec:
  definitionRef:
    name: schema.operatoroutput.oam.dev
```

Table E.2.7 DICOM Operator Output Attributes

Attribute	Type	Required	Default Value	Description
destPath	string	Y		This is to the folder level
dataTypes	array	Y		This is to ensure output is consistent, what the application must output. DICOM objects to be specified as dicom.
userType	string	N		Must be basic, userIdPasscode, kerberos, or saml
username	string	N		Identification used to access resource if required
passcode	string	N		Passcode used to access resources if required
signatureType	string	N		Must be baseRsa, creatorRsa, authorizationRsa, or structuredReportRsa
macAlgorithm	string	N		Used for key-confirmation if required
publicKey	string	N		Used to decrypt data if required
efsAlgorithm	string	N		The symmetric encryption algorithm used
efsKey	string	N		Used to decrypt data if required

Example Operator Output Trait Usage

```
traits:
  - name: OperatorOutput
    properties:
      destPath: \\mynetwork.org\appshare\data\out
      destTypes: dicom
```

E.2.8 DICOM REST API Provider Trait Schematic

This Entrypoint Trait is used to specify DICOM REST API endpoint information for an application acting as an API Provider. Note the specific API specification is not configured here, as this is just a reference to the API itself. The API may then expose additional entrypoints or services beyond the scope of this specification as part of its own specification. Table E.2.8 provides the attributes for the DICOM REST API

Provider Trait. The Definition Reference for REST API Provider is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: RestApiProvider
spec:
  definitionRef:
    name: schema.restapiprovider.oam.dev
```

Table E.2.8 DICOM REST API Provider Trait Attributes

Attribute	Type	Required	Default Value	Description
restApiName	string	Y		Examples are AcrModelApi, DicomWSDL
restApiVersion	string	Y		A string that identifies the version of the API
resourceUri	string	Y		Identifies a resource via a representation of its primary access mechanism.
uriType	string	Y		Examples are request, liveliness, readiness, log
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
authMethod	string	N		basicAuth, formAuth, clientCertAuth, oAuth, bearerAuth
apiKey	string	N		Key used to connect to the API
accessToken	string	N		The authorization of a specific application
refreshToken	string	N		Used to acquire new access token

Example REST API Provider Trait Usage

```
traits:
  - name: RestApiProvider
    properties:
      restApiName: DicomWSDL
      restApiVersion: 2021b
      resourceUri: https://mydicomserver.mysite.org/wsdlapi
      uriType: request
```

E.2.9 DICOM REST API User Trait Schematic

This Entrypoint Trait is used to specify DICOM REST API endpoint information for an application acting as an API User. Note the specific API specification is not configured here, as this is just a reference to the API itself. The API may then expose additional entrypoints or services beyond the scope of this specification as part of its own specification. Table E.2.9 provides the attributes for the DICOM REST API User Trait. The Definition Reference for REST API User is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: RestApiUser
spec:
  definitionRef:
    name: schema.restapiuser.oam.dev
```

Table E.2.9 DICOM REST API User Trait Attributes

Attribute	Type	Required	Default Value	Description
restApiName	string	Y		Examples are AcrModelApi, DicomWSDL
restApiVersion	string	Y		A string that identifies the version of the API
resourceUri	string	Y		Identifies a resource via a representation of its primary access mechanism.
uriType	string	Y		Examples are request, liveliness, readiness, log
tlsVersion	string	N		Only versions supported in Part 15 are acceptable
tlsCertificate	string	N		Trusted certificate for this communication
tlsPassword	string	N		Password for keystore to access certificate if required
tlsCipherSuite	string	N		Represented as documented in Part 15 Sections B.9 and B.10, for example TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
authMethod	string	N		basicAuth, formAuth, clientCertAuth, oAuth, bearerAuth
apiKey	string	N		Key used to connect to the API
accessToken	string	N		The authorization of a specific application
refreshToken	string	N		Used to acquire new access token

Example REST API User Trait Usage

```
traits:
  - name: RestApiUser
    properties:
      restApiName: DicomWSDL
```

```
restApiVersion: 2021b
resourceUri: https://yourdicomserver.yoursite.org/wsdlapi
uriType: request
```

E.3 Security Traits

The Security traits UserIdentitySecurity and License are reserved names with standardized schemas defined within this standard. These traits are used to provide security of the components and applications themselves.

E.3.1 DICOM User Identity Security Trait Schematic

This Security Trait is used to specify DICOM User Identity Security information for an application when not tied to an Application's Entrypoint. Table E.3.1 provides the attributes for the DICOM User Identity Security Trait. The Definition Reference for User Identity Security is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: UserIdentitySecurity
spec:
  definitionRef:
    name: schema.useridentitysecurity.oam.dev
```

Table E.3.1 DICOM User Identity Security Trait Attributes

Attribute	Type	Required	Default Value	Description
userType	string	Y		Must be basic, userIdPasscode, kerberos, or saml
username	string	N		Identification used to access resource if required
passcode	string	N		Passcode used to access resources if required

Example User Identity Security Definition

```
traits:
  - name: UserIdentitySecurity
    properties:
      userType: basic
      username: mydicomuser
```

E.3.2 DICOM License Trait Schematic

This Security Trait is used to specify DICOM License information for an application. Table E.3.2 provides the attributes for the DICOM License Trait. The Definition Reference for License is also provided as reference. The Definition itself can be found in Annex A.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: License
spec:
  definitionRef:
    name: schema.license.oam.dev
```

Table E.3.2 DICOM License Trait Attributes

Attribute	Type	Required	Default Value	Description
licenseKey	string	Y		Application defined license key string
machineKey	string	N		Machine specific code generated, example MAC or some other machine code

Example Time Sync Trait Usage

```
traits:
  - name: License
    properties:
      licenseKey: dui2p3jdoj28jd
```

F. Application Configuration

This section describes how applications are designed and deployed. The Application Configuration entity defines a list of components that will be instantiated or called once the application is deployed to a Platform. This portion of the specification is used directly from OAM in DICOM.

Users will specify the final parametrization of each component and the traits that are applied to augment their functionality or alter their behavior. Additionally, a scope or set of scopes grouping different subsets of components can be specified.

F.1 Top-Level Attributes of an Application

The top-level attributes of an application define its metadata, version, kind, and specification. Table F.1 provides the attributes for an Application Configuration.

Table F.1 Application Configuration Attributes

Attribute	Type	Required	Default Value	Description
apiVersion	string	Y		A string that identifies the version of OAM schema being used
kind	string	Y		Must be Application
metadata	Metadata (see B.1.1)	Y		Information about the application capabilities
spec	AppSpec (see F.2)	Y		Specification of the application configuration attributes

F.2 Application Specification

The specification of applications defines components to create, traits attached to each component, and a set of scopes the components drop in. Table F.2 provides the attributes for an Application Configuration Specification.

Table F.2 Application Configuration Specification Attributes

Attribute	Type	Required	Default Value	Description

components	[ ]Component (see Section F.2.1)	Y		Component instance definitions

F.2.1 Component

Table F.2.1 provides the attributes for the Component section of an Application Configuration.

Table F.2.1 Application Configuration Component Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The name of the component of which to create an instance.
type	string	Y		A reference to the component definition that will be instantiated by the application.
settings	[ ]ParameterSettings (see Section F.2.1.1)	N		A set of values assigned to the parameters exposed from the component schematic.
traits	[ ]TraitProperties (see Section F.2.1.2)	N		The traits to attach to this component instance.
scopes	[ ]Scopes (see Section F.2.1.3)	N		The scopes to be used in the component. A component joins a scope by referencing it.

In addition to being unique, the component name must follow these naming rules:
The name field is required and must be 63 characters or less, beginning and ending with an alphanumeric character ([a-z0-9A-Z]) with dashes (-), underscores (_), dots (.), and alphanumerics between.

F.2.1.1 Parameter Settings

Values supplied to parameters that are used to override the parameters exposed by other types. Table F.2.1.1 provides the attributes for the Parameter Settings of an Application Configuration Component.

Table F.2.1.1 Component Parameter Setting Attributes

Attribute	Type	Required	Default Value	Description
name	string	Y		The name of the component parameter for which to provide a value.
value	string	Y		The value of the parameter.

F.2.1.2 Trait Properties

Values supplied to parameters that are used to override the parameters exposed by other types. Table F.2.1.2 provides the attributes for the Trait Properties of an Application Configuration Component.

Table F.2.1.2 Component Trait Properties Attributes

Attribute	Type	Required	Default Value	Description
type	string	Y		A reference to the name of trait definition. For one type of trait,

				there could be only one configuration in one component.
properties	Properties (see Section F.2.1.2.1)	Y		The properties values to use this trait.

F.2.1.2.1 Properties

Properties specify the values that are associated with the entity attributes.

When properties are used on Traits, they set the values required by those entities to be instantiated. The structure is determined by the definition reference. It may be a simple value, or it may be a complex object. Properties are validated against the schema appropriate for the Trait. The schemas for all DICOM defined Traits can be found in Section E.

An example usage of traits in an Application Configuration Component is provided as reference.

```
traits:
  - type: auditTrail
    properties:
      syslogUri: https://arr.myhospital.org/applogs/myapp
  - type: license
    properties:
      licenseKey: doiqurp3idqij923d23jd2p9dk
  - type: timeSync
    properties:
      ntpTimeServer: time.nist.gov
```

F.2.1.3 Scopes

The scope section defines the scope into which the component should be deployed. Table F.2.1.3 provides the attributes for Scopes of an Application Configuration Component.

Table F.2.1.3 Component Scope Attributes

Attribute	Type	Required	Default Value	Description
scopeRef	ScopeRef (see Section F.2.1.3.1)	Y		The reference information of the Scope

F.2.1.3.1 Scope Reference

The scope section defines the scope into which the component should be deployed. Table F.2.1.3.1 provides the attributes for Scope Reference of an Application Configuration Component.

Table F.2.1.3.1 Component Scope Reference Attributes

Attribute	Type	Required	Default Value	Description
apiVersion	string	Y		The apiVersion of the Scope
kind	string	Y		The kind of the Scope
name	string	Y		The name of the Scope

F.2.2 Example Application Configuration

An example of an Application Configuration instance with a Component comprised of a Containerized Workload is shown. This is how a properly formatted resource would appear.

```
apiVersion: core.oam.dev/v1alpha3
kind: Application
metadata:
   name: my-example-dicom-application
spec:
   components:
      - name: example-containerized-dicom-server
        type: ContainerizedWorkload
        settings:
          hostname: myapp.myhospital.org
          ipaddress: 10.5.175.110
        traits:
          - type: auditTrail
            properties:
              syslogUri: https://arr.myhospital.org/applogs/myapp
          - type: license
            properties:
              licenseKey: doiqurp3idqij923d23jd2p9dk
          - type: timeSync
            properties:
              ntpTimeServer: time.nist.gov
          - type: operatorInput
            properties:
              path: /home/input/example/
              dataTypes:
              - csv
              userType: userIdPasscode
              username: mydicomuser
              passcode: my$ecretC0de
          - type: appCStoreProvider
            properties:
              aet: MYAPPAET
              port: 104
              tlsVersion: 1.2
              tlsPassword: 66E0E8D044A5D30187E7203279CC98FD95F0ED8F
              tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
          - type: appCStoreUser
            properties:
              destAet: MYHOSPITALVNA
              destPort: 104
              destHost: myvna.myhospital.org
              tlsVersion: 1.2
              tlsPassword: 5F1B5BDDA91826A48C86F942D673C3A2C5DF0F0E
              tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        scopes:
          - scopeRef:
              apiVersion: standard.oam.dev/v1alpha3
              kind: DicomOperationScope
              name: example-operation-scope
```

G. Manifests

The Manifest describes an Application. Per OAM, each Application OAM Manifest can contain multiple components, each of which specifies an implementation mechanism (such as a container or a remote service) and an Entrypoint via which the component interacts. These Entrypoints are described by a Component's Traits.

A Component may be implemented as a network service or URL that is always available. Scaling in these cases is the responsibility of the service and a way to check the service status should be provided. The Manifest may also describe the Service-Object Pair (SOP) classes accepted in the case of DIMSE or some other API, as well as the Service Class Provider (SCP) details of the Application.

Alternatively, a Component may be implemented as a container, executable, virtual machine or other local software component. In such cases, the Manifest will contain parameters to create and configure the Component. Entrypoints may be network-based, use local file systems or other local resources. Components may be transient, in which case scaling is the responsibility of the Platform (i.e. 1 container per request, and once fulfilled the service Component terminates), or persistent, in which case a single Component may fulfil multiple requests.

OAM Manifests are consumed and are registered by the Platform to represent an Applications' capabilities in relation to a known code scheme the Platform uses with its Task Requestors. The Applications know nothing of the request to a Platform from a Task Requestor, they perform a task upon instantiation of the Platform and being provided data at its Entrypoint. It is the responsibility of the Platform to convert a workitem to a job for the Application.

G.1 Manifest Example

An example of a complete Manifest where the Scope Definition and Component Definition referenced in the Application Configuration are provided in a single YAML instance. Although the Scope and Component Definitions can be provided separately to a Platform, and the most likely scenario, this is also an acceptable option. This is how a properly formatted resource would appear.

```
# ----- Scope Definition -----
apiVersion: standard.oam.dev/v1alpha3
kind: DicomOperationScope
metadata:
   name: example-operation-scope
spec:
   type: workitem
   code: RDES128
   codeSystem: https://radelement.org
   sopClasses:
   - 1.2.840.10008.5.1.4.1.1.2
   - 1.2.840.10008.5.1.4.1.1.2.1
   - 1.2.840.10008.5.1.4.1.1.4
   - 1.2.840.10008.5.1.4.1.1.4.1
   friendlyName: SubduralHematomaDetection
# ----- Scope Definition -----

# ----- Containerized Workload Component Definition -----
apiVersion: core.oam.dev/v1alpha3
kind: ComponentDefinition
metadata:
   name: example-containerized-dicom-server
   annotations:
      description: ExampleDicomServer
```

```
    version: 1.0.1
spec:
  workload:
    type: ContainerizedWorkload
    characteristics:
      auditTrail: true
      license: true
      operatorInput: true
      operatorOutput: true
      appCStoreProvider: true
      appCStoreUser: true
  schematic:
    kube:
      template:
        apiVersion: core.oam.dev/v1alpha3
        kind: ContainerizedWorkload
        metadata:
          name: example-containerized-dicom-server
        spec:
          osType: linux
          containers:
          - name: example-containerized-dicom-server
            image: example/exampledicomserver:1.0.1@sha256:verytrustworthyhash
            resources:
              cpu:
                required: 1.0
              memory:
                required: 100MB
              ports:
              - name: liveness
                containerPort: 8080
              - name: readiness
                containerPort: 8088
              - name: https
                containerPort: 8443
              livenessProbe:
                httpGet:
                  port: 8080
                  path: /live
              readinessProbe:
                httpGet:
                  port: 8088
                  path: /ready
            env:
              - name: hostname
                value: my.server.com
              - name: ipaddress
                value: 192.168.1.2
        parameters:
        - name: image
          required: false
          fieldPaths:
          - "spec.containers[0].image"
        - name: livenessPort
          required: false
          fieldPaths:
```

```
            - "spec.containers[0].resources.port[0].containerPort"
            - "spec.containers[0].resources.livenessProbe.httpGet.port"
          - name: livenessPort
            required: false
            fieldPaths:
            - "spec.containers[0].resources.port[1].containerPort"
            - "spec.containers[0].resources.readinessProbe.httpGet.port"
          - name: httpsPort
            required: false
            fieldPaths:
            - "spec.containers[0].resources.port[2].containerPort"
          - name: hostname
            required: false
            fieldPaths:
            - "spec.containers[0].env.value[0]"
          - name: ipaddress
            required: false
            fieldPaths:
            - "spec.containers[0].env.value[1]"
# ----- Containerized Workload Component Definition -----

# ----- Application Definition -----
apiVersion: core.oam.dev/v1alpha3
kind: Application
metadata:
  name: my-example-dicom-application
spec:
  components:
    - name: example-containerized-dicom-server
      type: ContainerizedWorkload
      settings:
        hostname: myapp.myhospital.org
        ipaddress: 10.5.175.110
      traits:
        - type: auditTrail
          properties:
            syslogUri: https://arr.myhospital.org/applogs/myapp
        - type: license
          properties:
            licenseKey: doiqurp3idqij923d23jd2p9dk
        - type: timeSync
          properties:
            ntpTimeServer: time.nist.gov
        - type: operatorInput
          properties:
            path: /home/input/example/
            dataTypes:
            - csv
            userType: userIdPasscode
            username: mydicomuser
            passcode: my$ecretC0de
        - type: appCStoreProvider
          properties:
            aet: MYAPPAET
            port: 104
            tlsVersion: 1.2
```

```
          tlsPassword: 66E0E8D044A5D30187E7203279CC98FD95F0ED8F
          tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        - type: appCStoreUser
          properties:
          destAet: MYHOSPITALVNA
          destPort: 104
          destHost: myvna.myhospital.org
          tlsVersion: 1.2
          tlsPassword: 5F1B5BDDA91826A48C86F942D673C3A2C5DF0F0E
          tlsCipherSuite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      scopes:
        - scopeRef:
          apiVersion: standard.oam.dev/v1alpha3
          kind: DicomOperationScope
          name: example-operation-scope
# ----- Application Definition -----
```

H. Registration

Registration involves the process by which Platforms become aware of Applications. These Applications can be invoked through instantiation, API, executable or a combination of methods. Applications need a Platform that supports its Components as well as its Trait requirements. Platforms should publish the Components and Traits types they support or can proxy as part of their DICOM Conformance Statement. Platforms should also publish the code system(s) that are supported to describe Application capabilities as this needs to be mapped or presented as part of the Application Scope Reference if they wish to automatically register Applications.

H.1 Registration Resources

Platforms may choose to have a manual configuration file or interface for users to enter the pertinent information from the Application Manifest. Should a Platform provide registration services for Application Manifest POST requests, DICOM resources should follow the resource representations defined in DICOM Part 18 Section 7.2. Table H.1 provides the acceptable resource paths for providing this as a web service.

Table H.1 Web Service Resource Paths

Resource Path	Contents
/applications	A Manifest registration resource endpoint for Application Configurations
/scopes	A Manifest registration resource endpoint for Scopes
/traits	A Manifest registration resource endpoint for Traits
/components	A Manifest registration resource endpoint for Components

The format for making a web service POST request is as follows:

POST: {server}/{resource path}/{resource name}

Example storage of unique resource instance:

```
curl --location --request POST 'https://myplatform.mysite.org/scopes/example-operation-scope' \
--header 'Content-Type: text/plain' \
--data-raw 'apiVersion: standard.oam.dev/v1alpha3
kind: DicomOperationScope
metadata:
```

```
    name: example-operation-scope
spec:
    type: workitem
    code: RDES128
    codeSystem: https://radelement.org
    sopClasses:
    - 1.2.840.10008.5.1.4.1.1.2
    - 1.2.840.10008.5.1.4.1.1.2.1
    - 1.2.840.10008.5.1.4.1.1.4
    - 1.2.840.10008.5.1.4.1.1.4.1
    friendlyName: SubduralHematomaDetection
‘
```

H.2 Manifest Registration

To register a Manifest, the Platform must parse the Manifest for resources specified in the Application Configuration section. The Scopes, Traits and Components specified must all be defined as part of the Platform for the Application Configuration to be valid. A Manifest may include all the necessary resource definitions for the Application Configurations or may define resources in advance of making a POST request of the Application Configuration. Applications are responsible for publishing the most recent version of its Application Configuration to the Platform.

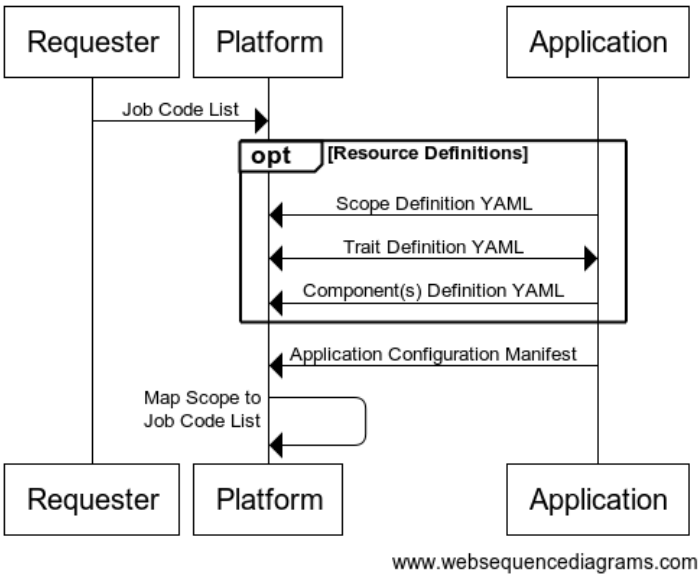
Applications may not change their Scopes without removing their previous registration and performing a new registration. Even if the Manifest was originally sent via POST as a single YAML file, the Application must allow for the Platform to process these separately, therefore PATCH commands are not allowed for Application Configurations, these must use DELETE and then a new POST. A Platform will only issue a successful DELETE if there are no resource dependencies. Should there be another resource dependent upon one where a DELETE is issued, the Platform must return an error which should include information about the dependent resources.

H.2.1 Scaling Considerations

An Application when registering itself can specify in its Traits the number of concurrent instances a Platform can instantiate should the Application be containerized and allow scaling (see J.1 Scaling as an example). The Application may also have a feature where it registers an updated or additional Application Configuration Manifests to allow for additional job requests. In this case the Application is responsible to keep track of active Manifests and remove those no longer valid. Applications defined as services or invoked through API must perform scaling as of function of themselves and respond properly to liveliness and heath requests as appropriate.

H.3 Registration Workflow

Registration Workflow Figure H.3.1




```
title Registration Workflow Figure H.3.1
participant Requester
participant Platform
participant Application
Requester -> Platform : Job Code List
opt Resource Definitions
Application -> Platform : Scope Definition YAML
Application <-> Platform : Trait Definition YAML
Application -> Platform : Component(s) Definition YAML
end
Application -> Platform : Application Configuration Manifest
Platform -> Platform : Map Scope to\nJob Code List
```

For example, a hospital purchases an Application Platform, as well as several add on Applications from various vendors. The Platform first must get a list of job codes from the hospital for which requests will be coded for Applications to perform work. The Platform will then need to map these codes to the Application Manifests Scopes it has registered so that when a job request is received it can pass that request to an appropriate Application. Platforms can have one or more Applications registered for a particular job code. In this case the Platform will need to manage the job assignments.

I. Discovery

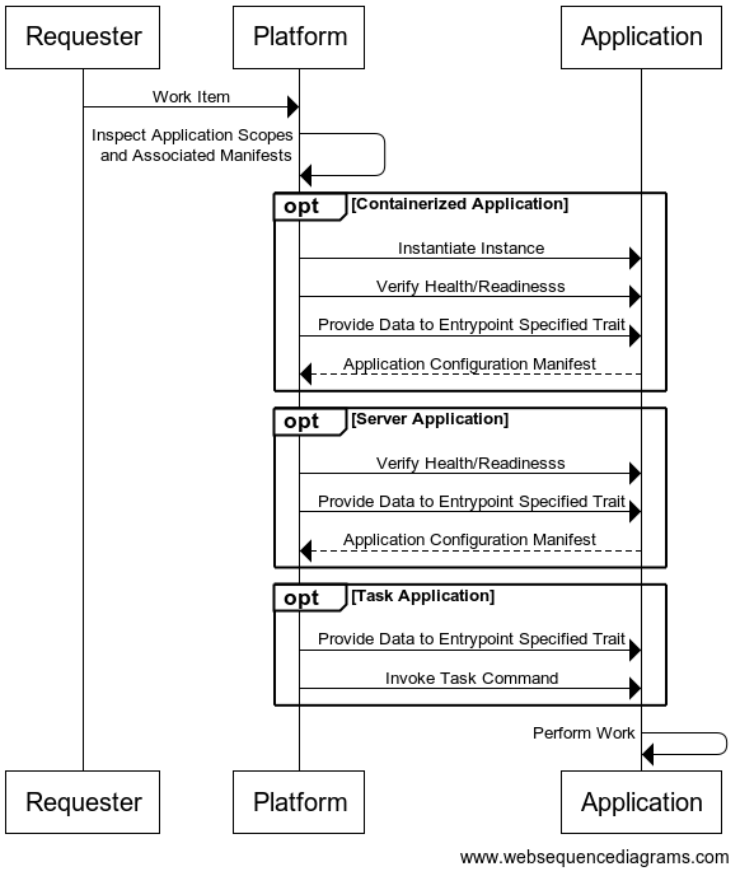
I.1 Platform Discovery

Platform Discovery is the process by which the Platform requests and inspects the Application Manifest prior to invoking the Application to assigning the Application jobs. When a Platform receives a request for work to be done, it will inspect the list of registered applications which it hosts. Once the appropriate Application has been determined, the Platform will either check the heath and/or readiness of the Application or in the case of containerized Application, instantiate the instance. Once it is verified that the Application is available, the Platform shall invoke the Application as described in the Manifest and assign the job to the Application. Application Manifests must be inspected with each job to ensure the proper Traits are used. Applications may wish to use annotations in the Application Configuration to provide

version information to a Platform for Manifest exchange. This will allow Platforms to limit the amount of parsing required between jobs and verify a Manifest has not changed between registration and requests.

I.1.1 Platform Discovery Workflow

Discovery Workflow Figure I.1.1




```
title Discovery Workflow Figure I.1.1
participant Requester
participant Platform
participant Application
Requester -> Platform : Work Item
Platform -> Platform : Inspect Application Scopes \nand Associated Manifests
opt Containerized Application
Platform -> Application : Instantiate Instance
Platform -> Application : Verify Health/Readinesss
Platform -> Application : Provide Data to Entrypoint Specified Trait
Application --> Platform : Application Configuration Manifest
end
opt Server Application
Platform -> Application : Verify Health/Readinesss
Platform -> Application : Provide Data to Entrypoint Specified Trait
Application --> Platform : Application Configuration Manifest
end
opt Task Application
Platform -> Application : Provide Data to Entrypoint Specified Trait
Platform -> Application : Invoke Task Command
end
Application -> Application : Perform Work
```

I.2 Application Discovery

Application Discovery is the process by which an Application queries a Platform for registered Scopes, Traits, and Components which it can use to build its own Application Configuration Manifest. Applications may wish to discover other Applications hosted by the Platform which could be used for nesting functionality whereby one hosted Application may become a requestor of another Application. For example, Application A may ask Application B to perform segmentation or other data transformation prior to executing its own code. In this case Application A must request work of Application B via the Platform and not directly and becomes a Requester. Table H.1 provides the acceptable resource paths for providing this as a web service.

The format for making a web service GET request is as follows:

GET: {platform}/{resource path}/{resource name} – unique resource definition

GET: {platform}/{resource path}/ – list of all defined resource definitions

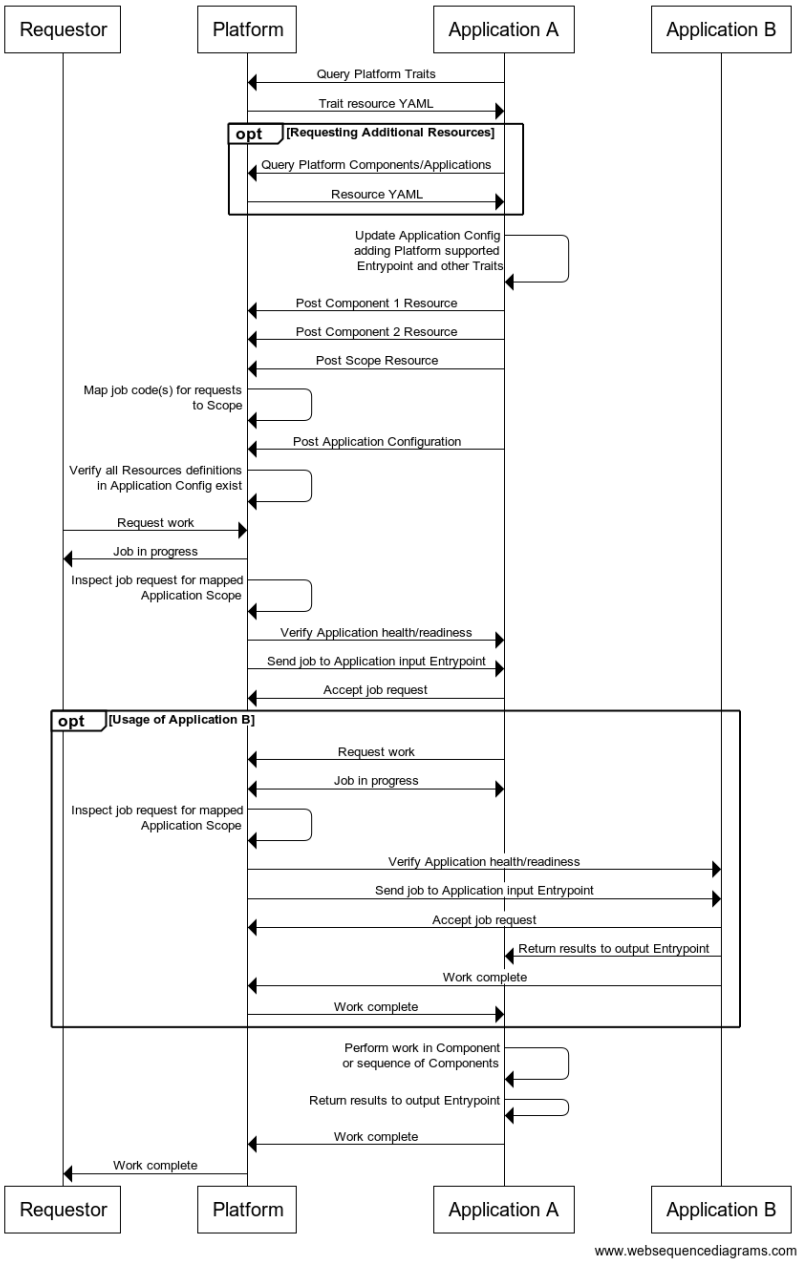
Note that when retrieving a resource by name, the entire resource definition must be returned by the Platform, but when requesting the list from the root resource path, the Platform has the option to return just the resource names, requiring the requestor to make a second request to get the entire definition for a specific resource.

Example request for a unique resource instance:

curl --location --request GET 'https://myplatform.mysite.org/scopes/example-operation-scope'

I.2.1 Application Discovery Workflow

Application Discovery Workflow Figure I.2.1



title Application Discovery Workflow Figure I.2.1
participant Requestor
participant Platform
participant Application A
participant Application B
Application A -> Platform : Query Platform Traits
Platform -> Application A : Trait resource YAML
opt Requesting Additional Resources
Application A -> Platform : Query Platform Components/Applications
Platform -> Application A : Resource YAML
end
Application A -> Application A : Update Application Config\nadding Platform supported\n Entrypoint and other Traits
Application A -> Platform : Post Component 1 Resource
Application A -> Platform : Post Component 2 Resource

Application A -> Platform : Post Scope Resource
Platform -> Platform : Map job code(s) for requests\nto Scope
Application A -> Platform : Post Application Configuration
Platform -> Platform : Verify all Resources definitions\nin Application Config exist
Requestor -> Platform : Request work
Platform -> Requestor : Job in progress
Platform -> Platform : Inspect job request for mapped\nApplication Scope
Platform -> Application A : Verify Application health/readiness
Platform -> Application A : Send job to Application input Entrypoint
Application A -> Platform : Accept job request
opt Usage of Application B
Application A -> Platform : Request work
Platform <-> Application A : Job in progress
Platform -> Platform : Inspect job request for mapped\nApplication Scope
Platform -> Application B : Verify Application health/readiness
Platform -> Application B : Send job to Application input Entrypoint
Application B -> Platform : Accept job request
Application B -> Application A : Return results to output Entrypoint
Application B -> Platform : Work complete
Platform -> Application A : Work complete
end
Application A -> Application A : Perform work in Component\nor sequence of Components
Application A -> Application A : Return results to output Entrypoint
Application A -> Platform : Work complete
Platform -> Requestor : Work complete

J. Control

Control is the process by which the Platform is in control of instantiation or execution of an Application. The Platform may additionally control scaling of one or more components within the Application. Control can also represent features a Platform provides such as proxy and data transformation. When providing proxy services, the Platform can specify itself as the host providing Entrypoint Traits.

J.1 Scaling

Manual Scaler is a core trait definition in OAM which can be leveraged. As a core trait, the Manual Scaler Trait must be part of OAM implementation, therefore a trait definition is not needed for it. The following snippet from an application configuration shows how the manual scaler trait is applied and configured for a component. Table J.1 provides the attributes for the DICOM Job Timeout Trait. The Definition Reference for Job Timeout is also provided as reference.

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: ManualScaler
spec:
  definitionRef:
    name: schema.manualscaler.oam.dev
```

Table J.1 Manual Scaler Trait Trait Attributes

Attribute	Type	Required	Default Value	Description
replicaCount	integer	Y		Specifies the number of replicas for the component.

Example Manual Scaler Trait Usage

```
apiVersion: core.oam.dev/v1alpha3
kind: ApplicationConfiguration
```

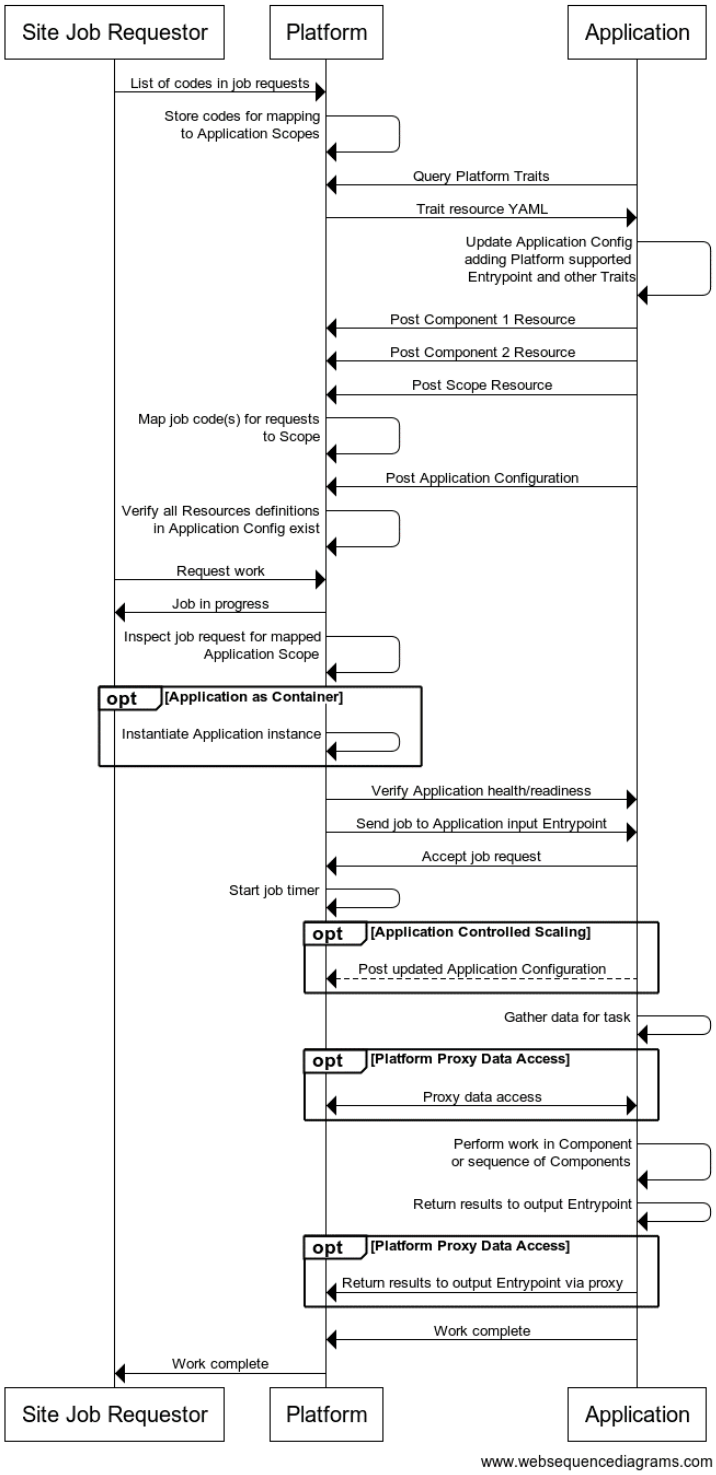
```
metadata:
  name: custom-single-app
  annotations:
    version: v1.0.0
    description: "Customized version of single-app"
spec:
  components:
    - componentName: frontend
      traits:
        - trait:
            apiVersion: core.oam.dev/v1alpha3
            kind: ManualScaler
            spec:
              replicaCount: 5
```

J.2 Proxy

A Platform may provide services outside the scope of the institutional systems capabilities. For example, a Platform may provide DICOMweb Entrypoints for institutional systems that only have DIMSE services. This is to enable greater Platform flexibility for the Applications it hosts. These Entrypoint setting will be configured by Applications the same way they would be if they were directly provided by the institution. An Application will be indifferent to what systems are providing services but will function independently as per the direction of their configuration and the job request.

J.3 Application Manifest Driven Workflow

Application Manifest Driven Workflow Figure J.3




```
title Application Manifest Driven Workflow Figure J.3
participant Site Job Requestor
participant Platform
participant Application
Site Job Requestor -> Platform : List of codes in job requests
```

```
Platform -> Platform : Store codes for mapping\nto Application Scopes
Application -> Platform : Query Platform Traits
Platform -> Application : Trait resource YAML
Application -> Application : Update Application Config\nadding Platform supported\n Entrypoint and other Traits
Application -> Platform : Post Component 1 Resource
Application -> Platform : Post Component 2 Resource
Application -> Platform : Post Scope Resource
Platform -> Platform : Map job code(s) for requests\nto Scope
Application -> Platform : Post Application Configuration
Platform -> Platform : Verify all Resources definitions\nin Application Config exist
Site Job Requestor -> Platform : Request work
Platform -> Site Job Requestor : Job in progress
Platform -> Platform : Inspect job request for mapped\nApplication Scope
opt Application as Container
Platform -> Platform : Instantiate Application instance
end
Platform -> Application : Verify Application health/readiness
Platform -> Application : Send job to Application input Entrypoint
Application -> Platform : Accept job request
Platform -> Platform : Start job timer
opt Application Controlled Scaling
Application --> Platform : Post updated Application Configuration
end
Application -> Application : Gather data for task
opt Platform Proxy Data Access
Platform <-> Application : Proxy data access
end
Application -> Application : Perform work in Component\nor sequence of Components
Application -> Application : Return results to output Entrypoint
opt Platform Proxy Data Access
Application -> Platform : Return results to output Entrypoint via proxy
end
Application -> Platform : Work complete
Platform -> Site Job Requestor : Work complete
```

Annex A - DICOM Standardized Resource Definition Schemas

1. Scopes

1.1 DICOM Operation Scope

1.1.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: ScopeDefintion
metadata:
  name: DicomOperationScope
spec:
  definitionRef:
    name: schema.dicomoperationscope.oam.dev
```

1.1.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: ScopeDefinition
metadata:
    name: DicomOperationScope
spec:
    type:
        description: Must be workitem, route, or invoked.  Details the expected responsibility of the Platform as it relates to the Application.
        enum:
        - workitem
        - route
        - invoked
        type: string
```

```
    vendor:
      description: The vendor of the Application
      type: string
    version:
      description: The version of the Application
      type: string
    code:
      description: A reference to a code defined by a terminology system
      type: string
    codeSystem:
      description: The identification of the code system that defines the meaning of the symbol in the code
      type: string
    sopClasses:
      description: SOP Classes as defined by DICOM Part 4.  This is represented as an array of SOP Classes for which the Application capabilities are valid.  When an SOP Class is not present the platform may assign work to this application scope at its discretion
      type: array
    friendlyName:
      description: Name given to the Work Item to understand its basic purpose
      type: string
required:
- type
additionalProperties: false
```

2. Workloads

2.1 DICOM Server Workload

2.1.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: WorkloadDefintion
metadata:
  name: DicomServerWorkload
spec:
  definitionRef:
    name: schema.dicomserverworkload.oam.dev
```

2.1.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: WorkloadDefinition
metadata:
    name: DicomServerWorkload
spec:
    host:
      description: Host name or ip of the server.  Note in workloads where the host name or ip may also be specified elsewhere in the settings and either is to be made mutable the parameter fieldPath for both locations needs to be specified.
      type: string
    livenessProbe:
      description: Instructions for assessing whether the server is alive.
      type: object
      properties:
        exec:
          description: Instructions for assessing container health by executing a command. Either this attribute or the httpGet attribute or the tcpSocket attribute MUST be specified. This attribute is mutually exclusive with both the httpGet attribute and the tcpSocket attribute.
```

```
            type: object
            properties:
               command:
                  description: A command to be executed inside the container to assess its health. Each
space delimited token of the command is a separate array element. Commands exiting 0 are considered
to be successful probes, whilst all other exit codes are considered failures.
                  type: string
                  items:
                     type: string
         httpGet:
            description: Instructions for assessing container health by executing an HTTP GET request.
Either this attribute or the exec attribute or the tcpSocket attribute MUST be specified. This attribute is
mutually exclusive with both the exec attribute and the tcpSocket attribute.
            type: object
            properties:
               path:
                  description: The endpoint, relative to the port, to which the HTTP GET request should be
directed.
                  type: string
               port:
                  description: The TCP socket within the container to which the HTTP GET request should
be directed.
                  type: integrer
               httpHeaders:
                  description: Optional HTTP headers.
                  type: object
                  properties:
                     name:
                        description: An HTTP header name. This must be unique per HTTP GET-based
probe.
                        type: string
                     value:
                        description: An HTTP header value.
                        type: string
                  required:
                  - name
                  - value
            required:
            - path
            - port
         tcpSocket:
            description: Instructions for assessing container health by probing a TCP socket. Either this
attribute or the exec attribute or the httpGet attribute MUST be specified. This attribute is mutually
exclusive with both the exec attribute and the httpGet attribute.
            type: object
            properties:
               port:
                  description: The TCP socket within the container that should be probed to assess
container health.
                  type: integer
         initialDelaySeconds:
            description: Number of seconds after the container is started before the first probe is initiated.
            type: integer
            default: 0
         periodSeconds:
            description: How often, in seconds, to execute the probe.
```

```
            type: integer
            default: 10
         timeoutSeconds:
            description: Number of seconds after which the probe times out.
            type: integer
            default: 1
         successThreshold:
            description: Minimum consecutive successes for the probe to be considered successful after
having failed.
            type: integer
            default: 1
         failureThreshold:
            description: Number of consecutive failures required to determine the container is not alive
(liveness probe) or not ready (readiness probe).
            type: integer
            default: 3
   readinessProbe:
      description: Instructions for assessing whether the server is alive.
      type: object
      properties:
         exec:
            description: Instructions for assessing container health by executing a command. Either this
attribute or the httpGet attribute or the tcpSocket attribute MUST be specified. This attribute is mutually
exclusive with both the httpGet attribute and the tcpSocket attribute.
            type: object
            properties:
               command:
                  description: A command to be executed inside the container to assess its health. Each
space delimited token of the command is a separate array element. Commands exiting 0 are considered
to be successful probes, whilst all other exit codes are considered failures.
                  type: string
                  items:
                     type: string
         httpGet:
            description: Instructions for assessing container health by executing an HTTP GET request.
Either this attribute or the exec attribute or the tcpSocket attribute MUST be specified. This attribute is
mutually exclusive with both the exec attribute and the tcpSocket attribute.
            type: object
            properties:
               path:
                  description: The endpoint, relative to the port, to which the HTTP GET request should be
directed.
                  type: string
               port:
                  description: The TCP socket within the container to which the HTTP GET request should
be directed.
                  type: integrer
               httpHeaders:
                  description: Optional HTTP headers.
                  type: object
                  properties:
                     name:
                        description: An HTTP header name. This must be unique per HTTP GET-based
probe.
                        type: string
                     value:
```

```
                    description: An HTTP header value.
                    type: string
                required:
                - name
                - value
            required:
            - path
            - port
        tcpSocket:
            description: Instructions for assessing container health by probing a TCP socket. Either this attribute or the exec attribute or the httpGet attribute MUST be specified. This attribute is mutually exclusive with both the exec attribute and the httpGet attribute.
            type: object
            properties:
               port:
                  description: The TCP socket within the container that should be probed to assess container health.
                  type: integer
        initialDelaySeconds:
            description: Number of seconds after the container is started before the first probe is initiated.
            type: integer
            default: 0
        periodSeconds:
            description: How often, in seconds, to execute the probe.
            type: integer
            default: 10
        timeoutSeconds:
            description: Number of seconds after which the probe times out.
            type: integer
            default: 1
        successThreshold:
            description: Minimum consecutive successes for the probe to be considered successful after having failed.
            type: integer
            default: 1
        failureThreshold:
            description: Number of consecutive failures required to determine the container is not alive (liveness probe) or not ready (readiness probe).
            type: integer
            default: 3
required:
- host
additionalProperties: false
```

2.2 DICOM Task Workload

2.2.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: WorkloadDefintion
metadata:
  name: DicomTaskWorkload
spec:
  definitionRef:
    name: schema.dicomtaskworkload.oam.dev
```

2.2.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: WorkloadDefinition
metadata:
   name: DicomTaskWorkload
spec:
   exec:
      description: The path or uri to the executable.
      type: object
      properties:
         command:
            description: A command to be executed.  Each command will be executed sequentially.
Commands exiting 0 are considered successful
            type: string
            items:
               type: string
   env:
      description: Environment variables.  For Task Workload types environmental variables such as
operating system or runtime component requirements should be specified here.
      type: object
      properties:
         name:
            description: The environment variable name. Must be unique per container.
            type: integer
         value:
            description: The environment variable value.
            type: integer
      required:
      - name
required:
- exec
additionalProperties: false
```

3. Traits

3.1 DICOM Job Timeout Trait

3.1.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: JobTimeout
spec:
  definitionRef:
    name: schema.jobtimeout.oam.dev
```

3.1.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
   name: JobTimeout
spec:
   seconds:
      description: Used to set a timeout for a job.
      type: integer
      default: 30
```

```
additionalProperties: false
```

3.2 DICOM Audit Trail Trait

3.2.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AuditTrail
spec:
  definitionRef:
    name: schema.audittrail.oam.dev
```

3.2.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AuditTrail
spec:
    tlsVersion:
        description: Only versions supported in Part 15 are acceptable
        type: string
    tlsCertificate:
        description: Trusted certificate for this communication
        type: string
    tlsPassword:
        description: Password for keystore to access certificate if required
        type: string
    tlsCipherSuite:
        description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        type: string
    syslogUri:
        description: Identifies the resource by name at the specified location or URL
        type: string
required:
- syslogUri
additionalProperties: false
```

3.3 DICOM Time Sync Trait

3.3.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: TimeSync
spec:
  definitionRef:
    name: schema.timesync.oam.dev
```

3.3.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: TimeSync
spec:
```

```
    ntpTimeServer:
      description: Server address used for time synchronization.
      type: string
additionalProperties: false
```

3.4 DICOM Application C-Store Provider Trait

3.4.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppCStoreProvider
spec:
  definitionRef:
    name: schema.appcstoreprovider.oam.dev
```

3.4.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AppCStoreProvider
spec:
    aet:
      description: AET the entrypoint uses for inbound data transfers, the entrypoint's storage SCP AET
      type: string
    port:
      description: Port the entrypoint uses for inbound data transfers. Note in workloads where the port
may also be specified elsewhere in the settings and is to be made mutable the parameter fieldPath for
both locations needs to be specified
      type: integer
    tlsVersion:
      description: Only versions supported in Part 15 are acceptable
      type: string
    tlsCertificate:
      description: Trusted certificate for this communication
      type: string
    tlsPassword:
      description: Password for keystore to access certificate if required
      type: string
    tlsCipherSuite:
      description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      type: string
    scuAet:
      description: AET of SCU if required by SCP
      type: string
    scuHost:
      description: Host name or ip of SCU if required by SCP
      type: string
required:
- aet
- port
additionalProperties: false
```

3.5 DICOM Application C-Store User Trait

3.5.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppCStoreUser
spec:
  definitionRef:
    name: schema.appcstoreuser.oam.dev
```

3.5.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AppCStoreUser
spec:
    destAet:
        description: AET the entrypoint uses for inbound data transfers, the entrypoint's storage SCP AET
        type: string
    destPort:
        description: Port the entrypoint uses for inbound data transfers. Note in workloads where the port
may also be specified elsewhere in the settings and is to be made mutable the parameter fieldPath for
both locations needs to be specified
        type: integer
    destHost:
        description: Host name or ip the entrypoint uses for inbound data transfers.  Note in workloads
where the host name or ip may also be specified elsewhere in the settings and either is to be made
mutable the parameter fieldPath for both locations needs to be specified
        type: string
    tlsVersion:
        description: Only versions supported in Part 15 are acceptable
        type: string
    tlsCertificate:
        description: Trusted certificate for this communication
        type: string
    tlsPassword:
        description: Password for keystore to access certificate if required
        type: string
    tlsCipherSuite:
        description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        type: string
required:
- destAet
- destPort
- destHost
additionalProperties: false
```

3.6 DICOM Application WADO User Trait

3.6.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppWadoUser
```

```
spec:
  definitionRef:
    name: schema.appwadouser.oam.dev
```

3.6.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AppWadoUser
spec:
    resourceUri:
        description: Identifies a resource via a representation of its primary access mechanism.
        type: string
    acceptHeaders:
        description: Section 9.1.2.2.1.  The value of this parameter, if present, shall be either
application/dicom, or one or more of the Rendered Media Types.
        type: string
    annotation:
        description: Section 8.3.5.1.2 May be patient and/or technique.  Patient indicates that the rendered
images shall be annotated with patient information.  Technique indicates that the rendered images shall
be annotated with information about the procedure that was performed.
        type: array
        items:
            type: string
            enum:
            - patient
            - technique
    quality:
        description: Section 8.3.5.1.3.  Is an unsigned integer between 1 and 100 inclusive, with 100 being
the best quality.
        type: integer
    viewport:
        description: Section 8.3.5.1.3  vw and vh are positive integers specifying the width and height, in
pixels, of the rendered image or video. Both values are required.  sx and sy are decimal numbers whose
absolute values specify, in pixels, the top-left corner of the region of the source image(s) to be rendered.
If either sx or sy is not specified, it defaults to 0. A value of 0,0 specifies the top-left corner of the source
image(s).  sw and sh are decimal numbers whose absolute values specify, in pixels, the width and height
of the region of the source image(s) to be rendered. If sw is not specified, it defaults to the right edge of
the source image. If sh is not specified, it defaults to the bottom edge of the source image. If sw is a
negative value, the image is flipped horizontally. If sh is a negative value, the image is flipped vertically.
        type: string
    window:
        description: Section 8.3.5.1.4 Center, width, function – center is a decimal number containing the
window-center value.  Width is a decimal number containing the window-width value and function is one
of the following ‘linear’, ‘linear-exact’ or ‘sigmoid’
        type: string
    iccProfile:
        description: Section 8.3.5.1.5 Must be ‘no’, ‘yes’, ‘srgb’, ‘adobergb’, or ‘rommrgb’.
        type: string
        enmun:
        - no
        - yes
        - srgb
        - adobergb
        - rommrgb
    tlsVersion:
```

```
      description: Only versions supported in Part 15 are acceptable
      type: string
   tlsCertificate:
      description: Trusted certificate for this communication
      type: string
   tlsPassword:
      description: Password for keystore to access certificate if required
      type: string
   tlsCipherSuite:
      description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      type: string
required:
- resourceUri
- acceptHeaders
additionalProperties: false
```

3.7 DICOM Application STOW Provider Trait

3.7.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppStowProvider
spec:
  definitionRef:
    name: schema.appstowprovider.oam.dev
```

3.7.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
   name: AppStowProvider
spec:
   resourceUri:
      description: Identifies a resource via a representation of its primary access mechanism.
      type: string
   contentTypeHeaders:
      description: Section 8.7.3.5 DICOM Media Type Syntax
      type: string
   tlsVersion:
      description: Only versions supported in Part 15 are acceptable
      type: string
   tlsCertificate:
      description: Trusted certificate for this communication
      type: string
   tlsPassword:
      description: Password for keystore to access certificate if required
      type: string
   tlsCipherSuite:
      description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      type: string
required:
- resourceUri
- contentTypeHeaders
```

additionalProperties: false

3.8 DICOM Application STOW User Trait

3.8.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: AppStowUser
spec:
  definitionRef:
    name: schema.appstowuser.oam.dev
```

3.8.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: AppStowUser
spec:
    destResourceUri:
        description: Identifies a resource via a representation of its primary access mechanism.
        type: string
    destContentTypeHeaders:
        description: Section 8.7.3.5 DICOM Media Type Syntax
        type: string
    tlsVersion:
        description: Only versions supported in Part 15 are acceptable
        type: string
    tlsCertificate:
        description: Trusted certificate for this communication
        type: string
    tlsPassword:
        description: Password for keystore to access certificate if required
        type: string
    tlsCipherSuite:
        description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        type: string
required:
- destResourceUri
- destContentTypeHeaders
additionalProperties: false
```

3.9 DICOM Operator Input Trait

3.9.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: OperatorInput
spec:
  definitionRef:
    name: schema.operatorinput.oam.dev
```

3.9.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
```

```
kind: TraitDefinition
metadata:
  name: OperatorInput
spec:
  path:
    description: To the folder level, everything in the folder will be ingested
    type: string
  userType:
    description: Must be basic, userIdPasscode, kerberos, or saml
    type: string
    enum:
    - basic
    - userIdPasscode
    - kerberos
    - saml
  username:
    description: Identification used to access resource if required
    type: string
  passcode:
    description: Passcode used to access resources if required
    type: string
  dataTypes:
    description: When not present MIME types are considered to be DICOM and are configured as part
of the scope via SOP classes.
    type: array
      items: string
  signatureType:
    description: Must be baseRsa, creatorRsa, authorizationRsa, or structuredReportRsa
    type: string
    enum:
    - baseRsa
    - creatorRsa
    - authorizationRsa
    - structuredReportRsa
  macAlgorithm:
    description: Used for key-confirmation if required
    type: string
  publicKey:
    description: Used to decrypt data if required
    type: string
  efsAlgorithm:
    description: The symmetric encryption algorithm used
    type: string
  efsKey:
    description: Used to decrypt data if required
    type: string
required:
- path
additionalProperties: false
```

3.10 DICOM Operator Output Trait

3.10.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
```

```
  name: OperatorOutput
spec:
  definitionRef:
    name: schema.operatoroutput.oam.dev
```

3.10.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: OperatorOutput
spec:
    destPath:
        description: This is to the folder level
        type: string
    dataTypes:
        description: This is to ensure output is consistent, what the application must output.  DICOM objects
to be specified as dicom.
        type: array
            items: string
    userType:
        description: Must be basic, userIdPasscode, kerberos, or saml
        type: string
        enum:
        - basic
        - userIdPasscode
        - kerberos
        - saml
    username:
        description: Identification used to access resource if required
        type: string
    passcode:
        description: Passcode used to access resources if required
        type: string
    signatureType:
        description: Must be baseRsa, creatorRsa, authorizationRsa, or structuredReportRsa
        type: string
        enum:
        - baseRsa
        - creatorRsa
        - authorizationRsa
        - structuredReportRsa
    macAlgorithm:
        description: Used for key-confirmation if required
        type: string
    publicKey:
        description: Used to decrypt data if required
        type: string
    efsAlgorithm:
        description: The symmetric encryption algorithm used
        type: string
    efsKey:
        description: Used to decrypt data if required
        type: string
required:
- destPath
- dataTypes
```

additionalProperties: false

3.11 DICOM REST API Provider Trait

3.11.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: RestApiProvider
spec:
  definitionRef:
    name: schema.restapiprovider.oam.dev
```

3.11.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: RestApiProvider
spec:
    restApiName:
        description: Examples are AcrModelApi, DicomWSDL
        type: string
    restApiVersion:
        description: A string that identifies the version of the API
        type: string
    resourceUri:
        description: Identifies a resource via a representation of its primary access mechanism.
        type: string
    uriType:
        description: Examples are request, liveliness, readiness, log
        type: string
    tlsVersion:
        description: Only versions supported in Part 15 are acceptable
        type: string
    tlsCertificate:
        description: Trusted certificate for this communication
        type: string
    tlsPassword:
        description: Password for keystore to access certificate if required
        type: string
    tlsCipherSuite:
        description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        type: string
    authMethod:
        description: basicAuth, formAuth, clientCertAuth, oAuth, bearerAuth
        type: string
        enum:
        - basicAuth
        - formAuth
        - clientCertAuth
        - oAuth
        - bearerAuth
    apiKey:
        description: Key used to connect to the API
        type: string
```

```
    accessToken:
      description: The authorization of a specific application
      type: string
    refreshToken:
      description: Used to acquire new access token
      type: string
required:
- restApiName
- restApiVersion
- resourceUri
- uriType
additionalProperties: false
```

3.12 DICOM REST API User Trait

3.12.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: RestApiUser
spec:
  definitionRef:
    name: schema.restapiuser.oam.dev
```

3.12.4 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: RestApiUser
spec:
    restApiName:
      description: Examples are AcrModelApi, DicomWSDL
      type: string
    restApiVersion:
      description: A string that identifies the version of the API
      type: string
    resourceUri:
      description: Identifies a resource via a representation of its primary access mechanism.
      type: string
    uriType:
      description: Examples are request, liveliness, readiness, log
      type: string
    tlsVersion:
      description: Only versions supported in Part 15 are acceptable
      type: string
    tlsCertificate:
      description: Trusted certificate for this communication
      type: string
    tlsPassword:
      description: Password for keystore to access certificate if required
      type: string
    tlsCipherSuite:
      description: Represented as documented in Part 15 Sections B.9 and B.10, for example
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
      type: string
    authMethod:
```

```
      description: basicAuth, formAuth, clientCertAuth, oAuth, bearerAuth
      type: string
      enum:
      - basicAuth
      - formAuth
      - clientCertAuth
      - oAuth
      - bearerAuth
    apiKey:
      description: Key used to connect to the API
      type: string
    accessToken:
      description: The authorization of a specific application
      type: string
    refreshToken:
      description: Used to acquire new access token
      type: string
required:
- restApiName
- restApiVersion
- resourceUri
- uriType
additionalProperties: false
```

3.13 DICOM User Identity Security Trait

3.13.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: UserIdentitySecurity
spec:
  definitionRef:
    name: schema.useridentitysecurity.oam.dev
```

3.13.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: UserIdentitySecurity
spec:
    userType:
      description: Must be basic, userIdPasscode, kerberos, or saml
      type: string
      enum:
      - basic
      - userIdPasscode
      - kerberos
      - saml
    username:
      description: Identification used to access resource if required
      type: string
    passcode:
      description: Passcode used to access resources if required
      type: string
required:
```

```
- userType
additionalProperties: false
```

3.14 DICOM License Trait

3.14.1 Reference

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefintion
metadata:
  name: License
spec:
  definitionRef:
    name: schema.license.oam.dev
```

3.14.2 Definition

```
apiVersion: standard.oam.dev/v1alpha3
kind: TraitDefinition
metadata:
    name: License
spec:
    licenseKey:
        description: Application defined license key string
        type: string
    machineKey:
        description: Machine specific code generated, example MAC or some other machine code
        type: string
required:
- licenseKey
additionalProperties: false
```