

# Solution Deployment Descriptor (SDD), Part 1: An emerging standard for deployment artifacts

Skill Level: Intermediate

[Julia McCarthy \(julia@us.ibm.com\)](mailto:julia@us.ibm.com)  
Software Architect  
IBM

[Brent A. Miller \(bamiller@us.ibm.com\)](mailto:bamiller@us.ibm.com)  
Autonomic Computing Chief Architect  
IBM

29 Apr 2008

The Solution Deployment Descriptor (SDD) is an emerging standard for a set of XML documents that define deployment metadata about deployment artifacts and the aggregation of deployment artifacts. Externalizing deployment knowledge that has been more commonly buried in code or documentation provides multiple benefits. Consumers of SDDs, including humans and software, can use the knowledge provided about both the requirements for and results of successful deployment to better plan for and execute successful changes to their software environments. This article describes the SDD and provides a high-level overview of the support provided.

## Introduction

The Solution Deployment Descriptor (SDD) is an emerging standard from the Organization for the Advancement of Structured Information Standards (OASIS). SDD defines the format for *metadata* about the requirements, inputs, and results of the deployment artifacts that are processed to deploy software resources. SDD also defines the format for metadata about the *aggregation* of multiple deployment artifacts into a solution. This metadata can be used to guide deployment as well as a variety of other deployment-related activities, such as deployment planning. This article presents the concepts behind SDD and describes what SDD is.

The article discusses SDD and provides a high-level overview of the support

provided by the SDD for the expression of deployment-related knowledge through standardized, externalized metadata. The article is written for those who want to understand where the SDD fits into the deployment world and those who intend to learn the details of the SDD standard, especially those who are involved in developing, integrating, or deploying software. A general knowledge of the current state of software deployment technology, especially in complex environments, is assumed.

Future articles will describe in more detail how to use the SDD. In these articles you will explore the elements of the SDD that realize the concepts described in this introductory article. This and future articles will complement, but not replace, the SDD specification and associated documentation published by OASIS (see [Resources](#)).

## What SDD is

Fundamentally, SDD provides a standard way to encode and externalize deployment information. One Solution Deployment Descriptor consists of two paired XML descriptor files, a package descriptor, and a deployment descriptor, that contain deployment metadata; that is, descriptive information about deployment artifacts and the aggregation of those artifacts. The package descriptor describes the identity and contents of the deployment package. The deployment package consists of a set of artifacts used to perform deployment lifecycle operations on a group of related resources that make up a solution. Artifact processing results in the deployment of software. The deployment descriptor describes the inputs, requirements, variability, and results of the deployment artifacts.

The package descriptor describes the contents of the deployment package, but does not describe or assume anything about the format of the deployment package. Anything in any location that can be identified by a URI (Uniform Resource Identifier) can be part of the deployment package contents. The package descriptor's content list contains, at a minimum, the URI of the deployment descriptor for the package. It also contains URIs for all deployment artifacts and supporting files. Other types of content that can be identified include readme files, license agreements, package descriptors of aggregated SDDs (aggregation is described later), and bundles of language translations for the SDD. In general, the contents include everything required for the software to be deployed, whatever that might be.

The deployment descriptor provides information about the *artifacts* used to perform the deployment. It defines base, selectable, or localization content for each supported deployment operation. *Base content* consists of metadata about artifacts that are processed to accomplish some deployment operation for the core content of the software that is being deployed. *Selectable content* consists of metadata about artifacts that are processed to act on content that can vary depending on selections made during a specific deployment. *Localization content* consists of metadata about

artifacts that are processed to act on resources that localize the software to support multiple languages.

A deployment descriptor supports *aggregation* (described later) of other SDDs by defining metadata about requirements, conditions, inputs, and results associated with the aggregation. Artifacts that conceptually make up the base, selectable, and localization content of the top-level software may actually be defined in the aggregated SDDs.

## What artifacts are

Deployment *artifacts* are package contents that can be processed to create or modify software resources in the deployment environment. These resources collectively make up the software whose deployment is described by the SDD and include items such as executable files and database table definitions. Examples of deployment artifacts are Linux® RPM files, Microsoft® MSI files, setup.exe, ZIP, and custom installation executable files. The SDD emerging standard does not define, or in any way replace, deployment artifact formats.

Artifacts can be either atomic or monolithic. *Atomic artifacts* (which should not be confused with *atomic content units*) are processed to accomplish particular deployment operations including installation, configuration, and localization (language translations); that is, they can be installation artifacts, uninstallation artifacts, configuration artifacts, and so on. Atomic artifacts are described by the *content units* defined for each of these operations (*installable unit*, *configuration unit*, and *localization unit*). Atomic artifacts themselves do not expose selectable content. Atomic artifacts can be aggregated, in which case the aggregated artifacts are described by *composite installable* content units.

*Monolithic artifacts* are single artifacts that expose selectable content. Monolithic artifacts are processed by deployment software that can select the appropriate content to deploy from the single artifact (contrast this with selecting one or more particular atomic artifacts during the deployment process). Like atomic artifacts, monolithic artifacts are processed to accomplish particular deployment operations and are described by the content units defined for those operations.

The SDD supports both atomic and monolithic artifacts that can be rendered in a format appropriate for the deployment software. For example, the deployment of server software that provides an optional feature-selected client could be implemented with one atomic artifact to deploy the server and a second atomic artifact to deploy the client. Alternatively, it could be implemented with a single monolithic artifact that receives the client feature selection as an input. The SDD can be used to describe either of these cases. The SDD can also be used to describe content that requires both atomic and monolithic artifacts. Atomic artifacts are preferred because they allow for a more complete expression of the deployment characteristics of the software. For example, the results of deploying the client

feature can be distinguished more completely from the results of deploying the server feature when atomic artifacts are used. SDD support for monolithic artifacts is provided primarily to allow adoption of the SDD in situations that require monolithic artifacts for other reasons, such as when treating an entire installation program as an artifact.

## What aggregation is

*Aggregation* is the composition of metadata about multiple deployment artifacts that function together to deploy a software solution. This can be a simple aggregation of a few artifacts that make up a small software product or it can be an aggregation of multiple complex software products into a large solution. Any set of artifacts that have value when delivered in a single deployment package can be aggregated in an SDD.

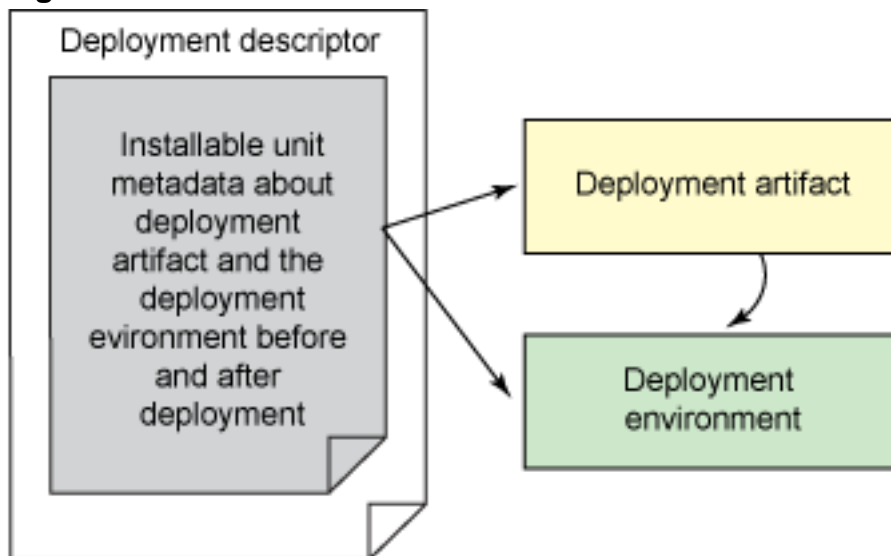
One way that the SDD supports aggregation is by allowing metadata about multiple deployment artifacts to be defined within a single SDD. Another way that aggregation is supported is by allowing one SDD to identify another SDD as part of its contents. This effectively creates a hierarchy of SDDs with a single SDD at the root of the hierarchy. The individual constituent SDDs could be deployed independently, but aggregating them enables a more complete description of the overall solution.

Not all artifacts described by a single SDD or SDD hierarchy are necessarily used in any particular deployment. The SDD can express variable deployment results that depend on specific environmental conditions, deployer inputs, and deployer selections that are present during a particular deployment. *Features* defined in the SDD support artifact selection by enabling optional functions to be selected by the deployer. *Conditions* defined in the SDD support artifact selection based on environmental conditions, such as the type of the operating system. *Variables* and *Arguments* support variable input to artifact processing that enables the processing to be customized to the specific needs of the particular deployment.

No matter how the aggregation is defined, or what variability is described, each particular deployment results in processing an unambiguous set of deployment artifacts. The result is the creation of the base, selected, and localized content of the software that is described by the SDD and is appropriate for the deployment environment.

## Metadata about artifacts

The metadata in the deployment descriptor enhances the usability of the deployment artifacts by describing and externalizing the system state and software required for processing the artifacts; by associating user inputs and resource properties from the deployment environment with inputs to artifact processing; and by specifying the results of artifact processing.

**Figure 1. An artifact and associated metadata**

As shown in Figure 1, an artifact is deployed into a deployment environment. The SDD contains metadata about artifacts, and artifact metadata includes:

- **Identity:** This metadata identifies the packaging unit containing the artifact (if any). For example, an SDD might specify identity information including the name and supplier of an artifact that is packaged in the SDD for deployment. The package descriptor contains the identity of the complete package. Identity is used in *Installable Units* when it is useful to identify the portion of the package associated with the artifact. Identity is optional.
- **Inputs and outputs:** These can be user provided, obtained from system state information, defined in the SDD with fixed values, or any combination of these. For example, an SDD might specify an input that is the root installation directory to be used for installing the software, along with a default value for that input that is derived from the operating system's standard installation location property (for example, "C:\Program Files" on Windows®). User input can be enabled to override the default and specify a different root installation directory.
- **Conditions:** Conditions are system state information, expressed in terms of resources, that describe circumstances that can vary across deployments. For example, an SDD might specify a condition based on the operating system type of the deployment environment that results in selecting the appropriate artifact to deploy for that operating system type.
- **Requirements:** These are system states that must exist for a successful deployment, expressed in terms of resources. For example, an SDD might specify a requirement for a particular version of a particular database to be present and usable in the deployment environment as a

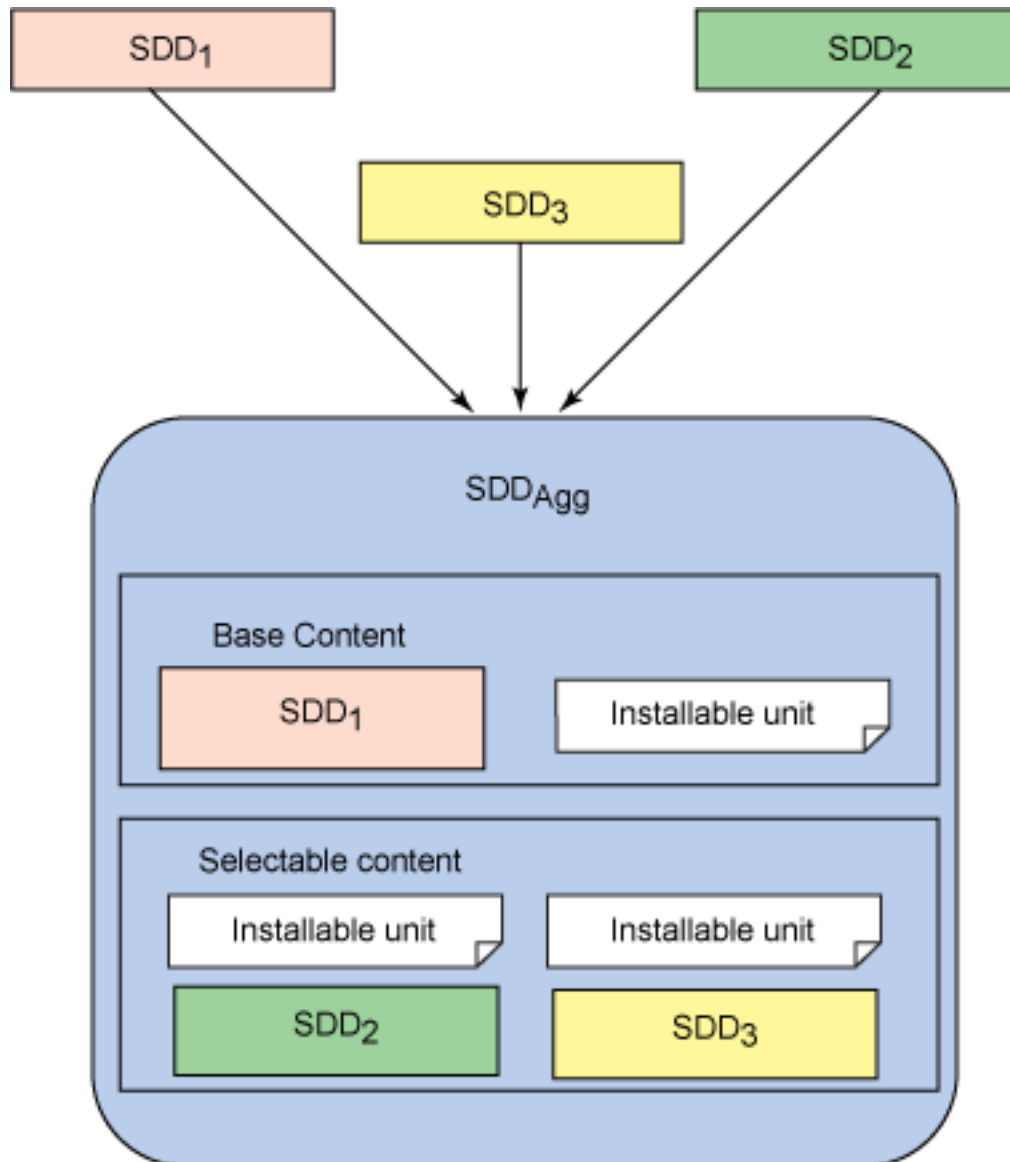
prerequisite to installing new software. The expression of requirements can include expressing alternative ways to satisfy the requirement.

- **Results:** These identify new and changed resulting resources that will be present in the environment after successful deployment. For example, an SDD might specify that its results include the presence of a new application after that application is installed using that SDD.

### **Metadata about aggregation**

The metadata about aggregation in the deployment descriptor enhances the ability of humans and software to understand the interactions, dependencies, and associations of the artifacts that make up the aggregation and the unique requirements that exist because of the aggregation.

### **Figure 2. Aggregation**



As shown in Figure 2, multiple SDDs and multiple Installable Units can be aggregated in a single SDD. Aggregation metadata includes:

- **Composite metadata:** This includes the identity, requirements, conditions, results, and inputs associated with the aggregation as a whole. When multiple artifacts are aggregated, information about the aggregation is needed in addition to the information about each individual artifact. Requirements, conditions, and inputs that are common to multiple artifacts can be "factored up" in the content hierarchy to avoid repetition and ease understanding.
- **Contained packages:** These are references to SDDs that are part of the aggregation. Contained package metadata includes input maps and

resource maps, as well as requirements associated with including the constituent SDD in the aggregation.

- **Resource maps:** These are associations between resources described in the aggregating SDD and resources described in an aggregated SDD. Resources defined for the solution need to be mapped to the resources defined for the individual parts of the solution. For example, a solution that uses one machine as a database server and another machine as an application server would map the operating system resource that serves as the solution's application server to the operating system resource in each of the SDDs that describe deployment of solution applications. It would also map the operating system resource that serves as the solution's database server to the operating system resource in each of the SDDs that describe the deployment or configuration of the database.
- **Input and output maps:** These are associations between variables in the aggregating SDD and input parameters in an aggregated SDD. This mapping is especially useful when inputs to individual artifacts overlap. For example, mapping inputs is useful when the same database administrator ID should be used by all artifacts that interact with the same database in the context of the solution.
- **Dependencies:** These are dependencies between artifacts. When used together in a solution, sometimes one artifact depends on another. That is, one artifact may need to be processed before another can be processed. For example, a configuration artifact that configures a database is likely to depend on an artifact that deploys the database. Dependencies between artifacts can be expressed in the SDD. The types of dependencies defined in the SDD are pre-requisite (artifact must be processed prior to the one expressing the dependency), co-requisite (artifact must be processed in the same particular deployment) or ex-requisite (artifact must not be used in the same particular deployment).
- **Base content:** This is a definition of the artifacts that are considered to be the core, or base, content of the software described by the SDD. There may be some variability in base content based on the system state, but there is no selectability of base content by the deployer. Note that artifacts may be directly defined within base content or they can be included using a contained package definition that aggregates another SDD.
- **Selectable content:** This is a definition of the artifacts associated with optional content. Artifacts within selectable content are processed only when explicitly selected by the deployer using features. As with base content, artifacts may be directly defined within selectable content or they can be included using a contained package definition that aggregates another SDD.
- **Features:** These are named groupings of artifacts defined in the

selectable content hierarchy. A solution may allow the deployer to optionally select some content. Features are used to associate deployer selections with some subset of the artifacts that make up the solution.

- **Localization content:** This is a definition of artifacts associated with localization software. Localization is the process of enabling software for a particular language by providing translations in that language and delivering software specialized for the use of that language. Localization content may be associated with the software described in the base and selectable content hierarchies or it may be used alone in an SDD to describe the deployment of a localization package (for example, to add new language capabilities to already deployed software). As with base and selectable content, artifacts may be directly defined within localization content or they can be included using a contained package definition that aggregates another SDD.

## The SDD defines what constitutes a successful deployment

As described in the previous sections, the SDD communicates the SDD authors' knowledge of "what" must be done to achieve a successful deployment. The SDD does not address "how" the deployment is accomplished. For example, the SDD may declare that a particular version of a particular operating system is required for successful deployment. How that information is discovered in the deployment environment and compared to the SDD's declaration to determine whether or not the requirement is met is outside the scope of the SDD. The SDD may declare that a particular artifact of a particular type will create a particular resource with particular property values when it is processed by a particular target resource. This tells the SDD consumer what the intended results are, but how the artifact and accompanying information is provided to the target resource for processing is outside the scope of the SDD.

The SDD encodes and externalizes the SDD authors' knowledge about the system state before and after a successful deployment as well as the artifacts and inputs required to achieve the resulting state. Software that consumes the SDD decides how to process this information and for what purposes the information will be used. Software that orchestrates deployment using information in the SDD makes many choices about how the deployment is achieved. *Profiles* (which are described later) help to bridge between the "what" described by the SDD and the "how" of the implementation by stating what environment-related values are present in the SDD that the implementation must know how to handle.

## Implementations define how the deployment is accomplished

As stated previously, implementations that consume SDDs have many implementation choices to make. These choices affect how SDDs are written and they can affect interoperability among implementations. A primary use of the

metadata in the SDD is to orchestrate deployment. The COSMOS runtime is one example of deployment software that uses the SDD metadata to orchestrate deployment. Note that this orchestration augments but does not replace the deployment actions that occur when artifacts are processed.

#### **SDD, IUDD, and Eclipse COSMOS**

Standards and implementations both need starting points. The starting point for the SDD standard was the Installable Unit Deployment Descriptor (IUDD). IBM submitted the IUDD to the OASIS SDD Technical Committee to serve as a basis for the SDD standard. Of course, SDD has many differences from IUDD, but it is an evolution of IUDD and retains many concepts and design principles of IUDD.

A starting point for implementations that can create and consume SDDs has begun within the Eclipse COSMOS (Community-driven Systems Management in Open Source) project. This project intends to produce an open source, basic reference implementation for SDD tooling and runtime code. The implementation developed in COSMOS can serve as a basis for other implementations. A common basic reference implementation can promote interoperability and accelerate adoption of the SDD standard.

See [Resources](#) for more information about IUDD and Eclipse COSMOS.

Using metadata about inputs and system state, along with information about the particular deployment environment, the deployment software can determine the artifacts to be used for a particular deployment. For example, the package might include artifacts for different operating systems. At deployment time, the deployment software gathers information about the platform for the deployment target and selects the appropriate artifact for that operating system. Similarly, the deployment software can gather information about language choices and then select the artifacts that support the specific languages to be used for this particular deployment

The analysis required for artifact selection includes the comparison of properties in the SDD that describe deployment environment state information, such as requirements, conditions, and property values, with the corresponding actual values that exist in the particular deployment environment. The more sophisticated the implementation, the larger the range of resources and resource properties it can discover. For example, a very simple implementation might be able to determine only the operating system type and version, whereas a more sophisticated implementation might be able to determine values for a very broad range of resources by using multiple sources of information such as system registries, file scanning, and configuration databases. But even the most sophisticated implementation is not likely to be able to evaluate every possible value that could be specified by every SDD author. Hence, implementations are encouraged to provide a mechanism to extend their capabilities in this respect. Software developers can then provide extensions for the capabilities required to process the broader range of

values in an SDD's metadata. A particular set of values supported by an SDD is called a *profile*. The SDD 1.0 specification does not specify a format for how to extend implementation capabilities, but it does provide an open schema model that allows most elements and attributes to be extended. Profiles also can be extended to expand the common vocabulary as implementations are extended.

Another deployment implementation choice is that of determining which types of artifacts can be processed, and once again, implementations can vary greatly in their sophistication. A simple implementation might support only one or two well-known artifact types such as ZIP files or Linux RPM files. A sophisticated implementation might support a dozen or more artifact types. As with environment evaluation, no implementation is likely to be capable of processing all artifacts that could be specified by all SDD authors. Hence, deployment implementation developers are encouraged to provide a mechanism to extend this capability to enable processing for additional artifact types similar to what was described earlier for environment evaluation.

### **Bridging "what" and "how"**

Like many standards initiatives, one goal of the SDD is to promote interoperability. Although the SDD by itself is not sufficient to ensure interoperability, it is a necessary and important part of enabling interoperability for deployment software. Bridging the "what" specified by the SDD to the "how" determined by implementations is a key element for interoperability.

Concretely, this implies that information specified in SDDs must be aligned with the capabilities of the deployment software that processes those SDDs. For example:

- If an SDD defines a requirement for Linux, that SDD can be processed only by implementations that understand the vocabulary used to describe Linux and also know how to determine whether / the deployment environment operating system is Linux.
- If an SDD defines a variable based on the trace setting of IBM® WebSphere® Application Server, that SDD can be processed only by implementations that understand the vocabulary used to describe the trace setting and also know how to discover and interpret that trace setting.
- If an SDD includes a Linux RPM artifact, then that SDD can be processed only by implementations that understand and can process such artifacts.

Similarly, all resources and artifacts, no matter how obscure, specified in an SDD need to be aligned with the deployment software capabilities.

*Profiles* define the vocabulary that enables SDD producers and consumers to accomplish this alignment for particular use cases and deployment environments.

For example, to specify requirements for different operating systems, the SDD producers and consumers must agree about how to specify those operating systems--that is, if the SDD specifies metadata for a Linux operating system, then producers and consumers need to agree on a common way to express "Linux", and similarly for other deployment environments. Moreover, requirements, conditions, inputs, outputs, and other items in an SDD may refer to information in the deployment environment. For example, requirements for disk space need to be expressed in the units, such as megabytes or blocks, that the deployment environment uses. Properties associated with processors, such as the processor type and speed, also need to be expressed in a common way.

Profiles define this common vocabulary by defining values used to specify properties associated with deployment environments. Any set of such values can be collected into a profile. A profile is not required to conform to the SDD specification, but profiles help to promote interoperability. Software that consumes SDDs can declare the profiles that it supports; SDD authors can then restrict the content of their SDDs to values defined in the profiles supported by the target deployment software. It is expected that the SDD community will develop profiles that support use cases and deployment environments that are commonly encountered.

The OASIS SDD Technical Committee has published a *starter profile* that defines values sufficient to address the various examples that are also published along with the SDD specification (see Resources); the starter profile does not cover all values that could be used in all SDDs. This starter profile uses the Distributed Management Task Force's (DMTF) Common Information Model (CIM) resource model (see [Resources](#)) as its basis. Other profiles can be generated to cover additional cases, and they may use other resource models as their basis.

Because myriad use cases and deployment environments exist, profiles by themselves cannot ensure the alignment that is required between SDD producers and consumers. An additional mechanism that can be used to achieve this alignment is to extend implementations so that they can understand and process a wider array of SDD constructs (requirements, variables, artifacts, and so on). The SDD starter profile, created and published by the OASIS SDD Technical Committee, recommends that implementations be written so that they can be extended for this purpose. So, for example, if an existing software deployment implementation cannot handle an IBM WebSphere Application Server trace setting, then that software could be extended to understand, discover, and interpret that setting so that it could process SDDs that declare such metadata. Because such extensions to the deployment software are themselves software to be deployed, these extensions could actually be included in an SDD so that the deployment software processes its own extensions. Those extensions, in turn, enable the deployment software to process metadata in that SDD that could not otherwise be handled.

The extensibility of the SDD schema, along with the use of profiles to define a common vocabulary, provides the flexibility needed to maintain the alignment

between SDDs and implementations. Interoperability among implementations can emerge naturally over time through convergence on extension methods and profiles proven in real-world situations.

## Uses of the SDD

Metadata in the SDD can be used for more than just deployment. The SDD precisely communicates the SDD author's knowledge about a successful deployment, but how the metadata is used and processed is up to the implementation. Functions such as environment analysis can be used for other deployment-related purposes that can benefit from the deployment metadata encoded and externalized in the SDD. Software can be created to achieve these other purposes using the SDD metadata. One example is deployment planning, which involves checking the deployment environment state prior to executing the actual deployment operations to determine what prerequisite software needs to be deployed. Such deployment planning can assist with scheduling deployments, especially when multiple deployments constituting a new release are planned. This can improve the change management and release management processes in data centers.

Another example of a deployment-related purpose that can leverage the SDD is software inventory. Information about resulting resources in the SDD can be harvested and can serve as one mechanism for determining what resources are installed on a system and what the requirements of installed resources are. Software that populates registries or configuration databases could be written to harvest this information during deployment. Note that this is an opportunity to derive additional value from SDDs. Registering information from the SDD is not a requirement for using SDDs.

## Conclusion

The SDD offers a standard way to record and externalize information about software deployment and lifecycle management--installation, configuration, updates and upgrades, localization, and uninstallation. SDDs capture knowledge from developers and solution aggregators that typically is not recorded and, hence, is typically not available during deployment. Even when such information is recorded and made available during deployment, it typically exists in a proprietary form.

The SDD improves this situation considerably by enabling SDD authors to record information about their artifacts, including how they are packaged, where they came from, and what is necessary in the deployment environment, such as disk space, prerequisite software, and so on, for a successful deployment.

The SDD is significant because it bridges the gap between software providers and deployers, through standardized, externalized metadata. The SDD is an important

and necessary element of an interoperable software lifecycle management system.

This introductory article offers an overview of the concepts and structure of the SDD. There is much more to say about SDD. Future articles will continue to explore this important emerging standard.

## Resources

- [Participate in the discussion forum for this content.](#)
- Find everything about SDD at the [OASIS SDD public page](#).
- This [index](#) provides the most current SDD specification as well as the SDD starter profile, SDD examples, and the SDD primer.
- Learn about [CIM](#), the basis for the SDD Starter Profile.
- See the [IUDD submission](#) that served as a starting point for SDD.
- Find out more about the [Eclipse COSMOS open source project](#).

## About the authors

### Julia McCarthy

Julia McCarthy, SWG AIM Install Strategy and Development, was editor of the Solution Deployment Descriptor (SDD) 1.0 specification. As the secretary of the OASIS SDD technical committee, she is currently focusing on facilitating adoption of SDD within IBM. She has worked for IBM for 20 years, with assignments ranging from strategy to tools development.

---

### Brent A. Miller

Brent A. Miller, Senior Technical Staff Member, Autonomic Computing Architecture. Brent is Chief Architect for IBM's Autonomic Computing team. He has worked for IBM for 24 years, with assignments including printer development, mobile clients, mobile software and pervasive computing. He currently chairs the OASIS SDD technical committee.

## Trademarks

The, IBM, the IBM logo, Tivoli, DB2, Rational and WebSphere are trademarks of International Business Machines Corporation in the United States, other countries or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems,

Inc. in the United States, other countries or both.