

Configuration Description, Deployment, and Lifecycle Management

CDDL M Deployment API Draft 2005-05-20

Status of this Memo

This document provides information to the community regarding the specification of the Configuration Description, Deployment, and Lifecycle Management (CDDL M) Language. Distribution of this document is unlimited. This is a DRAFT document and continues to be revised.

Abstract

Successful realization of the Grid vision of a broadly applicable and adopted framework for distributed system integration, virtualization, and management requires the support for configuring Grid services, their deployment, and managing their lifecycle. A major part of this framework is a language in which to describe the components and systems that are required. This document, produced by the CDDL M working group within the Global Grid Forum (GGF), provides a definition of the service API whereby a Grid Resource is configured, instantiated, and destroyed.

GLOBAL GRID FORUM

office@ggf.org

www.ggf.org



Full Copyright Notice

Copyright © Global Grid Forum (2004-2005). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the GGF or other organizations, except as needed for the purpose of developing Grid Recommendations in which case the procedures for copyrights defined in the GGF Document process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the GGF or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE GLOBAL GRID FORUM DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property Statement

The GGF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the GGF Secretariat.

The GGF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this recommendation. Please address the information to the GGF Executive Director (see contact information at GGF website).

1 Table of Contents

1	Table of Contents.....	3
2	Introduction.....	4
2.1	CDDLWM-WG and the Purpose of this Document.....	4
2.2	XML Namespaces used in this document	5
3	Purpose of the Deployment API.....	5
3.1	Use Cases.....	6
3.2	Fault Tolerance.....	6
3.3	File upload.....	6
4	Architecture.....	6
4.1	Core Architecture.....	6
4.2	System State	8
4.3	Fault Tolerance.....	9
4.4	Other Architectural Features.....	9
4.5	WS-DM Integration.....	12
5	Deployment API Overview.....	13
5.1	Architecture of the Deployment System	13
5.2	Portal Endpoint.....	14
5.3	System Endpoint	16
6	Portal.....	18
6.1	Portal Properties.....	18
6.2	Operations	20
7	System.....	20
7.1	System Properties.....	20
7.2	System Operations.....	21
8	Notification	25
8.1	Notification Policy	25
8.2	WS-Notification Support	25
8.3	Portal Notifications.....	26
8.4	System Notifications.....	26
8.5	Fault-Tolerant Notification.....	26
9	Fault Policy.....	26
9.1	Fault Categories	26
9.2	Fault Security.....	28
9.3	Internationalization.....	28
9.4	Faults	28
9.5	Fault Error Codes	29
10	Implementation Requirements	30
11	Security.....	30
12	Editor Information	30
13	References.....	31
13.1	Normative References	31
13.2	Non-Normative References.....	31
	Appendix A: Event Topics	32

2 Introduction

The CDDL M framework needs to provide a deployment API for programs submitting jobs into the system for deployment, terminating existing jobs, and probing the state of the system.

This document defines the WS-Resource Framework-based deployment API for performing such tasks. It is targeted at those who implement either end of the API.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2.1 CDDL M-WG and the Purpose of this Document

The CDDL M WG addresses how to: describe configuration of services; deploy them on the Grid; and manage their deployment lifecycle (instantiate, initiate, start, stop, restart, etc.). The intent of the WG is to gather researchers, developers, practitioners, and theoreticians in the areas of services and application configuration, deployment, and deployment life-cycle management and to explore the community need for a broader effort in this area. The target of the CDDL M WG is to come up with the specifications for CDDL M a) language, b) component model, and c) basic services.

This document defines the WS-Resource Framework-based deployment API for performing such tasks. A CDDL M deployment infrastructure must implement this service in order for remote callers to create applications on the infrastructure.

This document is accompanied by an XML Schema (XSD) file and a WSDL service declaration. The latter two documents are to be viewed as the normative definitions of message elements and service operations. This document is the normative definition of the semantics of the operations themselves.

1.1.1 Configuration Description Language

The CDDL M Configuration Description Language (CDL) is an XML-based language for declarative description of system configuration that consists of components (deployment objects) defined in the CDDL M Component Model. The Deployment API uses a deployment descriptor in CDL in order to manage deployment lifecycle of systems. The language provides ways to describe properties (names, values, and types) of components including value references so that data can be assigned dynamically with preserving specified data dependencies. A system is described as a hierarchical structure of components. The language also provides prototype-based template functionality (i.e. prototype references) so that the user can describe a system by referring to component descriptions given by component providers.

The CDDL M Component Model outlines the requirements for creating a deployment object responsible for the lifecycle of a deployed resource. Each deployment object is defined using the CDL language and mapped to its implementation. The deployment object provides a WS-ResourceFramework (WSRF) compliant "Component Endpoint" for lifecycle operations on the managed resource. The model also defines the rules for managing the interaction of objects with the CDDL M Deployment API in order to provide an aggregate, controllable lifecycle and the operations which enable this process.







The Deployment API is a WSRF-based API for deploying applications to one or more target computers. Every set of computers to which systems can be deployed hosts one

or more “Portal Endpoints”, WSRF resources that provide a means to create new “System Endpoints”. Each System Endpoint represents a deployed system. The caller can upload files to it, then submit a CDL descriptor describing the system to deploy. The CDDLM implementation will then instantiate the component model components that constitute the system, according to the declarations in the CDL descriptor.

A System Endpoint is effectively a component in terms of the Component Model specification—it implements the properties and operations defined in that document. It also adds the ability to resolve references within the deployed system, enabling remote callers to examine the state of components with it.

2.2 XML Namespaces used in this document

Throughout the document, the following prefixes refer to the listed namespaces:

<i>prefix</i>	<i>URI</i>	<i>description</i>
xsd	http://www.w3.org/2000/10/XMLSchema	XML Schema Types
wsa	http://schemas.xmlsoap.org/ws/2003/03/addressing	WS-Addressing types
api	http://www.gridforum.org/cddlm/deployapi/2005/02	Deployment API
cdl	http://www.gridforum.org/namespaces/2005/02/cddlm/CDL-1.0	XML CDL
cmp	http://www.gridforum.org/cddlm/components/2005/02	Component Model
wsrf-bf	http://docs.oasis-open.org/wsrf/2004/06/  wsrf-WS-BaseFaults-1.2-draft-01.xsd	WS-BaseFaults
wsrf-rl	http://docs.oasis-open.org/wsrf/2004/06/  wsrf-WS-ResourceLifetime-1.2-draft-01.xsd	WS-Resource Framework
wsrf-rp	http://docs.oasis-open.org/wsrf/2004/06/  wsrf-WS-ResourceProperties-1.2-draft-01.xsd	WS Resource Properties
wsrf-nt	http://docs.oasis-open.org/wsn/2004/06/  wsn-WS-BaseNotification-1.2-draft-01.xsd	WS-BaseNotification
wsrf-top	http://docs.oasis-open.org/wsn/2004/06/  wsn-WS-Topics-1.2-draft-01.xsd	WS-Topics
s12	http://www.w3.org/2003/05/soap-envelope	SOAP1.2 Envelope
xml	http://www.w3.org/XML/1998/namespace	XML attributes
muws-pl-xs	http://docs.oasis-open.org/wsdm/2004/12/  muws/wsdm-muws-part1.xsd	Management using Web Services

Unprefixed types in the document and accompanying schema are in the namespace of the deployment API, that referenced to by the `api:` prefix

3 Purpose of the Deployment API

The deployment API is the SOAP/WS-ResourceFramework (WSRF) API for deploying applications to one or more target computers, physical or virtual.

The API is written assuming that the end user is deploying through a console program, a portal UI or some automated process. This program will be something written to facilitate deployment onto a grid fabric or other network infrastructure which is running the relevant CDDLM services.

3.1 Use Cases

There are three different core use cases of the deployment API:

- 1 The deployment target is an OGSA-compliant Grid Fabric. Resource allocation and Job submission (using the JSDL language [JSDL] or equivalent) is part of the deployment process. In this use case, the deployment API must integrate with the negotiation, and deploy a CDDLM-language described system over the machines allocated by the resource manager.
- 2 The deployment target is a pre-allocated cluster set of machines. The resource allocation process is bypassed—it can be presumed to have happened out of band. The user needs to upload data files to the cluster as part of the deployment.
- 3 One instance of a CDDLM runtime is delegating part of a deployment to another host. There is no guarantee that the two runtimes are the same implementation of CDDLM, or, if they are, that they are the same version.

3.2 Fault Tolerance

The architecture is intended to enable fault-tolerant implementations, to the extent that a failure of the deployment endpoint may not terminate the application, and may not render the application unreachable.

To achieve this goal, any set of nodes onto which a system is deployed, must be visible to and manageable by more than one deployment endpoint. Furthermore, if the failure of this endpoint is not to prevent access, any SOAP endpoints that provide direct access to the system, must be hosted on the system nodes themselves.

3.3 File upload

Part of remote deployment often consists of providing files to the remote systems, both code and data. The preferred solution to this is a remote asset store of some form, with an efficient transport and secure, version-based access to assets. This is not something that falls within the scope of this working group, and has not been addressed here.

As an interim solution, pending the availability of such systems, the deployment API provides a means to submit files to the remote system. These files are submitted in the request, and a URL of type (file:, http: or https:) is returned. The URL can be used within the deployment descriptor, and passed to the applications.

Uploaded files remain present for the lifespan of the deployed application. There is no sharing between deployed applications, no way to update a file, and no way to delete a file. Clearly, therefore, it is a pale substitute for a full asset store—and that is its deliberate intent. When deploying to an infrastructure that hosts a full asset store, that store and its remote upload API should be used instead of the file upload mechanism described in this document.

4 Architecture

4.1 Core Architecture

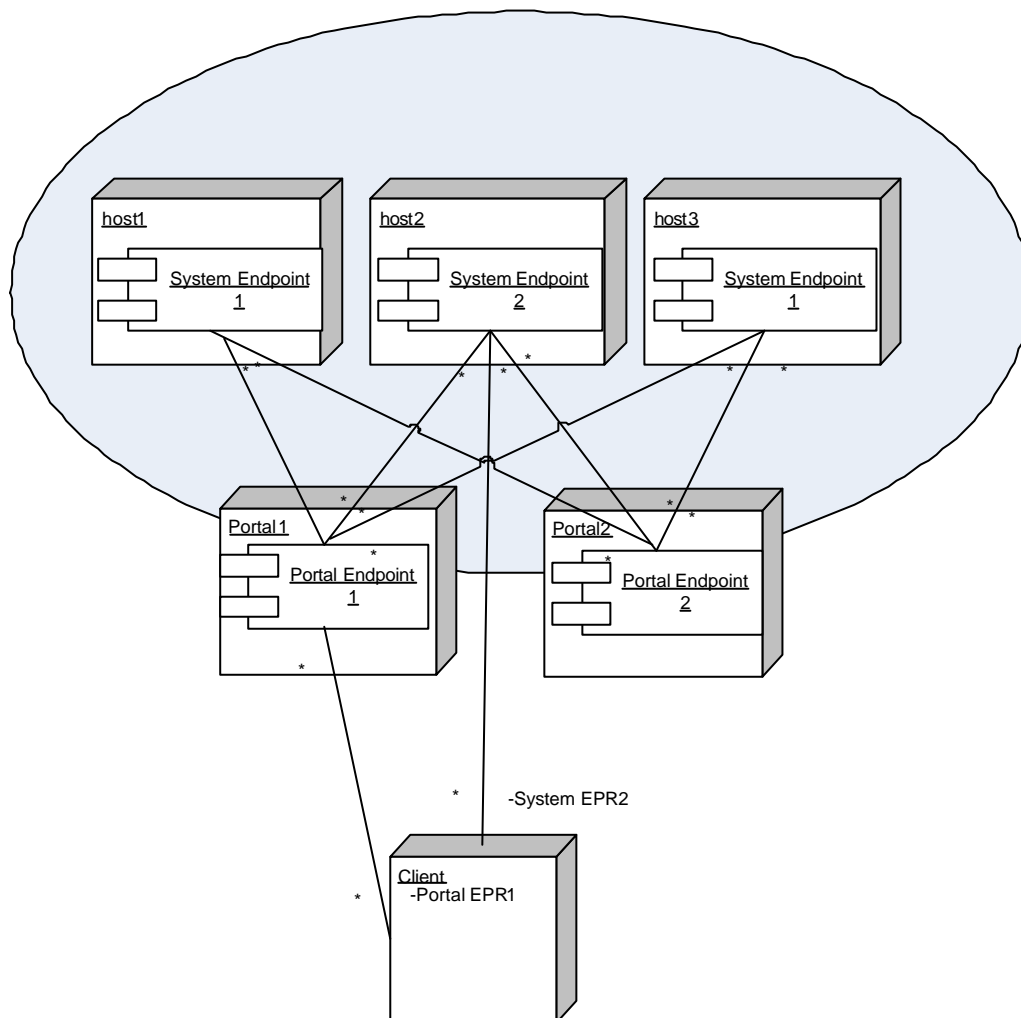
The API comprises a model for deployment, and a WS-ResourceFramework [WSRF] based means of interacting with this model.

A *deployment client* is an application that wishes to use the deployment API to deploy to one or more hosts that have been pre-allocated using a resource allocation system. A

deployment portal is a WSRF service endpoint that the deployment client communicates to, in order to deploy applications, an endpoint addressed via a WS-Addressing Endpoint Reference (EPR) [WS-A]. This specific EPR is referred to as the *Portal EPR*. The actual process for obtaining a Portal EPR is not in the scope of this document.

To deploy, the client first issues a request to the *Portal Endpoint* to create a system. This request includes a deployment descriptor in one of the CDDLM supported languages and potentially other information that describes and configures the application. This creation request returns a new EPR, which provides access to the state and operations of the system, the *System EPR*.

The *System EPR* can be bound to a *System Endpoint* hosted on any node that the Portal Endpoint chooses; there is no requirement that it is bound to the same node as the portal. For maximum availability, hosting the system endpoint on the same node that hosts the system may be the best approach. An example of this is shown in Figure 1 .



**Figure 1 . Model of how Portal and System endpoints may be distributed..
Multiple Portals can manage the same set of deployment nodes.**

The caller can then make a request to the *System Endpoint* to initialize the system. If successful, the application asynchronously enters the next state in its lifecycle,

initialized. Once a system has been initialized, it can be requested to enter through other stages of its lifecycle.

As a deployed system changes state, it sends lifecycle event notification messages to registered listeners, using a mechanism such as WS-BaseNotification [WS-BaseNotification]. The state of the system can also be determined by querying the appropriate resource property of the system, according to the WS-Resource Properties [WS-ResourceProperties] specification. There is also a synchronous, blocking call to probe the health of a system; this must be routed to the system itself, so that it can determine its own health. This will return its current state, and any custom status information the system chooses to return. If the system has failed, or terminated after a failure, the status information will include the fault information.

The Portal Endpoint supports other properties and operations. The list of currently deployed systems can be determined, along with their system EPRs. There are also static information and dynamic information documents which can be retrieved from the server; again these are represented as properties following the WS-Resource Properties specification.

The Portal Endpoint can raise events when new systems are created, using the WS-BaseNotification protocol.

4.2 System State

CDDL components have a uniform lifecycle, which is normatively described in the component model specification [Schaeffer05]. The state of a system mirrors that of the lifecycle of the components within. This is essential to permit aggregation of systems.

The first difference is the notion of a *destroyed* component. When a system is destroyed, all record of it is lost, along with any allocated storage. A terminated system, however, may still have state that is remotely accessible. In the deployment API, a terminated system remains visible until the endpoint is destroyed through the `<wsrf:Destroy/>` operation, or until the portal purges its set of terminated systems. Once a System Endpoint is destroyed, the system and any associated resources are no longer accessible.

The second difference is that state changing operations are asynchronous. A request to initialize, start or terminate a deployed system is received by the System Endpoint, validated, and, if valid, queued for execution. This makes communications somewhat resilient to communication faults.

The states of a system are as follows:

<i>instantiated</i>	The system has just been instantiated.
<i>initialized</i>	The system has been initialized.
<i>running</i>	The system is running
<i>failed</i>	The system has failed
<i>terminated</i>	The system has terminated

Instantiation and initialization represent the creation and configuration of a component, and when it is moved into *running* then it is actually functional. The state *failed* is entered automatically when a failure is detected; termination is the only exit

condition; *terminated* is the end state of a component and can be entered through a termination request.

The Portal's *create* operation will create and instantiate a system, a system which can then be directly manipulated via requests to its System Endpoint. The *run* operation will move the system to the running state, and *terminate* will move it to the terminated state.

4.3 Fault Tolerance

As stated, the architecture must enable fault-tolerant implementations. Here is how this is accomplished:

- Multiple Portal Endpoints can provide access to the same set of nodes.
- The failure of a portal does not imply the failure of a system.
- The failure of a node hosting a System Endpoint will result in the destruction of that system.
- Issuing a `<wsrf:Destroy>` request to a System Endpoint *will destroy the system*.
- Every system instance must have a WSRF property `muws-pl-xs:ResourceId` of type `xsd:anyURI` property that must be unique; this can be used for equality tests through simple string comparison.
- Portal Endpoints servicing a set of nodes should be discoverable by a client in some manner. Registration in a service group is one option [WS-ServiceGroup].
- Implementations may implement fault tolerant EPRs through the use of a dynamic DNS service, one in which the DNS entries for the hostname(s) of the portal are updated as portal instances appear and disappear. Client systems should be written with the knowledge that the IP addresses of an EPR may change, and not to cache resolved IP addresses indefinitely.

4.4 Other Architectural Features.

3.1.1 Named systems

Callers may provide a string name for a system. This system name, if provided, must be unique amongst all systems that a Portal Endpoint can manage.

The system name must begin with one of the characters in the set `[A..Za..z_.]` and continue with characters in the range `[A..Za..z09_.]`. This is a proper subset of the XSD type `NCName` element names, and is also a subset of the valid characters in a URL. This is intentional, and while the specification does not itself take advantage of the fact, languages may choose to do so.

3.1.2 Deployment Language Agnostic

The deployment API is agnostic as to which particular language, or version thereof, is used for a deployment descriptor. When a remote deployment is created, the language and version of the descriptor must be supplied. The sole requirement of a language is that it can either be nested inside an XML document, or that a URL to the descriptor is remotely accessible to the destination. In the case of the latter, the URL to the descriptor must be provide when initializing the system.

Every language is identified by a unique URI. This language URI must be supplied with the deployment descriptor or URI. A list of supported languages can be obtained from a Portal Endpoint.

3.1.3 Job Language Agnostic

Just as the API allows implementations to support deployment languages/versions, the API also permits multiple Job specification languages. For example, alongside JSDL, an implementation may support the Globus Resource Specification Language [GlobusRSL].

3.1.4 Deploy-time properties in the language and service API

Consider a deployment descriptor that wants to control onto which machine that it wants different components deployed. When the descriptor is written, the actual hosts are unknown. It is only during deployment that the mapping becomes apparent. Either the descriptor is rewritten with the fixed values, or we provide a way for subsidiary information to be passed alongside the descriptor.

The SmartFrog language [Goldsack04] supports this with the `PROPERTY` and `IPROPERTY` keywords, which bind keys in a Java `java.System.Properties` hashtable to string and integer values. For example, a deployment descriptor could be bound to three properties:

```
database extends Database {  
    sfHostname PROPERTY hosts.database;  
    password PROPERTY database.password;  
    localhost LAZY PROPERTY local.hostname  
}
```

At deployment time, each property string is looked up and assigned to the attribute, or a fault is raised. The `LAZY` keyword indicates that the evaluation must not take place in the context of the process interpreting the deployment descriptor, but instead the system actually hosting it. While the XML language does not explicitly contain such a feature [XML-CDL], a component could be designed to extract the values from the name/value list.

To enable this functionality within the Service interface, one of the deployment options declares a set of name/value pairs. How these tuples are exposed to a deployment language/framework is a language-specific feature.

3.1.5 Extensibility

The deployment API is designed to support extensible implementations, and future enhancements to the API over time.

3.1.5.1 Extra Operations

A service implementation may offer extra operations at any EPR.

- 1 Private extensions must not add new declarations to the XML namespaces used in this document: they must be in their own, private, namespace.
- 2 Implementations should document these operations and provide updated WSDL descriptions.
- 3 There is no requirement for the extra operations supported by an EPR to remain constant over any period of time. They may even change during the period in which an EPR remains valid.

1.1.1.1 Extra WSRF operations

This specification and the accompanying XSD/WSDL documents define the minimum set of WSRF operations that an endpoint must implement. There are other operations that the WSRF specification family and WSDM list as optional.

An implementation may choose to support these extra operations. If this is done, the messages and operations must match the relevant XSD and WSDL documents of the appropriate specification, and the semantics of the operations must match that of the specification itself.

1.1.1.2 Extra WS-Resource Properties

A service implementation may offer extra WS-Resource properties at any EPR. Again, they must be in their own, private, namespace. Implementations should document these properties and provide updated WSDL descriptions.

1.1.1.3 Extra deployment options

It is possible that extra deployment options may be offered on different implementations. The core of such customization should be in deployment descriptors themselves, yet there may be a need to provide extra deployment metadata.

This is implemented through an `<options>` element in the `<initialize>` message. This (optional) element contains a list of zero or more deployment options. These are extra parameters to the deployment request. Every option is named with a URI, and can have a string or integer attribute value, or contain nested XML. A `mustUnderstand` attribute is used to indicate whether or not an option must be understood.

The option list is a very powerful aspect of the API, but potentially dangerous. Any protocol standard which has optional aspects is harder to write clients against than one which does not, as there is likely to be less consistency between different implementations. To manage this risk, the deployment API has the following requirements on optional metadata parameters:

- All options must be that: optional. It must not be an error to deploy a system with no options declared.
- Every option is named by a URI.
- All URIs that begin with `http://gridforum.org/cddlm/` are reserved for options defined by the CDDLM working group.
- Options must contain either string, integer, Boolean or arbitrary XML values. String and integer values are supported via attributes; XML is supported as nested data.
- An option must contain only one value type. Implementations must raise a fault if multiple nested or attribute values are declared on the same option.
- All options that an implementation supports must be enumerated in the server information property of the Portal Endpoint.
- It is an error to include multiple options of the same URI in a descriptor. Implementations must raise a fault when this occurs.
- Options may be processed in any order. Options must not require a specific order of processing.

- Service implementations must ignore any options that they do not recognize, if `mustUnderstand="false"` for that option.
- Service implementations must understand all options which are supplied with `mustUnderstand="true"` for that option. If any such option is not understood, a fault must be raised.

The processing rules for deployment are as follows:

- 1 Option processing must take place before the system is moved to the running state.
- 2 An implementation must be able to deploy a system when the entire options portion of the request is empty or omitted.
- 3 Any option that is marked `mustUnderstand="true"` MUST be understood. If not, the Fault `"not-understood"` must be raised, identifying the particular option by its URI in the `extraData` field of the fault.
- 4 Implementations must not raise this fault when they do not understand any options that are marked `mustUnderstand="false"`, or for which there is no `mustUnderstand` attribute. These must be ignored.
- 5 Duplicate options must cause the operation to be rejected with a `bad-argument` fault, identifying the particular option by its URI in the `extraData` field of the fault.

4.5 WS-DM Integration

The deployment infrastructure is designed to integrate with a WS-DM management framework. Both Portal and System endpoints support the MUWS `ResourceId` and `ManageabilityCapability` attributes, to uniquely identify each endpoint, and to enumerate their management capabilities. All supported events are derived from the MUWS event type, and the state model of a System Endpoint is derived from the MUWS state types.

Implementations may add more management capabilities, as they see fit. For example, a Portal endpoint may export a MUWS property that describes the portal's operational state, and provides an event notifying listeners of changes in that state.

5 Deployment API Overview

The service API consists of two endpoint types, Portal Endpoints, addressed by Portal EPRs, and System Endpoints, addressed by System EPRs. Portal Endpoints return System EPRs to callers, either in response to lookup/mapping messages, or when a system is successfully created.

The two endpoint types are *Resources* within the terminology of the WS-Resource Framework specifications.

5.1 Architecture of the Deployment System

Everything is implemented as a resource in the WSRF framework, everything is manageable in the context of the MUWS infrastructure.

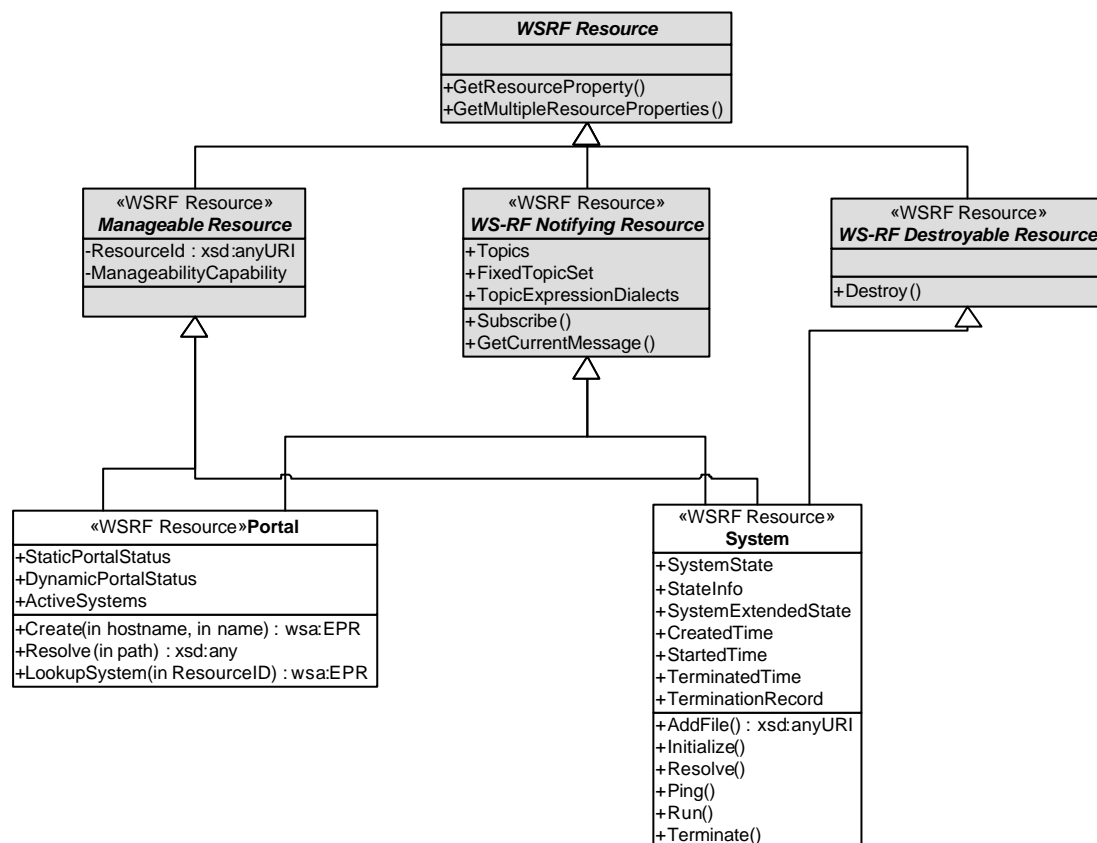


Figure 2 . Endpoint architecture. The items in grey are specified externally.

5.1.1 Sequence Diagram: a simple deployment

Figure 3 is a UML sequence diagram of an interaction with the deployment infrastructure. A client application connects a portal that it has knowledge of, and creates a system resource. It can then add files to the system, before moving the system into the functional state.

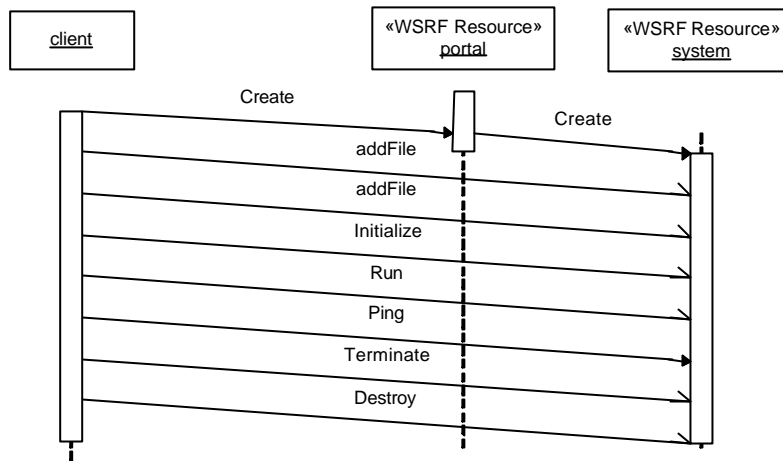


Figure 3 . Sequence diagram of a simple deployment

During the life of the system, it will respond to `Ping` and `Resolve` operations. Eventually it is terminated, and then finally the resource itself is destroyed.

5.1.2 Sequence Diagram: Subscribing to events from an existing system

Figure 4 shows a client connecting to a portal, and making a `LookupSystem` call to get the EPR of a system. One property of the portal, `ActiveSystems`, lists all active systems that the portal is aware of and to which the user may be allowed access.

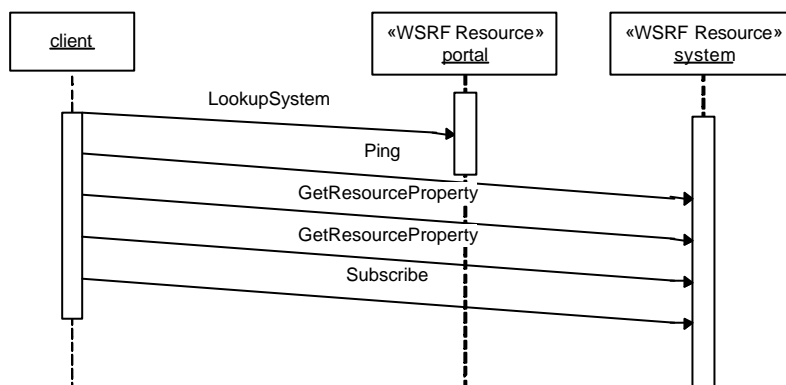


Figure 4 . Subscription sequence diagram

The client issues a `Ping` call to verify the health of the remote system, then two `GetResourceProperty` messages to access resource properties. Finally, it sends a `Subscribe` request to subscribe to the lifecycle event topic of the system.

Later, when lifecycle events take place, the system will send notifications to the EPR provided in the subscription message.

5.2 Portal Endpoint

The portal endpoint is the endpoint that the caller initially locates and communicates with. It can be used to create a new system within the set of nodes that it manages, it can be used to locate an existing system, and it can be used as a source of system creation events.

5.1.3 Portal Endpoint Properties

<i>Name</i>	<i>Type</i>	<i>Meaning</i>
muws-pl-xs:ResourceId	xsd:anyURI	Unique identifier for the portal
muws-pl-xs:ManageabilityCapability	xsd:anyURI	List of supported manageability capabilities.
StaticPortalStatus	StaticPortalStatusType	Static portal information; constant for the lifetime of the portal itself
DynamicPortalStatus	DynamicPortalStatusType	Dynamic server information; may be different on every time it is read
ActiveSystems	SystemReferenceListType	List of System EPRs
wsrf-nt:Topics	wsrf-nt:TopicExpressionType	List of supported notification topics
wsrf-nt:FixedTopicSet	xsd:boolean	A Flag to indicate whether the topic set is fixed
wsrf-nt:TopicExpressionDialects	xsd:anyURI	Dialect of the topicset

5.1.4 Portal Endpoint Operations

<i>Name</i>	<i>In</i>	<i>Out</i>
Create	hostname: xsd:string	wsa:EPR
	Create a system; hostname is optional	
LookupSystem	resourceID: xsd:anyURI	wsa:EPR
	Look up a single system returning its System EPR.	
Resolve	resourceID: xsd:anyURI, xsd:string	xsd:any
	Lookup a system and resolve a path against it.	
GetResourceProperties	wsrf-rp: GetResourcePropertyRequest	wsrf-rp: GetResourcePropertyResponse
	Get the value of a resource	
GetMultipleResourceProperties	wsrf-rp: GetMultipleResourcePropertiesRequest	wsrf-rp: GetMultipleResourcePropertiesResponse
	Read multiple resources	
Subscribe	wsrf-nt:Subscribe	wsrf-nt:SubscribeResponse
	Subscribe to events	
GetCurrentMessage	wsrf-nt: GetCurrentMessageRequest	wsrf-nt: GetCurrentMessageResponse

<i>Name</i>	<i>In</i>	<i>Out</i>
	Get the current message for a topic	

If a Portal Endpoint has a managed lifetime, then it may also extend the endpoint with the WS-ResourceLifetime properties and operations.

5.3 System Endpoint

This represents a system that has been created on the hosts managed by a portal. System EPRs are obtainable by creating one at the Portal Endpoint, or through a `LookupSystem` operation offered by the Portal.

5.1.5 System Endpoint Properties

<i>Name</i>	<i>Type</i>	<i>Meaning</i>
<code>muws-pl-xs:ResourceId</code>	<code>xsd:anyURI</code>	unique identifier
<code>muws-pl-xs:ManageabilityCapability</code>	<code>xsd:anyURI</code>	List of supported manageability capabilities.
<code>SystemState</code>	<code>cmp:LifecycleStateEnum</code>	current system state
<code>StateInfo</code>	<code>xsd:string</code>	Text state info
<code>SystemExtendedState</code>	<code>cmp:UnboundedXMLAnyNamespace</code>	Component state
<code>CreatedTime</code>	<code>xsd:dateTime</code>	Time system was created
<code>StartedTime</code>	<code>xsd:dateTime</code>	Time system was terminated
<code>TerminatedTime</code>	<code>xsd:dateTime</code>	end time (not present until system is terminated)
<code>TerminationRecord</code>	<code>cmp:TerminationRecordType</code>	termination record (present after termination)
<code>Topics</code>	<code>wsrf-nt:TopicExpressionType</code>	List of supported notification topics
<code>wsrf-nt:FixedTopicSet</code>	<code>xsd:boolean</code>	A Flag to indicate whether the topic set is fixed
<code>wsrf-nt:TopicExpressionDialects</code>	<code>xsd:anyURI</code>	Dialect of the topicset

5.1.6 System Endpoint Operations

<i>Name</i>	<i>In</i>	<i>Out</i>
<code>Initialize</code>	<code>job JobDescriptorType</code> <code>descriptor</code> <code>DeploymentDescriptorType</code>	<code>void</code>
	Initialize a system; pass in the job and component descriptors and build up the component graph.	

<i>Name</i>	<i>In</i>	<i>Out</i>
AddFile	mimetype xsd:string data xsd:base64Binary	xsd:anyURI
	Add a file to this document so that it is accessible by a URI from within the deployment descriptor.	
Run	void	void
	Start running an initialized system	
Ping	void	StatusType
	Probe a system's health.	
Resolve	xsd:string path	xsd:any
	Resolve a reference relative to this system. Can return EPRs to components; string or other data	
Terminate	xsd:string Message	void
	Terminate a system; pass in a message	
wsrf-rp:Destroy		
	Destroy the System Endpoint, terminating the System if it is not yet terminated	
wsrf-rp:GetResourceProperties	wsrf-rp: GetResourcePropertyRequest	wsrf-rp: GetResourcePropertyResponse
	Get the value of a resource	
wsrf-rp: GetMultipleResourceProperties	wsrf-rp: GetMultipleResourcePropertiesRequest	wsrf-rp: GetMultipleResourcePropertiesResponse
	Read multiple resources	
wsrf-nt:Subscribe	wsrf-nt:Subscribe	wsnt:SubscribeResponse
	Subscribe to events	
GetCurrentMessage	wsrf-nt: GetCurrentMessageRequest	wsrf-nt: GetCurrentMessageResponse
	Get the current message for a topic	

6 Portal

6.1 Portal Properties

6.1.1 *muws-p1-xs:ResourceId*

This is a MUWS-defined property. It contains a URI which uniquely identifies this instance of a Portal Endpoint.

6.1.2 *muws-p1-xs:ManageabilityCapability*

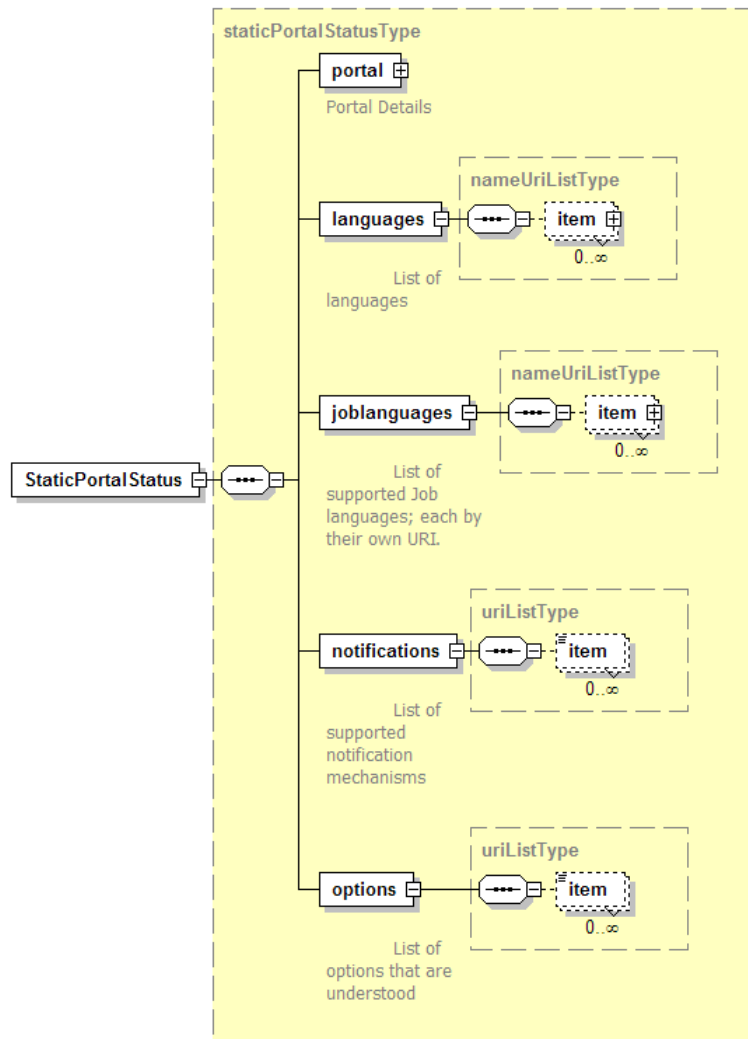
This is a MUWS-defined (multiple) property that lists all MUWS-related management features implemented in this endpoint.

The minimum set of properties that an endpoint must report are the `ResourceId` and `ManageabilityCapability` facilities themselves, as normatively described in the MUWS specification(s), and non-normatively as

```
<muws-p1-xs:ManageabilityCapability>
http://docs.oasis-open.org/wsdm/2004/12/mows/capabilities/ ↗
ManageabilityReferences
</muws-p1-xs:ManageabilityCapability>
<muws-p1-xs:ManageabilityCapability>
http://docs.oasis-open.org/wsdm/2004/12/muws/capabilities/ ↗
ManageabilityCharacteristics
</muws-p1-xs:ManageabilityCapability>
```

6.1.3 *StaticPortalStatus:*

This property contains static portal information; information constant for the lifetime of the portal instance. The elements contain static diagnostics information, such as product name and time zone.



The information lists are all lists of URIs that can be used to determine features.

6.1.4 **DynamicPortalStatus**

This is any dynamic status information. Implementations may include any information that they wish.

It is recommended that implementations provide information which will aid with diagnostics of any deployment problem, such as the versions of libraries used in the application, and other state information —though not any information that could expose security information.

6.1.5 **ActiveSystems**

This is a list of deployed systems which the portal is aware of. This may include systems which the portal did not deploy, but which a peer portal has deployed. It may also be restricted to those systems to which the caller has access rights. Network partitioning and other events may cause systems to be temporarily invisible to this list, and return later. Callers must view the list not as complete and accurate, but as a snapshot enumeration of all deployed systems that were visible at the time the request was processed.

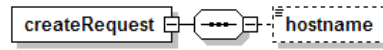
5.1.7 **Topic, FixedTopicSet, TopicExpressionDialects**

These three properties are published in adherence with the WS-BaseNotification specification.

6.2 Operations

6.2.1 *Create([hostname])*

This requests the creation of a new system instance, ready for configuration.



The `hostname` element specifies an optional hostname. If set, it nominates a host onto which the port should instantiate the system, and hence the System Endpoint. If unset, or if the identified host is deemed unsuitable/unavailable, the portal can instantiate the system on any host of its choosing. Thus, `hostname` is merely a hint, a hint to improve availability and performance.

The successful response is a System EPR to the instantiated System, an EPR which can be immediately used for direct communications. Creation of a System Endpoint is therefore a synchronous operation.

If any entity is registered with the portal for creation events, then the portal must send notification to that entity that new system has been created. The notification must not be sent until the system is ready for direct communication. There is no specification of the ordering of returning from the `create` operation and the sending of any notification mechanism. If there are multiple portals supporting deployment to a cluster of nodes, notification events *may* be sent to listeners on one portal, even if the deployment was requested on the other.

6.2.2 *LookupSystem(ResourceId:uri)*

This operation resolves a `ResourceId` to a system, and returns a System EPR.

6.2.3 *Resolve(ResourceId:uri, path:string)*

This operation resolves a `ResourceId` to a system, and then resolves a path against it. It has the same semantics as using `LookupSystem` to obtain an EPR, then invoking `Resolve` on that EPR.

6.2.4 *GetResourceProperty/GetMultipleResourceProperties*

These two operations are defined by the WS-ResourceProperties specification.

6.2.5 *Subscribe/GetCurrentMessage*

These two operations are defined by the WS-BaseNotification specification.

7 System

The System Endpoint represents the deployed system. After creation, it is still undefined, and must be configured before it can be moved to a running state.

7.1 System Properties

7.1.1 *muws-p1-xs:ResourceId*

This is a MUWS-defined property. It contains a URI that is unique to a particular instance of a system..

7.1.2 *muws-p1-xs:ManageabilityCapability*

This is a MUWS-defined (multiple) property that lists all MUWS-related management features implemented in this endpoint.

7.1.3 *CreatedTime/StartedTime/TerminatedTime*

These are all `xsd:dateTime` timestamps of when a system entered a particular state.

7.1.4 SystemTerminationRecord

This contains a `cmp:terminationRecordType` record. This It contains information about the reason for the system's termination. It is only present after a system has been terminated.

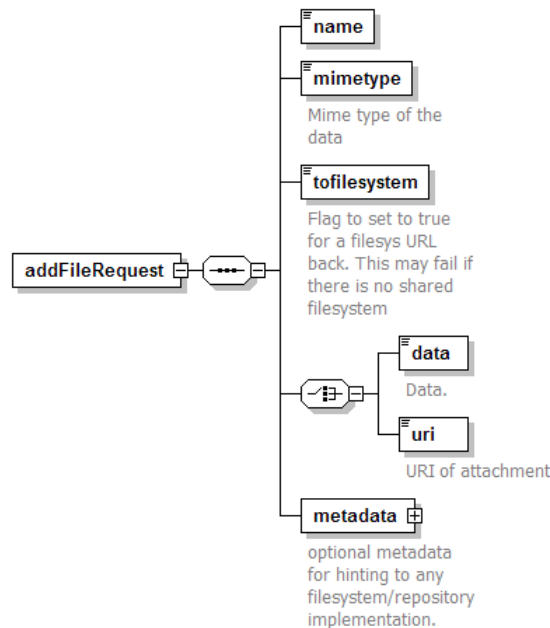
7.1.5 Topic, FixedTopicSet, TopicExpressionDialects

These three properties are published in adherence with the WS-BaseNotification specification.

7.2 System Operations

7.2.1 AddFile

This request uploads a file to the infrastructure, such that it is visible by deployed programs, and by the System Endpoint itself.



The request can include the file as base-64 encoded data. Unless both ends of the communication are specially written to stream large base-64 elements directly to and from storage, the `addFileRequest/data` contents must be sent as using the MTOM transmission mechanism. If DIME or Soap with Attachments is used then the attachment must be assigned a URI, a URI that must then be declared in the `addFileRequest/uri` element. Any or all of those mechanism may be implemented, though of course DIME, is somewhat deprecated.

Implementations *should* resolve remote URL references in the `addFileRequest/uri` element, using the delegated identity of the created job to grant access rights. This should be an asynchronous retrieval.

The `tofilesystem` flag in the request indicates whether the request should result in a URI of the `file:` schema. This is only possible if the deployment targets have a shared, distributed file system. If this is not the case, the request must result in an error.

The request supports a metadata element that contains arbitrary XML. This could be RDF metadata, file hash values for efficient re-use, or even file lifetime hints. All such metadata is implementation-specific, and is not defined in this edition of the deployment API specification.

Note that the WSDL accompanying this document does not declare how binary attachments are to be sent with the document. If the MTOM transmission mechanism is used [MTOM], then the `addFileRequest/data` element must contain the binary-marshaled data.

The response returns a URL to the uploaded file, a URL of a schema type `file:` or `http:`. The URL must be visible to all programs deployed in this system. It may be visible to other programs running with the same credentials, but this can not be guaranteed. If exposed as a file, it must be visible, read-only to all processes of the system, on any node in the network onto which it has been deployed.

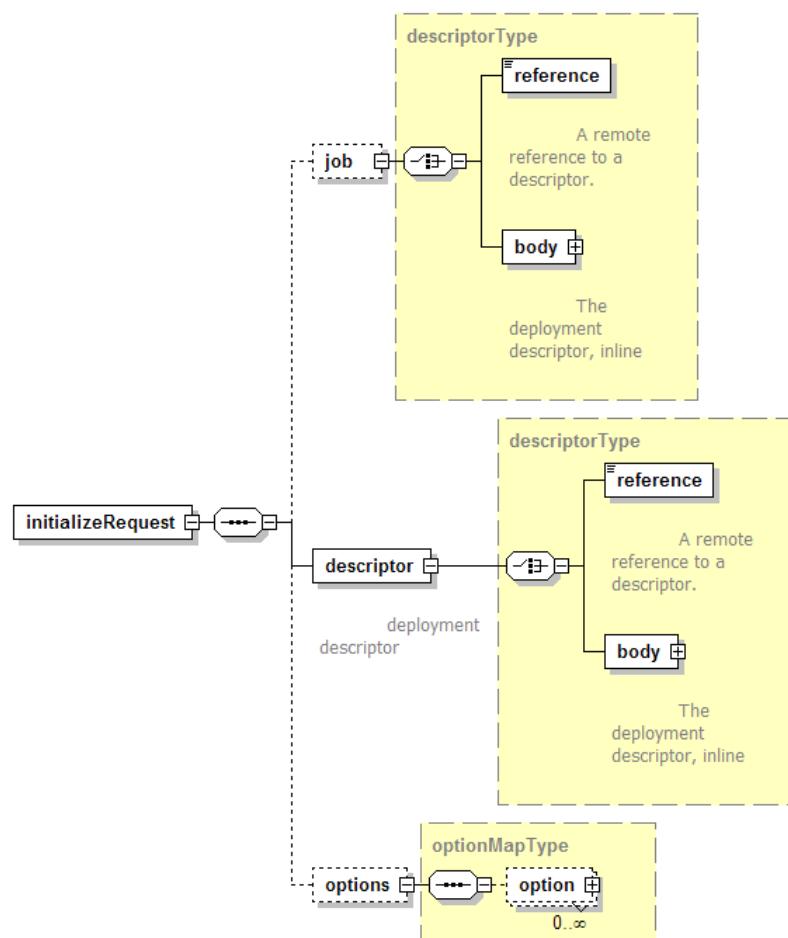
The lifespan of the uploaded file is bound to that of the created system; when the System Endpoint is destroyed, all uploaded files may be destroyed.

There is no guarantee of high-availability in deployment; failure of a single node may render the URL unreachable.

It is an error to call `AddFile()` with two files of the same URI within the same deployment. The second request must fail with an appropriate fault.

7.2.2 Initialize

This is a complex request, as it configures the system and moves it into the *initialized* state.



A deployment descriptor must be supplied; it consists of a language URI, and either an inline deployment descriptor or a URL to a location where the descriptor can be retrieved.

An optional `<job>` element contains the job description that was used when submitting the job to the front-end portal. As with the `<descriptor>`, it is of type `descriptorType`; it must have a language URI and either an inline body or a URL to the descriptor. The interpretation of this data by the service implementation is undefined.

The optional `<options>` element contains a list of zero or more configuration options. These are late-binding parameters to the deployment request, or to the deployment runtime.

When the request message is received, the System Endpoint must validate it (synchronously) and initialize the system. For CDDLM implementations, initialization implies that the deployment descriptor and job descriptor may be retrieved (if needed) and parsed. The application is then configured, and resolution begins. If successful, the system enters the *initialized* state. This is an asynchronous operation.

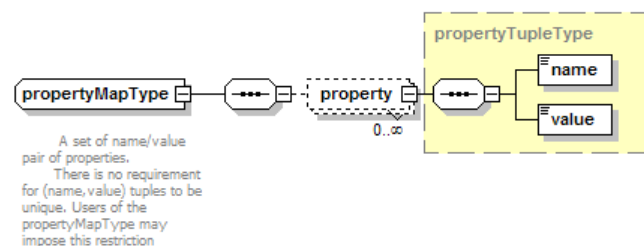
The response to a successful request is an empty response, `<initializeReponse/>`. Its presence implies that the initial validation was successful, and that initialization has begun, or has at least been scheduled.

If an initialize request is received and the application is in any state other than created, an error must be raised.

If the request is received while the application is already initializing itself, the contents of the message should be compared to that received previously. If the message is the same, then an `<initializeReponse/>` message should be returned. If they are different, that is, the caller is attempting to configure the System differently, a fault indicating this fact must be raised.

5.1.7.1 The *propertyMap* schema type

To aid those options that take a map of name/value pairs, there is a predefined XML Schema type that can represent the construct:



The `propertyMap` elements can be placed into the `<data>` child element of an option. Both the name and value of a `propertyTuple` within a `propertyMap` element are of type `xsd:string`; individual options are free to declare extra restrictions on the value of properties, restrictions which can be validated when processing the option.

There is no requirement that the name/value pairs are unique within a `propertyMap` element; that is also a restriction that can be declared in a specification of a particular option.

7.2.3 Run

This request runs a system. This triggers an asynchronous action, as it may take some time to enter the running state. It is only valid from a state in which the lifecycle permits running to be reached; *initialized* and, implicitly, *running*. In the case of the latter, the operation is a no-op. If the system is initializing itself, as a result of an

Initialize request, the request should be queued for processing after the state transition is completed.

The response is an empty element. A response means that the system has been queued to enter the running state asynchronously, or that it now is in that state.

7.2.4 Ping

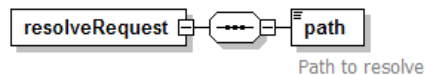
This is a synchronous request to the system, to query its health.

If the system is not running, the System Endpoint must return with the current state, as defined in the component model. In these states, the System Endpoint is free to provide whatever extended state information that it chooses.

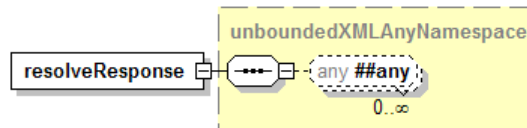
If the system is running, the request must be forwarded to the deployed system, which can return any extended state information, alongside the state “running”. This effectively acts as a liveness test upon the system. A successful response to the call implies that the system considers itself healthy.

7.2.5 Resolve

This operation resolves a path and returns its value or an error. It *must* be a valid operation when a system is initialized or running. It *may* be valid in a failed or terminated system.

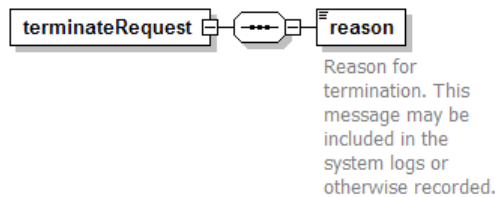


The response is arbitrary XML data, the contents of which depend upon what the path resolved to.



7.2.6 Terminate

This request terminates the system. To be idempotent, this call must not raise a fault when the system is already terminated, or when termination is in progress.



Upon receipt, the system must be terminated. Termination is asynchronous. The response is an empty element.

7.2.7 Destroy

The WS-ResourceLifetime `<wsrf-rl:Destroy/>` operation destroys the System endpoint itself. All files uploaded are destroyed, and the system is terminated if it is not already terminated.

After sending this message and receiving a response, service consumers should not make calls of the endpoint, as it may not be valid.

5.1.8 *GetResourceProperty/GetMultipleResourceProperties*

These two operations are defined by the WS-ResourceProperties specification.

5.1.9 *Subscribe/GetCurrentMessage*

These two operations are defined by the WS-BaseNotification specification.

8 Notification

Notification enables front-end applications to receive notification when a system changes state. It also enables management tools to track the number of running systems.

All implementations of the deployment API must support the WS-BaseNotification notification mechanism. The implementations are free to implement alternate mechanisms; that is beyond the scope of this document. What is covered, however, is a means of listing all notification mechanisms supported by an implementation. Every server instance is required to enumerate all supported mechanisms in a list included in its static server information property.

8.1 Notification Policy

- Implementations *must* support WS-Notification version 1.2-draft-01.
- Implementations *may* support alternate notification mechanisms.
- Implementations *must* list the URIs all supported notification mechanisms in the `StaticPortalStatus/notification` list.
- The URI of the supported version WS-Notification is
`http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotification-1.2-draft-01.xsd`
- Implementations *must* support the list of basic topics defined for each EPR type.
- Implementations *may* support any other notification topics.
- Implementations *may* also support Terminate notification events of WS-ResourceLifetime, which are raised after a EPR is destroyed.
- There is no requirement for fault-tolerant subscriptions. Implementations *may* include policy metadata that informs callers how to renew subscriptions in the event of system failure.

8.2 WS-Notification Support

As stated above, implementations *must* support WS-Notification; this does not prevent them also implementing supplementary mechanisms. There are specific topic spaces [WS-Topics] defined:

- Portal Endpoints must support a WS-TopicSpace that contains one topic: system creation events. This notifies callers that a new system has been created.
- System Endpoints must support the WS-TopicSpace and notifications defined in the Component Model specification. This includes a notification of changes in a system's lifecycle state.

8.3 Portal Notifications

The notification to be sent to listeners of a new System Endpoint is the following:

```
<xsd:complexType name="SystemCreatedEventType">
  <xsd:annotation>
    <xsd:documentation>Notification that a new
      System Endpoint has been created</xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="muws-pl-xs:ManagementEventType">
      <xsd:sequence>
        <xsd:element name="ResourceId" type="xsd:anyURI" />
        <xsd:element name="Reference"
          type="wsa:EndpointReferenceType" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

This notification supplies a System EPR referring to the created System, and identifies when the event occurred.

The normative declaration of the notification topic space is Appendix A.

8.4 System Notifications

System endpoints must support the lifecycle event notifications of the component model, `<cmp:LifecycleEvent/>`. The normative declaration of the notification topic space is Appendix A.

8.5 Fault-Tolerant Notification

Implementations are not required to provide fault-tolerant notification. The failure of portal may result in the loss of portal event subscriptions, and the failure of a system may result in the loss of system event subscriptions.

9 Fault Policy

Faults are based upon the WS-BaseFault model [WS-BF], taking on some of the lessons of [Loughran02], namely that extra information such as hostname and process is essential for locating which process among many has failed on a clustered system.

Faults are raised in response to errors either at the remote endpoint, in the local framework, or between the remote endpoint and other parts of the distributed system. They can be returned to callers in response to an operation on an endpoint, or sent as part of a notification event.

All faults that will be explicitly sent are derived from WS-BaseFault faults. Service implementations may implicitly raise SOAPFault faults, as that is inherent in most implementations.

9.1 Fault Categories

5.1.10 Service Faults

These are the faults that are raised by the service. They are grouped into a hierarchy of WS-BaseFault faults. There is a base fault class `DeploymentFault`, from which all others are derived.

All Service interfaces must declare that they raise these `DeploymentFault` instances, rather than list the specific faults. This is to provide forward extensibility.

The API lists specific subclassed faults of `DeploymentFault` that may be generated by a service or received by a client. These faults represent some of the faults that a service implementation may send.

If an implementation has a fault state whose meaning matches that of the predefined fault, the predefined fault must be thrown. If this predefined fault has standard elements for embedded fault information, they must be initialized with all the appropriate information, unless that information is unavailable. The implementation may add implementation-specific data within the `extra-data` element of the fault, to supplement this information. This extra data must not declare new types within the XML namespaces of the CDDLM specifications. The XML schema and semantics of this extra data should be documented.

If an existing fault type is not suitable, implementations may create new fault types. New fault types must extend the existing fault types which operations are declared as throwing. This effectively means that they must extend `DeploymentFault`. These new faults must not change the XML schemas of the deployment API, and they must be in a new namespace. The new faults and XML content should be publicly documented.

If an implementation adds new operations or properties at the existing endpoints, these new operations may raise whatever faults they see fit, within the constraints of the WS-BaseFault specification. Again, the implementation must not add new types to the deployment API namespace.

5.1.11 Transport faults

Transport faults will inevitably be raised as the appropriate fault for the system. For example, the Apache Axis SOAP client raises `AxisFault` faults for all SOAP events, wrapping stack trace and even HTTP Fault data within the fault as DOM elements. Microsoft .NET WSE has a similar fault class.

5.1.12 Relayed Faults

Relayed faults are those received by the far end and passed on. They may be WS-BaseFault Faults; HTTP error codes, SOAP faults, native language faults wrapped as SOAPFaults, or predefined deployment faults.

WS-BaseFault uses fault nesting for relaying faults; however, all faults must be a derivative of WS-BaseFault. This is addressed by defining a new WS-BaseFault derivative, a `WrappedSOAPFaultType`. This type is actually an extension of `CddlmFaultType`. This fault can nest any received SOAPFault, with an element containing the received XML data. Well-known elements in this fault data (such as the Apache Axis stack trace and HTTP fault code) should be copied into any fields in the main fault that fill the same role.

5.1.13 Fault Hierarchy

The UML representation of the fault hierarchy is shown in Figure 5 .

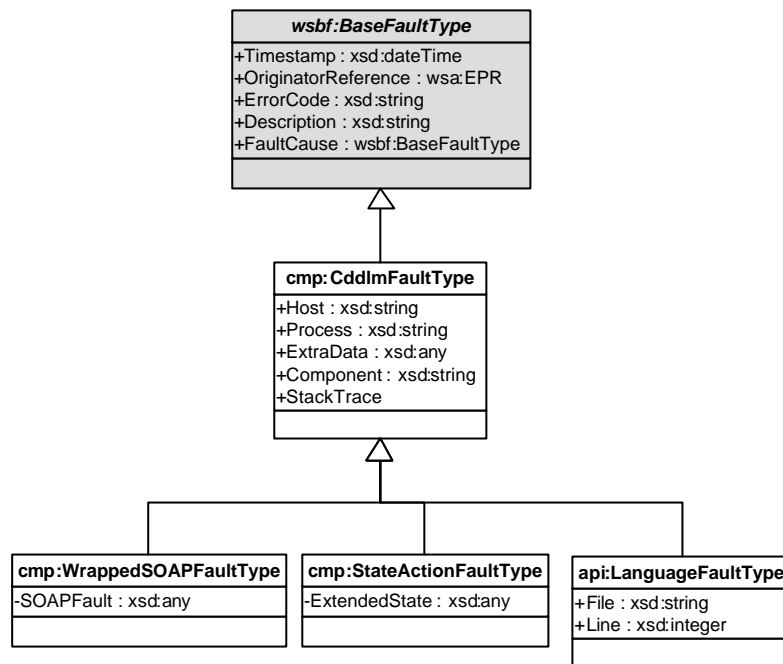


Figure 5 . Hierarchy of XSD datatypes used to describe faults

9.2 Fault Security

Sites offering deployment services, may, for security reasons, wish to strip out some information, such as stack trace data. Implementations should provide a means to enable such an action prior to transmitting faults to callers.

Host name and process information may be viewed as sensitive, yet again, this is exceedingly useful to operations. Implementations may provide a means to disguise this information, so that it does not describe the real hostname or process ID of a process, but instead pseudonyms that can still be used in communications with any operations team.

9.3 Internationalization

The WS-BaseFault specification makes no statement upon which language error descriptions are described.

If an implementation can return descriptions in one language, it must use `xml:lang` attributes to indicate the language of a description. Multiple descriptions, in different languages may be included. The client application should extract the description(s) whose language is the nearest match to that of the client.

9.4 Faults

9.4.1 CddlmFaultType

This type represents any fault thrown by the deployment infrastructure. All endpoint operations must declare that they throw this fault, and must not explicitly declare any derivative faults that they may throw.

<i>Element</i>	<i>Type</i>	<i>Meaning</i>
Host	xsd:string	Hostname or pseudonym

Process	xsd:string	Any process identifier suitable for diagnostics
ExtraData	cmp:unboundedXMLAnyNamespace	Extra fault data
Component	xsd:string	Path to component raising the fault
Stack	stringListType	Optional stack trace

Implementations must include a component reference if it is known. Implementations should include hostname and process information. Process information may be a low-level identifier (such as an operating system process ID), or it may be some application specific identifier. Its role is merely to distinguish which process amongst many in a load-balanced implementation raised the fault.

9.4.2 LanguageFaultType

A language fault represents any fault in language processing for which a file and line number are relevant.

<i>Element</i>	<i>Type</i>	<i>Meaning</i>
File	xsd:string	Filename/URI of file at fault
Line	xsd:integer	Line number within the file

If the error is in the inline deployment descriptor, the `File` element must be empty "" or omitted. Furthermore, the `Line` element must be relative not to the deployment request, but to the inline descriptor. Recipients of faults can then infer from the empty/absent file element that the fault was in the inline request.

Note that a consequence of this design is that implementations should preserve white space in the deployment descriptor when saving them to file.

9.4.3 WrappedSOAPFaultType

This type represents a mapping of a classic W3C `SOAPFault` [SOAP1.2] to a WS-BaseFault, as an extension of `DeploymentFault`. It adds two new elements to contain data unique to SOAPFaults.

<i>Element</i>	<i>Type</i>	<i>Meaning</i>
SoapFault	s12:Fault	Fault code information

The normative mapping of `SOAPFault` elements to `WrappedSOAPFault` elements is as follows:

<i>SOAP1.2</i>	<i>WrappedSOAPFault</i>
/s12:Fault	WrappedSOAPFault/api:SoapFault
SOAP endpoint	WrappedSOAPFault/wsrf-bf:originator

The SOAP endpoint must be translated into a `wsa:EndpointReference` if it is a simple URL/SOAPAction tuple.

Detail from SOAP stacks with well-known fault fields, such as the Apache Axis stack trace, may be imported into appropriate fields in the `DeploymentFault`.

9.5 Fault Error Codes

Specific fault error codes, and their meaning, will be covered in a separate informative document.

10 Implementation Requirements

Implementations may validate incoming requests against the XML Schema used to describe this service. If this is not done, the implementation should validate message by other means. It is an error if a required element is not included in a request, or it occurs more than is permitted.

Instances of system and portal endpoints must be re-entrant.

Implementations may provide the XSD/WSDL of the endpoints using the de-facto standard of `GET endpoint+"?wsdl"`, or by using some other mechanism.

The generation and processing of SOAP messages and HTTP error codes (if using SOAP over HTTP), must be in accordance with the WS-I Basic Profile 1.1 specification.

There are number of places in the specification in which the contents of remote URLs are to be retrieved. This retrieval must also be re-entrant, that is, any caching mechanism must be thread-safe. Furthermore, all such requests should use HTTP/1.1, implement a time-out mechanism on downloads, verify the length of retrieved data against the content-length header (which is required by HTTP1.1), and fail if an error occurs.

11 Security

The deployment requests must only be granted by suitably authorized individuals, or their suitably authorized agents. For deployment to a Grid infrastructure, that means that the standardized security model of the infrastructure must be used to authenticate callers. Only callers with the relevant rights may deploy systems.

When delegating deployments across nodes, the node issuing the deployments needs to have the rights to do so, and the deployment itself still needs to be authenticated as a legitimate request of the sender.

Along with deployment, the ability of a caller to list and manipulate running systems, introduces another security issue: that of who has access to the set of deployed systems.

Files uploaded via `addFile` must only be visible to the deployed application, and potentially other applications deployed under the same credentials. There must also be a limit to the total size of files uploaded by a single user, if a denial of service attack on the file system is to be prevented. Quota-enabled filesystems are one possible solution.

There are a number of places in the system in which remote URLs to data may be supplied, as an alternative to sending the information inline. In these situations, the service implementation must not retrieve this content with greater rights than that of the caller. Furthermore, to ensure that the content is that which the caller has chosen to publish, the HTTPS/TLS protocols should be preferred over HTTP, unless the downloaded content is itself authenticated by some form of signing mechanism.

12 Editor Information

Steve Loughran, HP Laboratories

steve_loughran@hpl.hp.com

13 References

13.1 Normative References

- [Goldsack04] Goldsack, *SmartFrog Language*, 2004
- [MOWS] Sedukhin I. et al, [Web Services Distributed Management: Management of Web Services \(WSDM-MOWS\) 1.0](#), OASIS, 2004.
- [MTOM] Gudgin, M., [SOAP Message Transmission Optimization Mechanism](#), W3C, 2005.
- [RFC2119] S. Bradner, RFC 2119 - Key words for use in RFCs to Indicate Requirement Levels, 1997
- [Schaeffer05] Schaeffer., CDDLM Component Model Specification, 2005
- [SOAP1.2] W3C, [SOAP Version 1.2](#), 2003.
- [XML-CDL] CDDLM XML Configuration Description Language Specification version 1.0 draft 2004-12-10.
- [WS-A] Gudgin, M. and Hadley S., *Web Services Addressing -Core*, 2004.
- [WS-BF] Tuecke et al., *Web Services Base Faults (WS-BaseFaults)*, 2004.
- [WS-BaseNotification] Graham et al., *Web Services Base Notification 1.0 (WS-BaseNotification)*, 2004.
- [WS-BrokeredNotification] Graham et al., *Web Services Brokered Notification 1.0 (WS-BrokeredNotification)*, 2004.
- [WS-Policy] Schlimmer et al., *Web Services Policy Framework (WS-Policy)*, 2004
- [WS-ResourceLifetime] Frey et al., *Web Services ResourceLifetime 1.1 (WS-ResourceLifetime)*, 2004.
- [WSRF] Tuecke et al., *Web Services Resource Framework (WSRF)*, 2004.
- [WS-ResourceProperties] Graham et al., *Web Services Resource Properties 1.1 (WS-ResourceProperties)*, 2004.
- [WS-ServiceGroups] Graham et al., *Web Services Service Group Specification 1.0 (WS-ServiceGroups)*, 2004.
- [WS-Topics] Graham et al., *Web Services Topics (WS-Topics)*, 2004.

13.2 Non-Normative References

- [Axis] Apache Software Foundation, *Apache Axis*,
- [DIME] Nilesen et al., [Direct Internet Message Encapsulation](#) Microsoft, 2002.
- [Foster04] Foster et al., *Modeling Stateful Resources with Web Services*, 2004.
- [GlobusRSL] Globus, [Resource Specification Language](#), 2004
- [JSDL] Job Service Description Language, 2004.
- [Loughran02] Loughran, [Making Web Services that Work](#), HP Laboratories, TR-HPL-2002-274, 2002.
- [Parastatidis03] Parastatidis et al., *A Grid Application Framework based on Web Services Specifications and Practises*, University of Newcastle, 2003.

Appendix A: Event Topics

The normative listing of the Portal event topics is

```
<wstop:TopicSpace name="SystemNotificationTopics"
  targetNamespace=
"http://www.gridforum.org/cddlm/deployapi/2005/02/events/system"
  xmlns:api="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
  xmlns:wstop=
"http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd"
  >
  <wstop:Topic name="SystemCreatedEvent"
    messageTypes="api:SystemCreatedEvent">
    </wstop:Topic>
</wstop:TopicSpace>
```

The normative listing of the System event topics is

```
<wstop:TopicSpace name="SystemNotificationTopics"
  targetNamespace=
"http://www.gridforum.org/cddlm/deployapi/2005/02/events/portal"
  xmlns:api="http://www.gridforum.org/cddlm/deployapi/2005/02"
  xmlns:cmp="http://www.gridforum.org/cddlm/components/2005/02"
  xmlns:wstop=
"http://docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.xsd"
  >

  <wstop:Topic name="LifecycleEvent"
    messageTypes="cmp:LifecycleEvent"/>

</wstop:TopicSpace>
```