This is a Non-Standards Track Work Product.
The patent provisions of the OASIS IPR Policy do not apply.

Solution Deployment Descriptor (SDD) V2.0 Primer Version 1.0
Committee Note Draft 01
16 May 2011

Work Product URIs:
This version:
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/cnd01/sdd-primer-v1.0-cnd01.doc (Authoritative)
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/cnd01/sdd-primer-v1.0-cnd01.html
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/cnd01/sdd-primer-v1.0-cnd01.pdf
Previous version:
http://docs.oasis-open.org/sdd/v1.0/cd01/sdd-primer-v1.0-cd01.doc (Authoritative)
http://docs.oasis-open.org/sdd/v1.0/cd01/sdd-primer-v1.0-cd01.html
http://docs.oasis-open.org/sdd/v1.0/cd01/sdd-primer-v1.0-cd01.pdf
Latest version:
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/sdd-primer-v1.0.doc (Authoritative)
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/sdd-primer-v1.0.html
http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/sdd-primer-v1.0.pdf

Technical Committee:
OASIS Solution Deployment Descriptor (SDD) TC

Chair:
Brent A. Miller, IBM Corp.

Editor:
Brent A. Miller, IBM Corp.

Related work:
This non-standards track work product replaces or supersedes:

- Solution Deployment Descriptor (SDD) Primer Version 1.0

This non-standards track work product is related to:

- [SDD] OASIS, Solution Deployment Descriptor Specification Version 2.0,
  http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-v2.0.html
Abstract:
This non-standards track work product provides non-normative information to supplement the Solution Deployment Descriptor (SDD) specification and serves as a “getting started” guide.

Status:
This document was last revised or approved by the OASIS Solution Deployment Descriptor (SDD) TC on the above date. The level of approval is also listed above. Check the “Latest version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/sdd/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/sdd/ipr.php.)

Citation format:
When referencing this specification the following citation format should be used:

[SDD-Primer]  
Solution Deployment Descriptor (SDD) V2.0 Primer Version 1.0. 16 May 2011. OASIS Committee Note Draft 01. http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-primer/v1.0/cnd01/sdd-primer-v1.0-cnd01.html.

Copyright © OASIS 2011. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the “OASIS IPR Policy”). The full Policy may be found at the OASIS website.

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 section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, 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 OASIS or its successors or assigns.
# Table of Contents

1 Introduction ............................................................................................................. 4  
1.1 Terminology ......................................................................................................... 4  
1.2 General Document Conventions ............................................................................ 4  
2 Why Use SDD? ........................................................................................................ 5  
3 Getting Started ....................................................................................................... 7  
4 Examples .................................................................................................................. 9  
4.1 General Deployment Model .................................................................................. 9  
4.2 Simple JRE Example ............................................................................................ 10  
4.2.1 PackageDescriptor ......................................................................................... 10  
4.2.2 DeploymentDescriptor .................................................................................... 11  
4.3 Composite Application Example ......................................................................... 14  
4.3.1 Referenced PackageDescriptors and DeploymentDescriptors ....................... 15  
4.3.2 Composite PackageDescriptor ....................................................................... 24  
4.3.3 Composite DeploymentDescriptor .................................................................. 25  
5 Additional Considerations ....................................................................................... 30  
6 Conclusion ............................................................................................................... 32  
   A. Complete Examples ......................................................................................... 33  
   B. Acknowledgements ......................................................................................... 34  
   C. Revision History .............................................................................................. 35
1 Introduction

This is a non-normative, expository, non-standards track work product that supplements the Solution Deployment Descriptor (SDD) specification. This document provides a quick way to get started with the pragmatics of the SDD, but it does not replace the specification or schema, which need to be understood.

This Primer describes why and how to use SDD, making use of examples produced by the OASIS SDD technical committee. This version concentrates on the use of SDD for installation; in the future, new versions or additional modules may be created to describe other uses of SDD, such as configuration and localization.

1.1 Terminology

Terminology in this document is consistent with the SDD Specification [SDDSP].

1.2 General Document Conventions

This document contains cross-references. Such references appear as the referenced section number inside square brackets, for example, [4.5]. In electronic versions of this specification, the cross-references can act as links to the target section.

Within the XML snippets (excerpts from SDD examples), schema elements and attributes that serve as ids and references are highlighted in blue underscored text. Schema element and attribute values that are expected to be found in SDD profiles are identified with red italic text in the XML snippets.
2 Why Use SDD?

Solution Deployment Descriptor [SDD] provides a standardized way to declare and externalize information about your software package and its deployment, as illustrated in Figure 1, with the explanation that follows.

Basic Information in a Solution Deployment Descriptor

Package Identity
What is this thing?
What does it contain?

Requirements
What is needed to deploy this package?
What must be maintained for the lifetime of the deployment?

Package Variability
What parts need to be deployed?

Software Package
(logical, not physical)

Results
What does this provide?
What effect will this have on my environment?

Figure 1: Overview of information provided in an SDD

Consider some sort of software package as illustrated by the box in Figure 1. To successfully deploy this software, information about it is required. This information is analogous to what might be provided on the box for software purchased on a CD or DVD—it tells you what you need to know about this software. The SDD provides a way to describe such software in a standardized form that can be electronically recorded and programmatically processed, in addition to providing information useful to humans.

- **Package identity**: this portion of the SDD describes the name and source of the software and its logical package structure, including the content (executable files, license agreements, documentation and so on) that makes up the software.
- **Requirements**: this portion of the SDD describes what is necessary for the software to be successfully deployed, including requirements for disk space, CPU capacity, and other declarations about the required state of the deployment environment (for example, other prerequisite software, configuration settings and so on).
- **Package variability**: this portion of the SDD describes options for deployment. Some parts of the software might not be used in every deployment. For example, the software package might contain software variations for two different operating systems; the condition of the deployment environment (that is, the actual operating system type) determines which of the variations is used in a particular deployment. The software package also might contain optional
features that the installer chooses at installation time. Also, different languages might be deployed for different installations.

- **Results:** this portion of the SDD describes what will happen once the software is deployed (for example, installation of new software, update of existing software, configuration or localization of existing software, and so on) and what effects it will have on the environment once it is deployed (for example, new applications are created or existing applications are updated).

All of this information enables deployers to analyze and make pre-deployment decisions. SDD producers can be developers, aggregators, service and maintenance support staff and others. SDD consumers can include humans and tools that perform composition, perform pre-deployment planning, make pre-deployment decisions and/or perform deployment operations.

The SDD can be used across a spectrum from simple packages to complex solutions. It enables lifecycle management of software (installation, configuration, localization, fix application, update/upgrade and uninstallation) to be more (if not fully) automated.

The standard representation provided by SDD enables solutions to be easily composed from existing components. SDD does not require that you discard, replace or rewrite all of your installers; SDD provides declarative metadata for software that can continue to be installed by existing deployment software.
3 Getting Started

SDD is represented as an XML schema [SDD-Schema]. The SDD specification [SDD] provides semantics and other normative and non-normative information.

An SDD consists of two major parts: the Package Descriptor and the Deployment Descriptor. As noted in the specification:

The package descriptor defines package content which includes artifacts whose processing results in deployment of the software package. The deployment descriptor defines metadata associated with those artifacts. The SDD package descriptor defines the package identity, the package content, and various other attributes of the package. Each SDD consists of exactly one deployment descriptor and one package descriptor. The deployment descriptor is where the topology, selectability, inputs, requirements, and conditions of the deployment are described.

You describe your software by declaring things about it using XML. Referring back to Figure 1, the information you can declare for each part illustrated there includes:

- **Package Identity** is part of the PackageDescriptor, and includes primarily the PackageIdentity and Contents elements. PackageIdentity contains other elements and attributes that name and describe the software package (including items such as version, human-readable descriptions and other identifying information). Contents includes the “files” that make up the package, including their purpose, where they can be found and other information, such as optional digital signatures.

- **Requirements** are contained in the DeploymentDescriptor, within the various types of content elements (InstallableUnit, ConfigurationUnit, LocalizationUnit, CompositeUnit and CompositeInstallable). The Requirements element describes resource constraints and dependencies for internal content elements that must be met, including versions, relationships, property values, capacity constraints and consumption constraints. Depending on the type of content element, the presence of required base software might also be specified. The Requisites element allows the SDD to incorporate other software that can help to satisfy some of these requirements if the deployment environment doesn’t already satisfy them.

- **Package variability** can be accomplished within the DeploymentDescriptor with the SelectableContent element that enables Features and Groups to be selected, as well as with the Condition construct that can be applied to multiple elements in the SDD so that certain items can be “conditioned” in or out of scope for a particular deployment.

- **Results** are contained in the DeploymentDescriptor, within the various types of content elements. Depending on the type of content element, the ResultingResource or ResultingChange elements describe the results of the deployment (what happens to the deployment environment as a result of performing the deployment described by the SDD).

In addition to the concepts illustrated in Figure 1, other items can be described in the SDD:

- **Topology**: The logical topology of the solution can be expressed in an SDD. The Topology element describes all of the resources relevant for deployment, including resources that are required, created or modified during deployment, as well as the relationships among these resources.

- **Artifacts**: These content files accomplish the deployment operations. Artifact files (for example, ZIP files, RPM files or executable installation files) are processed during deployment to install,
configure, localize or perform other deployment operations. The Artifact element describes artifacts, along with the inputs and outputs, including substitution values, used when processing those artifacts.

- **Atomic & composite content units**: Atomic content units define artifacts (just described); the three atomic content unit elements are InstallableUnit, ConfigurationUnit and LocalizationUnit (this version of the Primer focuses primarily on InstallableUnits). Atomic units suffice for many simple deployments; when more complex solutions are deployed (for example, when multiple SDDs are aggregated or when package variability exists in the form of selectable content), composite content units are used. The three types of composite content units are CompositeInstallable, CompositeUnit and CompositeLocalizationUnit. This version of the Primer focuses primarily on CompositeInstallables.

- **Variables and parameters**: Variables provide a way to obtain and derive values from resource properties and the deployment environment and human deployers. The variables can then be used in various portions of the SDD to influence the deployment process, including the use of variables as input arguments to artifacts, and values for resource constraints.

- **Operations**: Operations serve as the “verbs” for deployment described by an SDD. Operations describe the deployment steps that are performed; some of the operations are create, update and uninstall. This version of the Primer focuses primarily on the create operation that is associated with installing software.

- **Display information**: Although the SDD enables programmatic processing and automated deployment, it is also important to include descriptive information that humans can use. Many deployment operations are interactive, requiring humans to make selections, confirm operations, provide input, and so on. Many elements throughout the SDD support display information to provide human-understandable descriptions; these display elements are translatable to support multiple languages.

The remainder of this document uses examples to illustrate each of these aspects of the SDD and addresses additional considerations for producing and consuming useful SDDs. This version of the Primer focuses on concepts and practices for SDDs that perform installation; future publications may expand this focus with examples and considerations for other SDD operations such as configuration and localization (these latter aspects are introduced but addressed in less detail than installation).
4 Examples

To further illustrate how the SDD can be used, we refer to the examples published by the SDD TC [SDDEX] and describe those examples, along with information pertinent for SDD authors and consumers, in the following sections.

4.1 General Deployment Model

Before examining the details of the examples, it is useful to understand the general deployment model that underlies SDD; we call this the installable unit/hosting environment design pattern. This model is illustrated in Figure 2 and described next.

**Installable Unit / Hosting Environment Design Pattern**

![Diagram](image)

**Figure 2: Installable Unit/Hosting Environment Design Pattern**

In general, things that are deployed can fit into the installable unit/hosting environment design pattern. That is, as shown in the left side of Figure 2, an installable unit containing one or more artifacts can be installed into a hosting environment. The figure also shows the other major portions of an SDD, namely a descriptor for the installable unit and a package with the associated content.

As shown on the right side of Figure 2, this design pattern can be used at all levels of the resource stack. For example, an operating system installable unit can be installed into a hardware hosting environment; a software product (such as a native application) can be installed into an operating system hosting environment, and so on.

This design pattern makes it possible to standardize many aspects of software that coordinates deployment of diverse resource types, and it underlies the standardization of the declarative
metadata for the installable unit, including artifacts, package descriptors and deployment descriptors, embodied in the SDD standard.

Understanding this design pattern facilitates understanding the examples, which are explained next.

4.2 Simple JRE Example

This example is a simple one that illustrates a basic PackageDescriptor and corresponding DeploymentDescriptor for a Java™ runtime environment. For the complete example, refer to the files examples/CompositeApp/pkgs/JRE/jre.xml and examples/CompositeApp/pkgs/JRE/jre_pkg.xml in [SDDEX] (see Appendix [A]).

4.2.1 PackageDescriptor

- This simple PackageDescriptor contains Identity information, including Description, ShortDescription, Version, BuildInformation and Manufacturer information. All of this information is provided by the package author. The Identity information excerpt from this example follows.

```xml
<sdd-pd:PackageDescriptor
    [...]
    descriptorLanguageBundle="com.oracle.java.jre"
    <sdd-common:Description translationKey="DESC">Lets you develop and deploy Java(TM) applications on desktops and servers, as well as today's Embedded and Realtime environments. Java SE includes classes that support the development of Java Web Services and provides the foundation for Java Platform, Enterprise Edition (Java EE).</sdd-common:Description>
    <sdd-common:ShortDescription translationKey="SHORT_DESC">Java(TM) Platform for software development</sdd-common:ShortDescription>
    <sdd-common:Name translationKey="PKG_DISPNAME">Java(TM) Platform, Standard Edition Runtime Environment</sdd-common:Name>
    <sdd-common:Version>1.5.0</sdd-common:Version>
    <sdd-common:BuildInformation buildID="09122006-193602"/>
    <sdd-common:Manufacturer>
        <sdd-common:Name>Oracle Corp.</sdd-common:Name>
        <sdd-common:Location>
            <sdd-common:Address>500 Oracle Parkway, Redwood Shores, CA 94065</sdd-common:Address>
            <sdd-common:Country>U.S.A.</sdd-common:Country>
        </sdd-common:Location>
    </sdd-common:Manufacturer>
    <sdd-common:ContactInformation>https://support.oracle.com/CSP/ui/flash.html</sdd-common:ContactInformation>
</sdd-pd:PackageIdentity>
```

- The descriptorLanguageBundle attribute specifies that language bundle files that contain translations for display text elements in this PackageDescriptor (these display text elements include Description, ShortDescription and Name) are contained within this PackageDescriptor and are identified by the root file name “com.oracle.java.jre”. The package Contents, described next, illustrate one languageBundle file that corresponds to this descriptorLanguageBundle value.

---

1 Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries.
The Contents in this package consist of the artifact, with the assigned id “JRE_RPM”; a language bundled with the assigned id “EN_Bundle” and the corresponding DeploymentDescriptor (described next), with the assigned id “DD”. These ids are chosen by the package author and are used to reference these content files from the deployment descriptor. These and other ids are highlighted in blue underscored text in the following excerpt; their use as references is described in the next section.

<table>
<thead>
<tr>
<th><a href="">sdd-pd:Contents</a></th>
</tr>
</thead>
</table>
| |<sdd-pd:Content pathname="jre-1_5_0_09-AIX-rpm.bin" id="JRE_RPM" purpose="content" />
| |<sdd-pd:Content pathname="jre_update.xml" id="DD" purpose="deploymentDescriptor" />
| |<sdd-pd:Content pathname="com/oracle/java/jre_en.properties" id="EN_Bundle" purpose="languageBundle" /> |
| </sdd-pd:Contents> |

4.2.2 DeploymentDescriptor

This simple DeploymentDescriptor is used with the preceding PackageDescriptor. It contains information relevant to deploying the JRE that is described in that package descriptor.

- This simple DeploymentDescriptor illustrates the declaration of Topology and Identity for this deployment, as well as the use of Variables during deployment operations (in this case, install) and the declaration of Requirements (in this case, ResourceConstraints) necessary for deployment. It also identifies the Artifact used during deployment.
- Several values in this DeploymentDescriptor are taken from a profile. These values are identified with red italic text in the example excerpts included here. See [SDDSP] for the Starter Profile that illustrates the definition of values such as these.

In the Topology section, the SDD author declares a resource of the type CIM_OperatingSystem; this resource serves as a hosting environment (in the installable unit/hosting environment design pattern), and its type comes from a profile. This operating system resource hosts a resource of the type CIM_FileSystem; this file system serves as the actual hosting environment for the JRE that is deployed with this SDD, and its type also comes from a profile. So, this SDD is declaring that the JRE, when deployed, will be hosted by the file system (that in turn is hosted by an operating system). Moreover, it is known that this JRE is destined for an AIX® operating system, so a relevant property called Root, with the value /usr, is declared; this property declares the file system where the JRE will be installed. These values also come from a profile that defines relevant properties for the AIX operating system. Finally, the resource to be deployed (the JRE) is itself declared, with the type CIM_InstalledProduct, and that value also comes from a profile. Note that each of these resources has an id (os, UsrFilesys and JRE, respectively) that is used to refer to these resources elsewhere in the DeploymentDescriptor. The os and UsrFilesys resources need to be defined because each has an associated Requirement defined later in the DeploymentDescriptor; the JRE resource needs to be defined because it is later referred to as a resulting resource. These Topology elements are shown in the following example excerpt.

<table>
<thead>
<tr>
<th><a href="">sdd-dd:Topology</a></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>&lt;/sdd-dd:Topology&gt;</td>
</tr>
</tbody>
</table>

---

2 AIX is a registered trademark of IBM in the United States, other countries, or both

3 In fact, the os and UsrFilesys resources would need to be included in Topology even if Requirements were not defined for them, because of their relationships to another resource (in this case, the JRE) that is referred to elsewhere.
Because this DeploymentDescriptor describes the installation of the JRE, it declares an InstallableUnit whose targetResourceRef is the operating system that was previously declared with the id "os". This information tells the SDD consumer that the AIX operating system is capable of processing the JRE artifact file. In this example, the target resource is the operating system, whereas the hosting environment resource is the file system. In many cases, the hosting environment also serves as the target resource. These elements appear in the following excerpt.

```
<installableUnit id="ID000026" targetResourceRef="os">
  <installableUnit id="2000-123">
    <resource id="usr"/>
    <resource id="/usr">
      <property name="Root" value="/usr"/>
    </resource>
    <resource id="/usr">
      <property name="os" value="AIX"/>
    </resource>
  </installableUnit>
  <installableUnit id="JRE">
    <resource id="sp:CIM_InstalledProduct"/>
  </installableUnit>
</installableUnit>
```

This simple DeploymentDescriptor has a single InstallableUnit. That InstallableUnit’s Identity section happens to match the Identity information in the corresponding PackageDescriptor. However, the identity of the overall package (in the PackageDescriptor) and the identity of the content units (such as the InstallableUnit in this example) are distinct entities. In this case, the SDD author has determined that it is convenient for the InstallableUnit Identity and the overall package Identity to match; this enables identity information to be obtained from the DeploymentDescriptor without opening the PackageDescriptor. Note that these distinct identities need not (and often do not) match; see, for example, the composite application example in section [4.3], which has multiple InstallableUnits. The complete example contains this Identity information, which is similar to the Identity information in the PackageDescriptor.

This DeploymentDescriptor defines a Variable of type StringParameter that represents the logging level to be used when the JRE is deployed. Valid and default values for this parameter (defined in a profile) are defined. This parameter’s value will be set using a substitution (illustrated and described later), and the resulting value will be passed to the artifact when the artifact is invoked for processing. This Variable is shown in the following example excerpt.

```
<variables>
  <parameters>
    <stringParameter sensitive="false" name="LoggingLevel" default="INFO"/>
    <description>Default logging level for logging messages coming from JRE</description>
  </parameters>
  <validValues>
    <validValue>FINEST</validValue>
    <validValue>FINER</validValue>
    <validValue>FINE</validValue>
    <validValue>CONFIG</validValue>
    <validValue>SEVERE</validValue>
    <validValue>INFO</validValue>
    <validValue>WARNING</validValue>
  </validValues>
</variables>
```

This DeploymentDescriptor defines Requirements that must be met for a successful deployment. Note that the id values for the Requirements must be unique within this deployment descriptor.
and also must be unique across multiple deployment descriptors that are aggregated together. This is illustrated further in the Composite Application example later in this document.

- The first Requirement declares that the operating system type must be AIX and that the minimum version of the operating system is 5.1. This Requirement further declares that the "certified", or fully tested, versions for this JRE span AIX version 5.1 to 5.3. This expresses the SDD author's intent that the operating system type must be AIX and must be at least version 5.1; preferred versions are 5.1 to 5.3, so versions above 5.3 might be acceptable but probably should be tested or otherwise verified for this deployment. Note that some values for this Requirement come from a profile.

- The second Requirement declares the amount of free disk space that must be available for a successful deployment. In this case, 2688 512-byte blocks (the unit of measure used for disk space on this AIX operating system) must be available. Note that some values for this Requirement come from a profile.

These Requirements appear in the following example excerpt.

```
<sdd-dd:Requirements>
  <sdd-dd:Requirement id="OSLevel" operation="install use">
    <sdd-dd:Description>This JRE requires an AIX Operating System at a version of at least 5.1. It has been certified for all versions of AIX between 5.1 and 5.3.</sdd-dd:Description>
    <sdd-dd:ResourceConstraint id="osMinimumVersionRequirement" resourceRef="os">
      <sdd-dd:Description>This JRE requires a version of AIX or 5.1 or later. It has been certified on versions of AIX between 5.1 and 5.3.</sdd-dd:Description>
      <sdd-dd:PropertyConstraint>
        <sdd-dd:PropertyName>sp:CIM_OperatingSystem.OSType</sdd-dd:PropertyName>
        <sdd-dd:Value>AIX</sdd-dd:Value>
      </sdd-dd:PropertyConstraint>
      <sdd-dd:VersionConstraint>
        <sdd-dd:Supported>
          <sdd-dd:MinVersion>5.1</sdd-dd:MinVersion>
          <sdd-dd:MaxVersion inclusive="true">5.3</sdd-dd:MaxVersion>
        </sdd-dd:Supported>
      </sdd-dd:VersionConstraint>
    </sdd-dd:ResourceConstraint>
  </sdd-dd:Requirement>
  <sdd-dd:Requirement id="UsrDiskSpace" operation="install use">
    <sdd-dd:Description>This JRE requires 2688 512-byte blocks of available space on the /usr file system.</sdd-dd:Description>
    <sdd-dd:ResourceConstraint id="UsrDiskSpaceRequirement" resourceRef="UsrFilesys">
      <sdd-dd:ConsumptionConstraint>
        <sdd-dd:PropertyName>sp:CIM_FileSystem.AvailableSpace</sdd-dd:PropertyName>
        <sdd-dd:Value unit="512-blocks">2688</sdd-dd:Value>
      </sdd-dd:ConsumptionConstraint>
    </sdd-dd:ResourceConstraint>
  </sdd-dd:Requirement>
</sdd-dd:Requirements>
```
• The DeploymentDescriptor declares a ResultingResource that will result from successful deployment. In this case, a new resource, an instance of the JRE version 1.5.0, will be created in the deployment environment. This information is shown in the following excerpt.

```
<sdd-dd:ResultingResource resourceRef="JRE">
  <sdd-dd:Description> An instance of Java(TM) Runtime Environment, Standard Edition Version 5.0 is installed as a result of this deployment</sdd-dd:Description>
  <sdd-dd:Name>Java(TM) Runtime Environment, Standard Edition</sdd-dd:Name>
  <sdd-dd:Version>1.5.0</sdd-dd:Version>
</sdd-dd:ResultingResource>
```

• Finally, the DeploymentDescriptor declares the Artifact that is processed to accomplish the installation.

• This InstallArtifact element defines metadata about the artifact file (“JRE_RPM”, as identified in the contentRef attribute) that is used to create the JRE. This artifact is an RPM file (a typical type used by installers in the AIX operating system), as identified by the type attribute.

• The argument value for the logging level in this example comes from a file, and the value in the file requires a substitution before it can be used for artifact processing. The contentRef attribute specifies the file in which the substitution will take place (“LoggingProperties”). For the substitution, the required attribute indicates that the resulting value must be valid, and the limit attribute specifies that only one substitution should occur (even if the Pattern occurs more than once in the file). Pattern specifies the string in the file that will be replaced; Value specifies the variable expression that is evaluated to determine the text that will replace (be literally substituted for) Pattern in the file. In this example, “${LoggingLevel}” indicates that the value of the LoggingLevel variable (previously defined in the SDD and described earlier) is evaluated, and the result is substituted for the string “INFO” in the file. The file (with the substituted value) is passed with the artifact to the target resource for use when processing the artifact. The earlier declaration of the parameter enables checking the value that is set against valid values (and also defines a default value in case no value is set during the installation process). This information appears in the following example excerpt.

```
<sdd-dd:Artifacts>
  <sdd-dd:InstallArtifact type="rpm" contentRef="JRE_RPM">
    <sdd-dd:AdditionalContent type="text" contentRef="LoggingProperties">
      <sdd-dd:Substitution required="true" limit="1">
        <sdd-dd:Pattern>INFO</sdd-dd:Pattern>
        <sdd-dd:Value>${LoggingLevel}</sdd-dd:Value>
      </sdd-dd:Substitution>
    </sdd-dd:AdditionalContent>
  </sdd-dd:InstallArtifact>
</sdd-dd:Artifacts>
```

4.3 Composite Application Example

In this section, a more complex example is provided. It is an aggregation of a set of other SDDs for the deployment of the following packages:

• a 3-tier J2EE application with user interface, backend business logic and a database connection;

• an optional J2EE simple client (this client requires a JRE runtime with a minimum
version);

- a JRE runtime of a version that satisfies the J2EE simple client requirement; and
- some optional German and French language packs for the J2EE simple client.

These constituent SDDs are referred to as referenced packages in the aggregating SDD. Note that the individual constituent packages themselves are self-sufficient; that is, they are complete SDD packages and may be used to deploy their respective content independently, in addition to being used to deploy the aggregated solution that is described in this section.

A composite package descriptor-deployment descriptor pair combines these constituent SDDs into a composite application SDD.

For the complete set of example files described throughout this section, refer to the examples/CompositeApp folder in [SDDEX] (see Appendix [A]).

### 4.3.1 Referenced PackageDescriptors and DeploymentDescriptors

The complete solution consists of several SDDs. The Composite PackageDescriptor and DeploymentDescriptor sections describe the components that constitute the aggregated SDD. In this section, more details are provided for the individual referenced packages (the ones that are aggregated by the composite SDD), including:

- the 3-tier J2EE application package;
- the Simple J2EE client;
- the JRE runtime package; and
- the German and French language packs.

#### 4.3.1.1 J2EE Application Package

The J2EE application package (SC_pkg) has a PackageDescriptor located in pkgs\Composite\SimpleCompositeApp_pkg.xml and a DeploymentDescriptor located in pkgs\Composite\SimpleCompositeApp.xml in [SDDEX] (see Appendix [A]).

- The PackageDescriptor for the 3-tier J2EE application simply lists the identity of the package and the contents contained in the package. Note that the contents listed in the Content section are of different types; hence, their deployment processing will differ as described in the corresponding DeploymentDescriptor (for example, the J2EE client content has a different file type and format than the database content, so their respective installable unit processing varies).

- The DeploymentDescriptor first describes a Topology with nested resources for the components to be installed. There are three top-level resources described with Resource elements. The first is J2eeServletServer, of type CIM_J2eeServer, hosting hosted resource SimpleJ2eeServlet, of type CIM_J2eeServlet. The second top-level resource is appServer_os, of type CIM_OperatingSystem, and it hosts resource J2eeAppServer, of type CIM_J2eeServer, which in turn hosts resource SimpleJ2eeApp, of type CIM_J2eeApplication. The third top-level resource is os, of type CIM_OperatingSystem, and it hosts resource DatabaseServer, of type CIM_DatabaseServer, which in turn hosts resource SimpleDatabase, of type CIM_CommonDatabase. Note that declaring appServer_os and os as separate resources does not imply that they are separate operating systems (or separate hardware hosts), but rather allows for distributed deployment—for example, J2eeAppServer and DatabaseServer could reside on separate operating system instances. If, instead, J2eeAppServer and DatabaseServer were both hosted by the same hosting resource of type CIM_OperatingSystem, then it would be implied that they both reside on the
same operating system instance (and hardware host). As shown later in the deployment descriptor, the J2eeServletServer, J2eeAppServer and DatabaseServer are used in the Requirements sections to declare requirements for the deployment. The SimpleJ2eeServlet, SimpleJ2eeApp and SimpleDatabase are used to declare the resulting resources for this SDD, and the J2eeAppServer, SimpleJ2eeApp, DatabaseServer and SimpleDatabase are used to declare the resource relationships.

- The DeploymentDescriptor defines the InstallableUnits associated with this component of the composite application and specifies their different deployment characteristics based on Requirements. The InstallableUnits are combined in CompositeInstallable. Note the Parameters element within the Variables element: it declares two input parameters that apply to the entire CompositeInstallable: JDBC_User and JDBC_Password.
Each individual constituent InstallableUnit follows the pattern already described in the Simple JRE example. For example, there is an InstallableUnit called SimpleJ2EEServletUnit. This component is packaged into a Web archive (.war) format. The Requirements element in the InstallableUnit defines the pre-requisite for this component by using the ResourceConstraint and RelationshipConstraint. The ResourceConstraint in the example specifies that the servlet package must be deployed with an Apache Tomcat server pre-installed and the version of Apache Tomcat must be 5.5.20. The requirement section also specifies a RelationshipConstraint that specifies that there must be a J2EE application server running the RMI-IIOP protocol. Once the requirement for the InstallableUnit is met, the files specified in the InstallArtifact element can be deployed. In this case, the war file as specified in the SC_WAR reference is deployed as the Servlet unit. This InstallableUnit and its Requirements are shown in the following example excerpt.

An additional concept not illustrated in the Simple JRE example is shown here: the InstallArtifact has a weight attribute. One of the objectives of the SDD is to enable information to be included that can be used by runtimes to monitor and display the progress of deployment operations. Mechanisms for progress tracking could vary widely and results in various environments also are likely to differ. In addition, including precise information about deployment times in an SDD is impractical, especially because multiple packages from multiple sources can be aggregated and the processing of an SDD can vary from deployment to deployment (based on conditions, features, characteristics of the deployment environment and so on). Nevertheless, the SDD specification does include weight attributes for Artifacts and ReferencedPackages; these weights are relative times that SDD authors can specify and SDD consumers can use for estimation and coarse deployment progress expectations and tracking.

The key to the use of weight in the SDD is to understand that the weight for a particular Artifact or ReferencedPackage is an estimate of the time required to process that artifact or package relative to other artifacts and packages within that SDD. Weights are not absolute, they are not percentages and they do not apply across SDDs, only within a single SDD. In this example, the SimpleJ2eeServletUnit’s Artifact has a weight value of 2, as shown in the following example excerpt. In the complete SimpleCompositeApp example, the other InstallableUnits also have weight values: the SimpleJ2eeAppUnit’s Artifact has a weight value of 3; the OracleSimpleDatabaseUnit’s Artifact has a weight value of 4; and the DB2SimpleDatabaseUnit’s Artifact has a weight value of 4.

So, in this example, the expectation is that the SimpleJ2eeAppUnit Artifact (weight=3) will take approximately 1.5 times as long to process as the SimpleJ2eeServletUnit Artifact (weight=2); and either database InstallableUnit Artifact (both weight=4) will take approximately twice as long to process as the SimpleJ2eeServletUnit Artifact (weight=2).

(Similarly, either database InstallableUnit Artifact (both weight=4) is expected to take approximately 1.3 times as long to process as the SimpleJ2eeAppUnit Artifact (weight=3)). The weight values have no meaning in isolation and do not describe actual time elapsed. They simply provide an estimate, or hint, of relative time.

```
<sdd-dd:InstallableUnit id="SimpleJ2eeServletUnit"
targetResourceRef="J2eeServletServer">
  <sdd-dd:Identity>
    <sdd-dd:Description>The user interface for the Simple Composite Application.</sdd-dd:Description>
    <sdd-dd:Name>Simple Application Servlets</sdd-dd:Name>
    <sdd-dd:Version>1.0</sdd-dd:Version>
  </sdd-dd:Identity>
  <sdd-dd:Requirements>
```
J2EE application) that consists of software and/or its affiliates. 

DB2 is a registered trademark of IBM Corporation in the United States.

Other InstallableUnits in the J2EE Application package follow patterns similar to the preceding example.

Note that there are two installable units for database installation that have the same resulting resource reference: SimpleDatabase. These two installable units, however, have different conditions—the first requires database server Oracle® and the second requires database server DB2®. Therefore, only one of these two units will be processed during a particular deployment, resulting in only one SimpleDatabase resulting resource.

The J2EE application also illustrates the Condition construct, which is useful to enable flexible processing within an SDD by declaring which aspects of an SDD are applicable (or can be ignored) in certain circumstances. Conditions can be applied to Content (to determine if a content element is applicable), Variables (to choose values), Features (to determine when a feature is applicable), ResultingResources (to determine when a particular result is applicable) and CompletionActions (to determine if a particular completion action is necessary). The J2EE example illustrates one use of the Condition element to provide content for different deployment environments.

In this example, the SDD describes some content (a CompositeInstallableUnit for the overall J2EE application) that consists of software that is common to all deployments, plus some
content that varies. The variable content is the database for the 3-tier application, and it is
based on a Condition. The database for the 3-tier application can be either an Oracle 9iAS
database or an IBM® DB2 9 database. This DeploymentDescriptor describes an
InstallableUnit that consists of an Oracle database. The Condition checks DatabaseServer for
the value “Oracle9iAS”, and if the condition is met, the SDD specifies additional information
that applies for the Oracle database (in this case, Oracle database-specific Requirements
in addition to those specified for the common content, as well as the particular Oracle database
ResultingResource and Artifact). The SDD then describes another InstallableUnit that
consists of a DB2 database. The Condition checks DatabaseServer for the value “DB2 9”,
and if the condition is met, the SDD specifies additional information that applies for the DB2
database (in this case, DB2 database-specific Requirements in addition to those specified for
the common content, as well as the particular DB2 database ResultingResource and Artifact).
These Conditions are shown in the following example excerpt.

```
<sdd-dd:InstallableUnit id="OracleSimpleDatabaseUnit"
targetResourceRef="DatabaseServer">
  <sdd-dd:Identity>
    <sdd-common:Description>The RDBMS Database structure for the Simple
    Composite Application.</sdd-common:Description>
    <sdd-common:Name>Simple Application Database</sdd-common:Name>
    <sdd-common:Version>1.0</sdd-common:Version>
  </sdd-dd:Identity>
  <sdd-dd:Condition>
    <sdd-dd:ResourceConstraint id="OracleDatabaseServer.condition"
    resourceRef="DatabaseServer">
      <sdd-dd:Name>Oracle 9iAS</sdd-dd:Name>
    </sdd-dd:ResourceConstraint>
  </sdd-dd:Condition>

  [......] [Oracle database-specific Requirements, ResultingResource, Artifact and
          InstallableUnit]
</sdd-dd:InstallableUnit>

<sdd-dd:InstallableUnit id="DB2SimpleDatabaseUnit"
targetResourceRef="DatabaseServer">
  <sdd-dd:Identity>
    <sdd-common:Description>The RDBMS Database structure for the Simple
    Composite Application.</sdd-common:Description>
    <sdd-common:Name>Simple Application Database</sdd-common:Name>
    <sdd-common:Version>1.0</sdd-common:Version>
  </sdd-dd:Identity>
  <sdd-dd:Condition>
    <sdd-dd:ResourceConstraint id="DB2DatabaseServer.condition"
    resourceRef="DatabaseServer">
      <sdd-dd:Name>DB2 9</sdd-dd:Name>
    </sdd-dd:ResourceConstraint>
  </sdd-dd:Condition>

  [......] [DB2 database-specific Requirements, ResultingResource, Artifact and
          InstallableUnit]
</sdd-dd:InstallableUnit>
```

This example illustrates conditional Content; Conditions also can be applied to other
elements within the SDD (Variables, Features, ResultingResources and CompletionActions).

---

6 IBM is a registered trademark of IBM Corporation in the United States.
4.3.1.2 J2EE Client

The second referenced package is a simple client application with PackageDescriptor located in pkgs\Client\SimpleAppClient_pkg.xml and DeploymentDescriptor located in pkgs\Client\SimpleAppClient.xml in [SDDEX] (see Appendix [A]).

The PackageDescriptor for the application client is relatively simple and similar to previously described PackageDescriptors. This PackageDescriptor contains a jar file for the installer and a separate one for the uninstaller.

The DeploymentDescriptor contains several key elements that are critical for deploying this simple client.

- The first is the Topology section. Three top-level resources are described with Resource elements. The first is JRE, of type sp:CIM_InstalledProduct. The second is Filesys, of type sp:CIM_FileSystem and it hosts resource InstallDir, of type sp:CIM_Directory; the name of this directory is the value of the InstallLocation variable. The third top-level resource is SimpleAppClient, of type sp:CIM_Application. As shown later in the deployment descriptor, the JRE and Filesys are used in the Requirements section to declare the requirements for the deployment. The SimpleAppClient and InstallDir are used to declare the resulting resources for this SDD.

```xml
<sdd-dd:Topology>
  <sdd-dd:Resource id="os" type="sp:CIM_OperatingSystem">
    <sdd-dd:HostedResource id="JRE" type="sp:CIM_InstalledProduct" />
    <sdd-dd:HostedResource id="Filesys" type="sp:CIM_FileSystem">
      <sdd-dd:HostedResource id="InstallDir" type="sp:CIM_Directory">
        <sdd-common:Name>${InstallLocation}</sdd-common:Name>
      </sdd-dd:HostedResource>
    </sdd-dd:HostedResource>
    <sdd-dd:HostedResource id="SimpleAppClient" type="sp:CIM_Application" />
  </sdd-dd:Resource>
</sdd-dd:Topology>
```

- In the Requirements section, there are three requirements that must be met to deploy this package. These requirements include an operating system requirement, a Java Runtime requirement and a disk space requirement. The operating system requirement uses an Alternative element to specify that the package can be deployed either to an AIX operating system with version 5.3 or to a Windows™ 7 operating system with a version higher than 5.1.2600. The OperatingSystem type used here is an OSType of the CIM_OperatingSystem class defined in the Starter Profile [SDDSP]. The JRE requirement specifies that the minimum supported version of JRE is 1.4.1. The disk space requirement specifies that the minimum available disk space is 2000 blocks at the size of 512 bytes per block. The element that specifies disk space is ConsumptionConstraint with a property defined as AvailableSpace for a CIM_FileSystem class. These Requirements for the simple client are shown in the following example excerpt.

```xml
<sdd-dd:InstallableUnit id="App01" targetResourceRef="os">
  <sdd-dd:Identity softwareID="3000-123">
    <sdd-common:Description>This is a local client interface for the simple application.</sdd-common:Description>
    <sdd-common:Name>Simple Application Client</sdd-common:Name>
    <sdd-common:Version>1.0</sdd-common:Version>
  </sdd-dd:Identity>
</sdd-dd:InstallableUnit>
```

7 Windows is a registered trademark of Microsoft Corporation in the United States and other countries.
Requirements on Windows XP

Install Location

<sp:CIM_OperatingSystem.OSType value="Windows XP" />

Windows requiremen...
Note the Variables section. It declares an input parameter, InstalledLocation, for the J2EE Client deployment descriptor. The input parameter is declared as URIParameter, which implicitly declares the value to be a valid Uniform Resource Identifier.

Note the RelationshipConstraint element. It defines a relationship between two resources within the scope of a ResourceConstraint. The first resource is declared with the resourceRef attribute of the ResourceConstraint; the second is declared with relatedResourceRef in the RelationshipConstraint. The relationship is declared with the type attribute of the RelationshipConstraint (in this case, the type is "connects"). The deployment runtime determines the semantics associated with the particular RelationshipConstraint. The RelationshipConstraint is shown in the following example excerpt.
The first ResultingResource element declares that deployment of the J2EE Client will result in a resource called SimpleAppClient, with a property named Directory whose value will be equal to the value of the InstallLocation variable.

The second ResultingResource element declares that deployment of the J2EE Client will result in a directory type resource. As specified by the resourceRef attribute, this is the installation directory that was also declared earlier in Topology as a HostedResource (see the “Topology” description earlier in this section). Together, the Topology and ResultingResource information indicates that the InstallDir resource is a resulting resource that is hosted by the file system and takes its name from the value of the InstallLocation variable. This directory ResultingResource is declared because it is necessary to bring the installation directory resource into scope and to specify that this installation directory is, in fact, the one that was declared in Topology and it must be hosted by the file system that was declared in Topology.

The ResultingResources are shown in the following example excerpt.

```
<sdd-dd:ResultingResource resourceRef="SimpleAppClient">
  <sdd-dd:Name>Simple Application Client</sdd-dd:Name>
  <sdd-dd:Version>1.0</sdd-dd:Version>
  <sdd-dd:Property>
    <sdd-dd:PropertyName>Directory</sdd-dd:PropertyName>
    <sdd-dd:Value>${InstallLocation}</sdd-dd:Value>
  </sdd-dd:Property>
</sdd-dd:ResultingResource>

<sdd-dd:ResultingResource resourceRef="InstallDir"/>
```

- Once the requirements are met, the Artifacts element defines the files that need to be processed for the deployment operation. Besides the normal InstallArtifact, this example also has an UninstallArtifact that describes how to uninstall an existing simple application client. In this example, only the InstallArtifact is used in the context of the composite application, because the install operation declared by the CompositeInstallable for the composite application effectively “selects” the InstallArtifact in the SimpleAppClient (the UninstallArtifact might be used in other deployment contexts). Note also that the SimpleAppClient SDD itself is a Conformance Level 1 (CL1) SDD (it has no CompositeInstallable), even though in the context of this example it is being aggregated into a Conformance Level 2 (CL2) solution (see the “Conformance” chapter of the specification [see Section 0 of this document] for additional information). These Artifacts are shown in the following example excerpt.

```
<sdd-dd:Artifacts>
  <sdd-dd:InstallArtifact type="jar" contentRef="SAC_InstallArtifact">
    <sdd-dd:Arguments>
      <sdd-dd:Argument name="${InstallLocation}" />
    </sdd-dd:Arguments>
  </sdd-dd:InstallArtifact>

  <sdd-dd:UninstallArtifact type="jar" contentRef="SAC_UninstallArtifact">
    <sdd-dd:Arguments>
      <sdd-dd:Argument name="${InstallLocation}" />
    </sdd-dd:Arguments>
  </sdd-dd:UninstallArtifact>
</sdd-dd:Artifacts>
```
4.3.1.3 JRE Runtime

The third referenced package is a Java runtime that is a Requisite of the composite package. The JRE’s PackageDescriptor is located in pkgs\JRE\jre_pkg.xml and the DeploymentDescriptor is located in pkgs\JRE\jre.xml in [SDDEX] (see Appendix [A]).

- The JRE example is a simple package that was described in Section [4.2].
- This ReferencedPackage is associated with resource constraints via the properties of the resulting resources that are declared in its ResultingResourceMap. In this example, the Name and Version of the JRE are compared with constraints on the Name and Version of the JRE that are declared in Requirements in SDDs elsewhere in the composite application hierarchy (for example, the JRE01.reqt Requirement that is defined in SimpleAppClient—see section [4.3.1.2]).
- The resulting resource from the installation of this package has an id value of “JRE” and a type of “sp:CIM_InstalledProduct”.

4.3.1.4 Language Packs

The final referenced packages for this composite example are the language packages located in the pkgs\Languages directory in [SDDEX] (see Appendix [A]).

- These language packages are similar to other simple packages already described. Note that in these packages, the language package can be installed or updated. For the install case, the InstallArtifact is used; whereas in the update case, the UpdateArtifact is used.
- Note that both language package deployment descriptors declare, in their respective Variables section, a required resource property with the name “Directory” from the resource SimpleApp1. This is shown in the following example excerpt from the pkgs\Languages\French Language Pack.xml.

```xml
<sdd-dd:Variables>
  <sdd-dd:ResourceProperty resourceRef="SimpleApp1"
    id="InstallLocation" propertyName="Directory">
    <sdd-dd:Description>Install location for Simple Demonstration Application</sdd-dd:Description>
    <sdd-dd:ShortDescription>Install location for Simple Demonstration Application</sdd-dd:ShortDescription>
  </sdd-dd:ResourceProperty>
</sdd-dd:Variables>
```

4.3.2 Composite PackageDescriptor

The complete composite PackageDescriptor example for a J2EE application and its associated components is in the file examples/CompositeApp/CompositeApp_pkg.xml in [SDDEX] (see Appendix [A]).

- Compared with the simple PackageDescriptor as shown in the previous example, the composite PackageDescriptor has similar identity information, including Description, ShortDescription, Version, BuildInformation and Manufacturer information. All of this information is provided by the package author.
- The composite PackageDescriptor differs from the simple descriptor in the content section. The content section contains constituent and referenced PackageDescriptors that identify the components to be installed. These components include:
  - Three-tier J2EE application package (SC_pkg)
4.3.3 Composite DeploymentDescriptor

The complete composite DeploymentDescriptor example for a J2EE application and its associated components is in the file examples/CompositeApp/CompositeApp.xml in [SDDEX] (see Appendix [A]).

- The composite DeploymentDescriptor first describes a Topology with nested resources for the components to be installed. The Resource element is used to describe the top level resources (in this case, the operating systems in which each of the components of the composite application will be installed). The type of each operating system resource is described by using the CIM_OperatingSystem class defined in the Starter Profile [SDDSP]. Each operating system then contains HostedResources that are the components of the composite application: the servlet, application server, database and client portions, respectively. These nestedResource types, in turn, are defined by the types defined in the Starter Profile, namely, CIM_J2eeServer, CIM_J2eeServlet, CIM_J2eeApplication, CIM_DatabaseSystem, CIM_CommonDatabase, CIM_InstalledProduct and CIM_Application. Note that there are three top-level resources of type cim:CIM_OperatingSystem. The first declares the hosting resource where the J2EE application will be installed; the second declares the hosting resource where database will be installed; and the third declares the hosting resource where both the JRE and J2EE client will be installed. Because the JRE is required by the J2EE client, they are both hosted by the same resource. The three operating system resources do not explicitly declare that they represent three separate physical resources. Instead, they implicitly declare that these resources may represent different physical resources. Later in the deployment descriptor, these resources are mapped to the resources in the topologies in the referenced SDDs. This Topology is shown in the following example excerpt.

```xml
<sdd-dd:Topology>
  <sdd-dd:Resource id="servlet_os" type="sp:CIM_OperatingSystem">
    <sdd-dd:HostedResource id="J2eeServletServer" type="sp:CIM_J2eeServer">
      <sdd-dd:HostedResource id="SimpleJ2eeServlet" type="sp:CIM_J2eeServlet" />
    </sdd-dd:HostedResource>
  </sdd-dd:HostedResource>
</sdd-dd:Topology>
```
The CompositeInstallable element then describes the deployment of the aggregated packages and the associations among these packages. The content of the aggregating composite application DeploymentDescriptor in the example contains BaseContent, SelectableContent, and LocalizationContent.

The Variables element defines the input parameter for the composite deployment descriptor.

The BaseContent hierarchically defines the non-optional content for the deployment operation by using the ContainedPackage element. The ContainedPackage in this example is the 3-tier J2EE application package (SC_Pkg) as described in the PackageDescriptor. The ContainedPackage element in this case can pass arguments to the referenced package of SC_Pkg through the Argument element, such as JDBC_User and JDBC_Password (note that the DeploymentDescriptor also contains Parameter Variables corresponding to these Arguments). ContainedPackage also maps the resources in the composite DeploymentDescriptor to the referenced DeploymentDescriptor (the Topology in the referenced J2EE application SDD, SC_pkg) through ResultingResourceMap and RequiredResourceMap. The BaseContent is shown in the following example excerpt.

```
<root>
  <sdd-dd:BaseContent>
    <sdd-dd:ContainedPackage id="SimpleCompositeApp_PKG" contentRef="SC_pkg"/>
    <sdd-dd:Arguments>
      <sdd-dd:Argument name="JDBC_User" value="$(DatabaseUserName)"/>
      <sdd-dd:Argument name="JDBC_Password" value="$(DatabaseUserPassword)"/>
    </sdd-dd:Arguments>
  </sdd-dd:BaseContent>
</root>
```
• The SelectableContent defines contents that are selectable by feature. The Feature element within SelectableContent defines features that are associated with content elements that are defined within SelectableContent. In this example, only one feature is defined: the client application is considered to be an add-on feature and the client application package is defined as a ContainedPackage in SelectableContent. Note the mapping between resources in the composite deployment descriptor and resources in the topology of the referenced J2EE client SDD, SAC_pkg. The SelectableContent is shown in the following example excerpt.

• The following example excerpt also includes Requirements (discussed previously). Requirement id values require special consideration in composite (aggregated) SDDs. Note that the id values for the Requirements must be unique within any single deployment descriptor. Moreover, these id values must be unique across multiple deployment descriptors that are aggregated together. The SDD author must ensure that Requirement id values are unique throughout the composite SDD. SDD consumers may qualify the Requirement id values with the id value of each deploymentDescriptor; this effectively uses the deploymentDescriptor id value as a namespace to distinguish Requirement id values to further ensure uniqueness of the individual deploymentDescriptors that are aggregated.

```xml
<sdd-dd:SelectableContent>
  <sdd-dd:Features>
    <sdd-dd:Feature id="ClientFeature" addOn="true">
      <sdd-dd:DisplayName>Thick Client for Simple Application</sdd-dd:DisplayName>
      <sdd-dd:Multiplicity multiplesAllowed="true"/>
      <sdd-dd:ContentType contentElementRef="SimpleAppClient_PKG"/>
    </sdd-dd:Feature>
  </sdd-dd:Features>
  <sdd-dd:ContainedPackage id="SimpleAppClient_PKG" contentRef="SAC_pkg">
    <sdd-dd:Arguments>
      <sdd-dd:Argument name="InstallLocation" value="$(ClientInstallLocation)"/>
    </sdd-dd:Arguments>
    <sdd-dd:Requirements>
      <sdd-dd:Requirement id="solutionJRE.reqt" operation="installUse">
        <sdd-dd:ResourceConstraint id="SolutionJRE.check" resourceRef="JRE"/>
      </sdd-dd:Requirement>
    </sdd-dd:Requirements>
  </sdd-dd:ContainedPackage>
</sdd-dd:SelectableContent>
```
The LocalizationContent in the CompositeInstallable defines the localization information and resources. In this example, the ContainedLocalizationPackages define the packages whose contents enable resources to be localized for German and French. These ContainedLocalizationPackages point to individual PackageDescriptors (DE_Lang_PKG and FR_Lang_PKG) as defined in the composite PackageDescriptor. The LocalizationContent is shown in the following example excerpt.

```
<root>
  <dd:LocalizedPackageDescriptor>
    <dd:Supported>
      <dd:Range>
        <dd:MinVersion>1.4.1</dd:MinVersion>
      </dd:Range>
    </dd:Supported>
    <dd:Certified>
      <dd:Value>
        <dd:Version>1.5.0</dd:Version>
      </dd:Value>
    </dd:Certified>
    <dd:VersionConstraint>
      <dd:ResourceConstraint id="client_connectivity" resourceRef="Client_OS">
        <dd:RelationshipConstraint relatedResourceRef="appServer_os" type="connects">
          <dd:Requirement>
            <dd:Property>
              <dd:PropertyId>Protocol</dd:PropertyId>
              <dd:Value>TCP/IP</dd:Value>
            </dd:Property>
          </dd:Requirement>
          <dd:Requirements>
            <dd:ResultingResourceMap resourceRef="SimpleAppClient" foreignId="SimpleAppClient">
              <dd:Relationship relatedResourceRef="SimpleJ2eeApp" type="connects">
                <dd:Property>
                  <dd:PropertyId>Protocol</dd:PropertyId>
                  <dd:Value>HTTPS</dd:Value>
                </dd:Property>
              </dd:ResultingResourceMap>
            </dd:Requirements>
          </dd:Requirements>
        </dd:RelationshipConstraint>
      </dd:ResourceConstraint>
    </dd:VersionConstraint>
    <dd:ResourceConstraint id="Client_OS">
      <dd:RelationshipConstraint relatedResourceRef="appServer_os" type="connects">
        <dd:Requirement>
          <dd:Property>
            <dd:PropertyId>Protocol</dd:PropertyId>
            <dd:Value>HTTPS</dd:Value>
          </dd:Property>
        </dd:Requirement>
        <dd:Requirements>
          <dd:ResultingResourceMap resourceRef="Client_OS" foreignId="os">
            <dd:Requirement>
              <dd:Property>
                <dd:PropertyId>Protocol</dd:PropertyId>
                <dd:Value>HTTPS</dd:Value>
              </dd:Property>
            </dd:Requirement>
            <dd:Requirements>
              <dd:ResultingResourceMap resourceRef="JRE" foreignId="JRE"/>
            </dd:Requirements>
          </dd:ResultingResourceMap>
        </dd:Requirements>
      </dd:RelationshipConstraint>
    </dd:ResourceConstraint>
  </dd:LocalizedPackageDescriptor>
</root>
```

- The LocalizationContent in the CompositeInstallable defines the localization information and resources. In this example, the ContainedLocalizationPackages define the packages whose contents enable resources to be localized for German and French. These ContainedLocalizationPackages point to individual PackageDescriptors (DE_Lang_PKG and FR_Lang_PKG) as defined in the composite PackageDescriptor. The LocalizationContent is shown in the following example excerpt.
• Note the Languages element in the composite deployment descriptor. It declares as mandatory the language “en-US”, which is the default (implicit) language of the J2EE Client. Two optional languages, French and German, are declared; the first with a LanguageSet element that declares two languages, “fr-FR” and “fr-CA” (French and French Canadian) for the set French; the second with a Language element that declares a single language “de-DE” for language German. These languages map to the related ContainedLocalizationPackage element.

```xml
<sdd-dd:Languages>
  <sdd-dd:Mandatory>
    <sdd-dd:Language type="en-US" />
  </sdd-dd:Mandatory>
  <sdd-dd:Optional>
    <sdd-dd:LanguageSet>
      <sdd-dd:Description>French</sdd-dd:Description>
      <sdd-dd:Language type="fr-FR" />
      <sdd-dd:Language type="fr-CA" />
    </sdd-dd:LanguageSet>
    <sdd-dd:Language type="de-DE">
      <sdd-dd:Description>German</sdd-dd:Description>
    </sdd-dd:Language>
  </sdd-dd:Optional>
</sdd-dd:Languages>
```

• Besides ContainedPackage and ContainedLocalizationPackage, a Requisites element can also be used in the aggregation of SDDs. The Requisites element is used to identify an SDD package that can be deployed, if necessary, to satisfy a resource constraint. In this example, the Requisite uses a ReferencedPackage element to refer to a JRE package that can be deployed to satisfy the requirement for the client application if no JRE that satisfies the requirement is present in the deployment environment. ResultingResourceMap is used to map the resource JRE from the composite deployment descriptor to the resource JRE in the referenced JRE SDD, JRE_pkg. The Requisites element is shown in the following example excerpt.

```xml
<sdd-dd:Requisites>
  <sdd-dd:ReferencedPackage id="JRE_Requisite" contentRef="JRE_pkg">
    <sdd-dd:ResultingResourceMap resourceRef="JRE" foreignId="JRE">
      <sdd-dd:Name>Java(TM) Runtime Environment, Standard Edition</sdd-dd:Name>
      <sdd-dd:Version>1.5.0</sdd-dd:Version>
    </sdd-dd:ResultingResourceMap>
  </sdd-dd:ReferencedPackage>
</sdd-dd:Requisites>
```
5 Additional Considerations

The preceding examples illustrate basic concepts of the SDD for software installation and are intended to serve as “getting started” content. Future publications may detail other uses of SDD beyond installation, such as updates, configuration and localization. A few other considerations for SDDs that deal primarily with installation but do not appear in the preceding examples are described next.

- **Monolithic artifacts**: The composite application example described in Section [4.3] showed how multiple artifacts can be present in a single DeploymentDescriptor, enabling individual components to be appropriately installed to deploy the complete solution.

  Another form of an artifact that might be present in a DeploymentDescriptor is a monolithic artifact. A monolithic artifact is a single Artifact for a single target that has content variability (selectable content) and that a particular runtime understands how to process. A monolithic artifact exposes selectable content that can be deployed from the single artifact (rather than selecting particular artifacts for deployment, as was illustrated in the preceding Conditions example). SDDs with monolithic artifacts do not differ substantially from other SDDs; the DeploymentDescriptor can still describe Topology, Identity, Requirements, Variables, and so on. The example in examples/Monolithic/Monolithic.xml in [SDDEX] (see Appendix [A]) contains a DeploymentDescriptor that illustrates this concept.

  In that example, the DeploymentDescriptor contains a single artifact with selectable content described as two Features—a representative application feature and help files—that optionally can be deployed. The SDD standard was designed to support either model; this accommodates multiple kinds of artifacts, including existing or legacy artifacts (this design choice contributed to the decision to have two different conformance levels for the SDD standard; see the “Conformance” chapter of the specification [see Section 0 of this document] for additional information). Existing artifacts (monolithic artifacts and others) can still benefit from use of the SDD by “wrapping” the artifacts in the declarative SDD metadata that adds valuable deployment information. This then enables existing runtimes to benefit from the SDD for deployment planning and enhanced deployment.

- **One operation/one artifact rule**: As described in the SDD specification [see Section 0 of this document], restrictions of a single operation per CompositeInstallable and a single artifact per InstallableUnit apply for aggregated SDDs. Here we explain this rule in more detail, including its rationale.

  A CompositeInstallable aggregates multiple artifacts that together support the application of one operation to the overall software. The SDD specification stipulates that each CompositeInstallable defines only a single operation and that all InstallableUnits in a CompositeInstallable define only a single artifact. These rules prevent ambiguity that would otherwise arise because some of the elements of CompositeInstallable and InstallableUnit are defined per operation; whereas other elements have implicit meanings that vary based on the operation or typically apply only to a particular operation.

  Allowing only one operation per CompositeInstallable prevents ambiguity about the meaning of Feature, ResultingResource and ResultingChange elements defined in the CompositeInstallable. Features are mechanisms to select portions of content for a particular deployment and can be used with any operation. Features do not have long-lived identity; rather, they exist to enable choosing the content of the deployment package. A Feature defined for an install operation is likely to be unambiguous (typically representing a component or capability that optionally can be included during the installation). A Feature
defined for an update operation, however, could be equivocal (for example, is the *Feature* a particular, optional, portion of the update, similar to the install case, or is it rather an update to an existing *Feature* that was previously selected during install?). For SDD, a single definition must be chosen, and the SDD specification opts for the former (that is, *Features* select a particular, optional portion of the content for the operation that is specified). Hence, a *CompositeInstallable* defines only one operation because it can define only one set of *Features*.

When applying the one operation defined in the *CompositeInstallable* to the overall software, it may be necessary to perform a variety of operations on individual resources. For example, to update a database product, it might be necessary to create (install) a new component. The overall operation for this *CompositeInstallable* is *update*, but it contains—somewhere in the aggregation—an *InstallArtifact* for the new component. Limiting the *CompositeInstallable* to a single operation (update in this example) prevents uncertainty about contained artifacts that specify different operations—if the artifact is defined (even though, in this example, it is an *InstallArtifact*), it is intended to support the one overall operation (update in this example) of the *CompositeInstallable*.

Allowing only one artifact in an *InstallableUnit* that is defined within a *CompositeInstallable* also prevents ambiguity in the interpretation of *RequiredBase*, *ResultingResource* and *ResultingChange* elements. If multiple artifacts were permitted in a single *InstallableUnit*, then it could be unclear which artifact’s results were described by *ResultingResource* or *ResultingChange*; it also could be unclear as to which artifact(s) require(s) the *RequiredBase*. 
6 Conclusion

This Primer has offered “getting started” information that illustrates the value, rationale and use of SDD for software deployment, focusing mainly on software installation. Several of the published SDD examples were described.

As previously stated, this Primer does not take the place of the SDD specification, schema and profiles (see sections [0] and [Error! Reference source not found.]). The Starter Profile published by the OASIS SDD Technical Committee includes types and values used in the examples, but does not include all types and values that might be used in particular SDDs. However, existing profiles can be extended and new profiles generated to define such values, using the pattern established by the Starter Profile, as described in [SDDSP].

Finally, as implementation experience with the SDD standard is gained, Best Practices will be developed and shared in the SDD community.
A. Complete Examples

- The latest version of example SDDs [SDDEX] that illustrate the use of the schema can be found at:
  http://docs.oasis-open.org/sdd/sdd/v2.0/sdd-examples/

These examples include those that are excerpted and described elsewhere in this document.
B. Acknowledgements

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Dr. Howard Abrams, CA
Mr. Lazar Borissov, SAP AG
Ms. Debra Danielson, CA
Mr. Robert DeMason, SAS Institute, Inc.
Mr. Robert Dickau, Macrovision Corporation
Mr. Quenio dos Santos, Macrovision Corporation
Mr. Adrian Dunston, SAS Institute, Inc.
Mr. Randy George, IBM
Mr. Nico Groh, SAP AG
Mr. Jeff Hamm, SAS Institute, Inc.
Ms. Merri Jensen, SAS Institute, Inc.
Mr. Jason Losh, SAS Institute, Inc.
Ms. Julia McCarthy, IBM
Mr. Mark McCraw, SAS Institute, Inc.
Mr. Brent Miller, IBM
Mr. Ed Overton, SAS Institute, Inc.
Mr. Chris Robsahm, SAP AG
Mr. Thomas Studwell, Dell
Dr. Weijia (John) Zhang, Dell
C. Revision History

The following are the changes between the “Solution Deployment Descriptor (SDD) Primer v1.0” document and the “Solution Deployment Descriptor (SDD) Primer v2.0” document (this document).

- Changed version number from 1.0 to 2.0; updated publication dates
- Adapted document to new OASIS template for non-standards-track deliverables
- Moved all references from body to cover page to facilitate updates
- Added `descriptorLanguageBundle` illustration and explanation in section [4.2.1]
- Added clarification about `id` value uniqueness in sections [4.2.2] and [4.2.2]
- Various non-substantive editorial updates and modernizations
- Updated reference to SDD Examples in Appendix [Error! Reference source not found.] to cite v2.0 Examples.
- Updated list of contributors in Appendix [B]
- Added this Appendix [C] to list summary of changes to the document from version to version