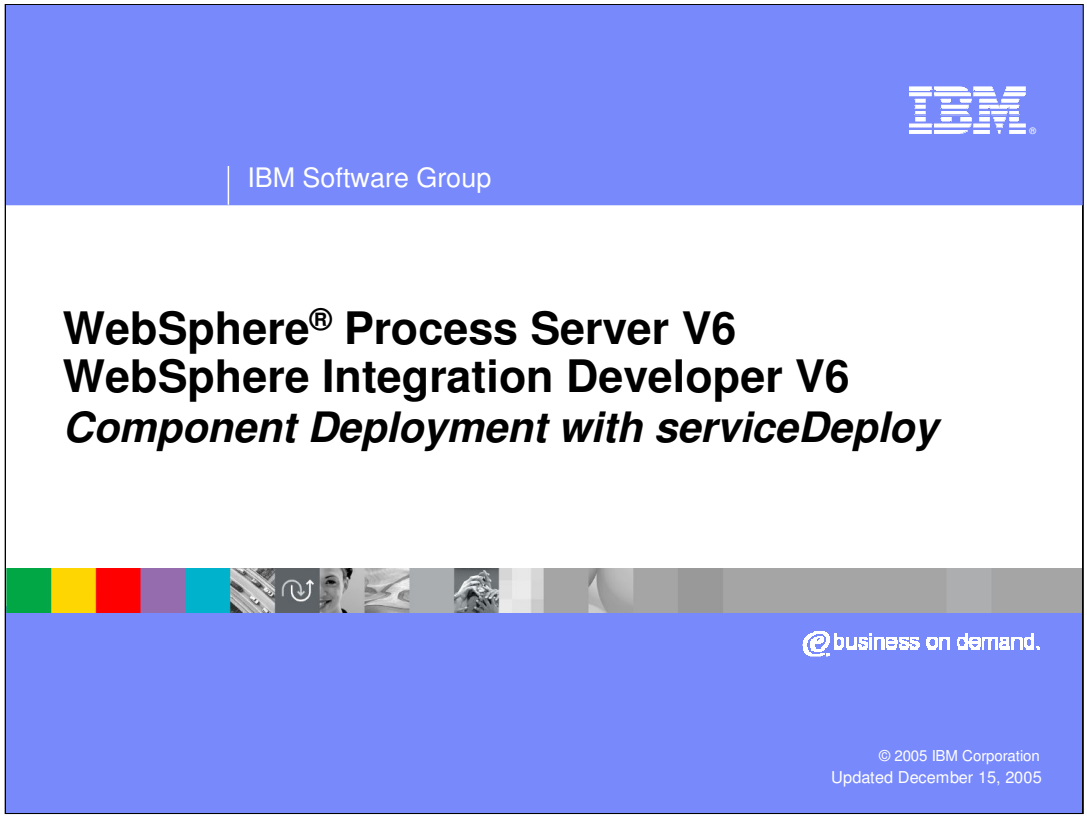


This presentation will focus on the serviceDeploy command included with WebSphere Process Server and WebSphere Integration Developer V6.0.

Goals

- Explain deployment automation with the serviceDeploy tool
- List known limitations and best practices along with troubleshooting information

The goals of this presentation are to explain the features and capabilities of the serviceDeploy tool for automating component deployment and to discuss the limitations, best practices, and troubleshooting information for serviceDeploy.

Agenda

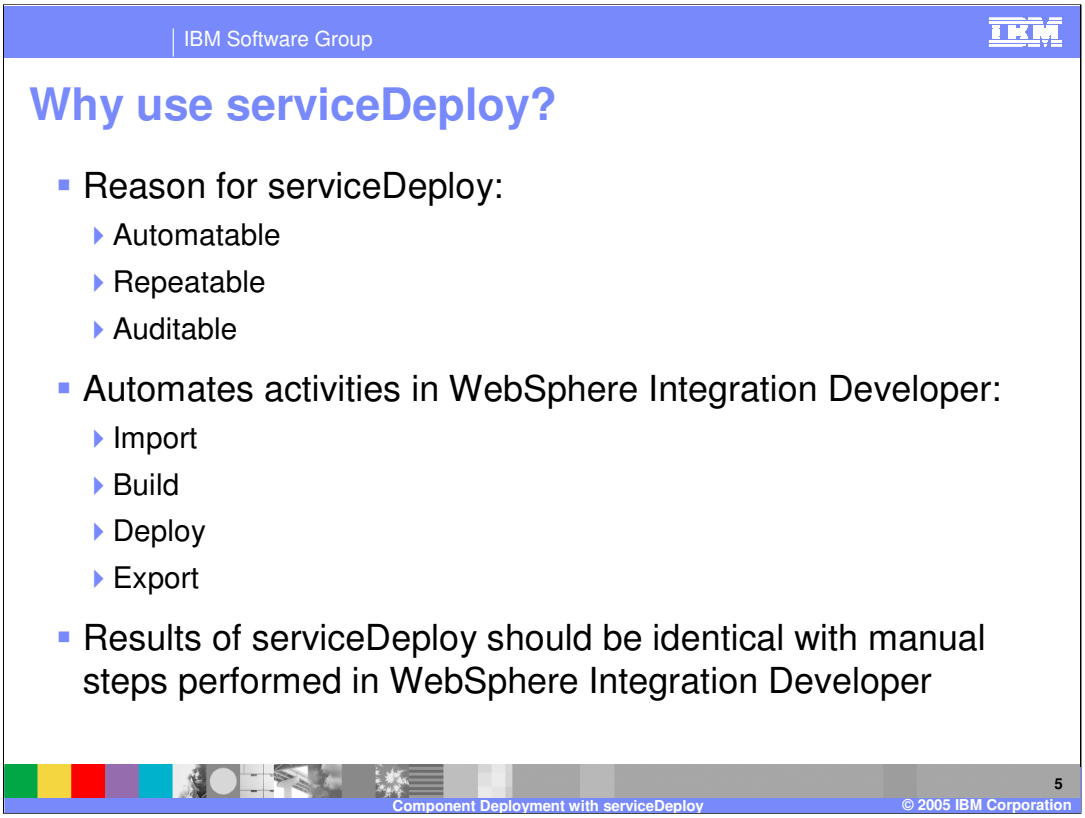
- serviceDeploy Overview
- serviceDeploy Command and Details
- Troubleshooting serviceDeploy
- Summary

This section will provide an overview of the serviceDeploy command.

serviceDeploy Overview

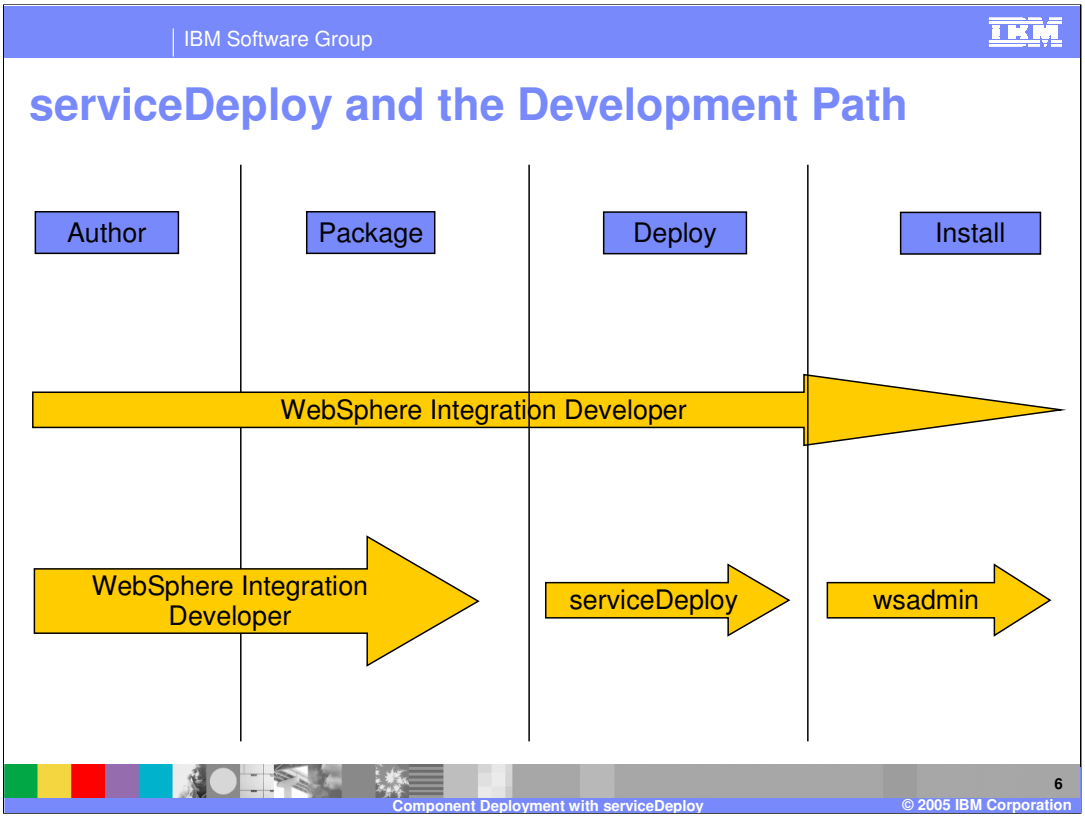
- Automated deployment (assemble/generate/compile) of Service Component Architecture (SCA) applications
- Command line or ANT task options
  - SCA resources (module) and other application components (WAR, EJB JAR, RAR, utility JAR) passed as a single input archive
  - Deployment provides basic assembly for service components and generates J2EE™ application artifacts and packages into a single EAR ready for installation
    - Installation must be completed using Administrative console or wsadmin
- Available on all WebSphere Process Server supported platforms

The serviceDeploy capability is designed to help automate deployment of SCA modules. This feature is designed to meet the requirement to automate the build process for enterprise development. It provides the capability to take multiple components built by multiple developers and bring them together into an application that can be installed. The serviceDeploy capability is available as a separate application that can be called from the command line. There is also an ANT task that will call serviceDeploy for larger build and deploy processes. The serviceDeploy command is available for all WebSphere Process Server supported platforms.

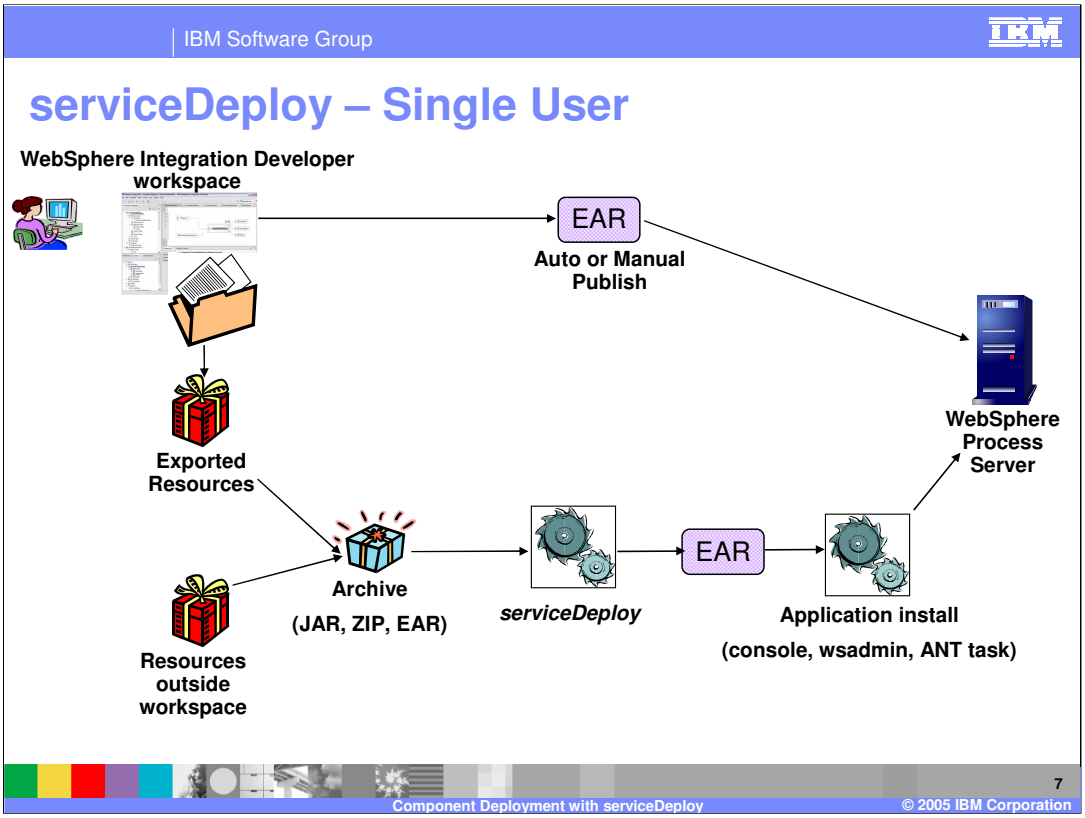


The serviceDeploy command is not intended to be an alternative to WebSphere Integration Developer. It is intended to support in production environments automated, repeatable, and auditable processes for turning the applications written by the development teams (or purchased from third-parties) into an application that will run in the production environment. The expected development model is to write and test fully in WebSphere Integration Developer. When complete, the authored projects are stored in a repository. These projects can be extracted, deployed, installed, and controlled by an automated process. Again, this parallels very closely the way ejbdeploy is positioned.

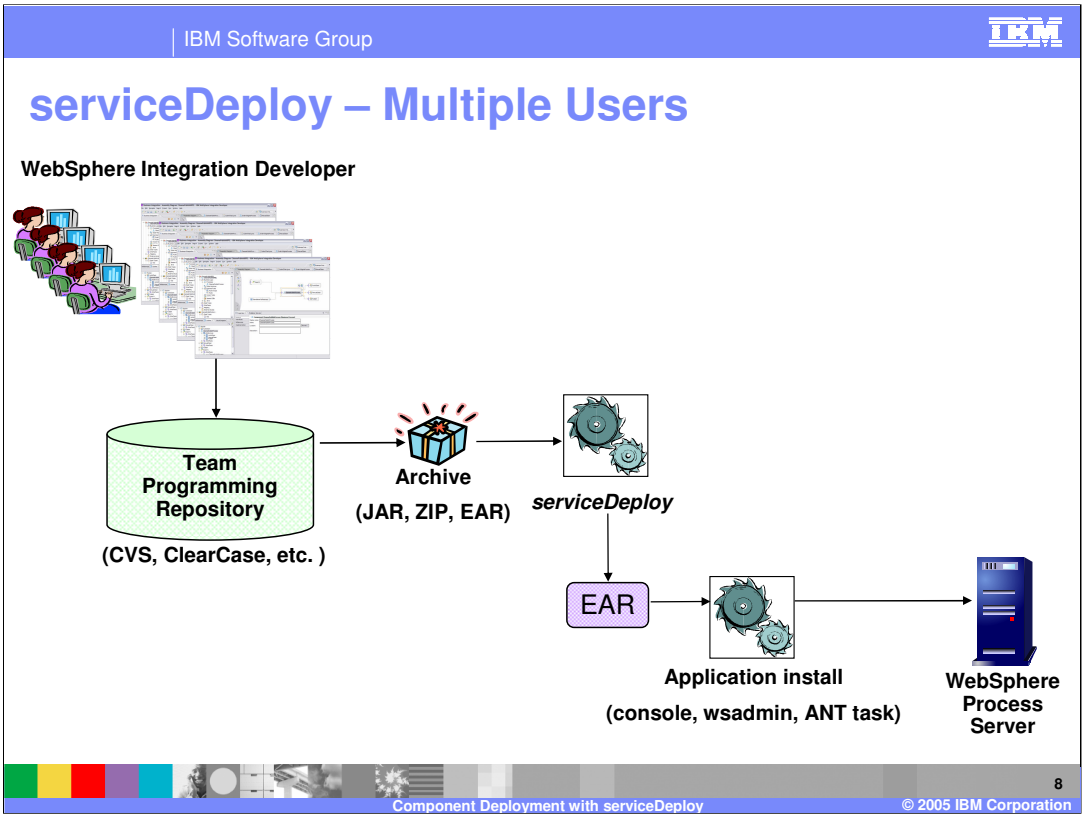
The serviceDeploy command simply automates those steps that a developer would follow in WebSphere Integration Developer. It imports the artifacts, builds the appropriate artifacts, runs the WebSphere-specific deployers (EJB and web service deployers) and exports the final EAR file. It does so by using the exact same code base that runs in WebSphere Integration Developer. The builders, the deployers, and the validators are, with a couple of small exceptions, the same whether driven by WebSphere Integration Developer or serviceDeploy. Because there is no graphical user interface, progress and status messages are written to the console.



Developers author applications according to the WebSphere Process Server V6.0 services programming model. The authored application is an abstraction that is transformed into a J2EE application to be installed and run on a WebSphere Application Server. The entire process can certainly be run from within WebSphere Integration Developer, including even installing the application using the WebSphere Test Environment support in WebSphere Process Server. In addition, authored applications can be exported from WebSphere Integration Developer and the transformation to a J2EE application can be performed using serviceDeploy, the WebSphere Process Server command line deployer. This is a very close parallel to WebSphere Application Server's EJBDeploy utility.



Shown here is a graphical representation of a single developer using serviceDeploy to perform all the tasks necessary for deployment. Once all the components are all placed on the assembly editor, wired, and saved, all the necessary code will be generated to an EAR file that can be deployed manually or using an auto-publish mechanism to the server. A single user might want to use serviceDeploy for situations where there are resources outside the workspace that they want to add into the EAR file. ServiceDeploy is capable of creating the artifacts necessary for making that service component available to be run or installed and called as well as bringing in other resources and dependencies such as another Web module where there are web interfaces provided to call some component. This is for artifacts that you do not have in your workspace, but are available, for instance on a network server. It could also be used to bring multiple components together in a single EAR file. You would first create an archive, then pass it in to a serviceDeploy script and the output would be an EAR file. At that point, you could use some of the other install mechanisms such as the Administrative Console, wsadmin, or an ANT task. serviceDeploy is a script to help with deployment and does not install the application.



Shown here is a graphical representation of an environment with multiple developers working on different components. Their work could then be brought together, for instance in a nightly batch job, and the application installed with all the components on the server for testing the next morning. Developers can work in a team programming environment, storing resources in a repository such as IBM Rational® ClearCase® or CVS and using the appropriate APIs or commands to retrieve those artifacts out of the repository, placing them in an archive, passing that archive to the serviceDeploy tool and installing the resultant EAR file with the Administrative Console, wsadmin, or an ANT task.

Agenda

- serviceDeploy Overview
- serviceDeploy Command and Details
- Troubleshooting serviceDeploy
- Summary

This section will provide details of the serviceDeploy command.

serviceDeploy Command

- serviceDeploy command
  - Located at ${WPS_INSTALL_ROOT}\bin\serviceDeploy.bat(.sh) or ${PROFILE_DIR}\bin\serviceDeploy.bat(.sh)
  - Syntax

```
serviceDeploy archive [<-workingDirectory temppath>
<-outputAppliation outputpathname.ear> <-noJ2eeDeploy true|false>
<-freeform true|false > <-cleanStagingModules true|false>
<-keep true|false> <-ignoreErrors true|false>
<-classpath jarpathname;rarpathname;warpathname;...  -help]
```

- serviceDeploy example
  - Creates an application called MyTestApp.ear
  - Specifies dependencies mytestmoduleres.rar and mytestlibrary.jar
  - Runs the J2EE deployers
  - Keeps the temporary files generated during deployment

```
serviceDeploy MyTestModule.jar -outputApplication c:\temp\MyTestApp.ear
  -classpath ''c:\java\mytestmoduleres.rar;c:\java\mytestlibrary.jar'' -keep
```


The serviceDeploy command is located in the bin directory of your WebSphere Process Server installation. If the test environment option is selected for WebSphere Integration Developer, the serviceDeploy command will be located under the runtimes\bi_v6\bin directory of the server. You can also find the serviceDeploy command under the bin directory of the server profile. There are a variety of parameters that you can specify on the serviceDeploy utility. The main ones are the archive that you will start with, including the name of the module that contains the components that you want generated into an installable EAR file and the output name of that EAR file. There are some other options for troubleshooting or providing additional resources that can be used at build and deploy time.

In the example provided, an application called MyTestApp.ear will be created from the serviceDeploy command. At run time, the mytestmoduleres.rar and mytestlibrary.jar will be used by the command for compiling. The J2EE deployers will also run when the command is called. The –keep option has been specified to keep the temporary files for troubleshooting purposes.

serviceDeploy option for ANT Tasks

- ANT task
  - ${WPS}/bin/ws_ant -f build.xml
- ANT task example

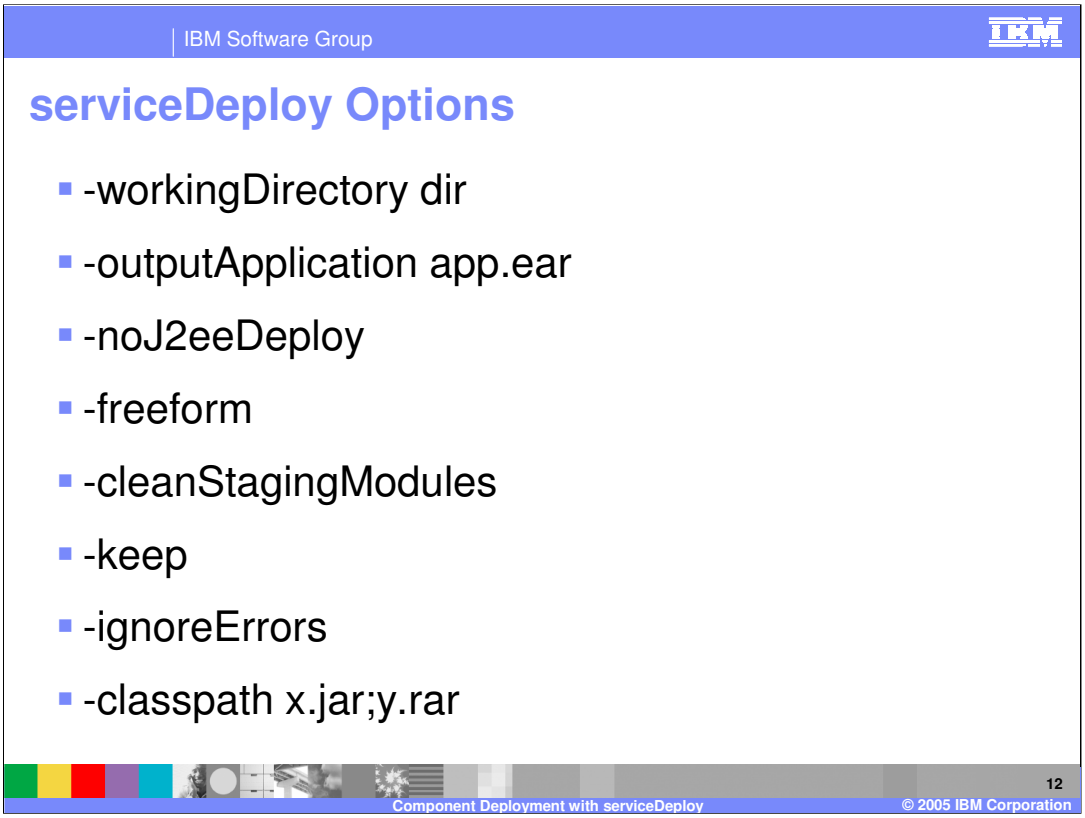
```
<?xml version="1.0"?>
<project name="scaDeploy" default="main" basedir=".">

      <taskdef name="servicedeploy
                classname="com.ibm.websphere.ant.tasks.ServiceDeployTask"/>

      <target name="main">
            <servicedeploy scaModule="AsyncSource.jar"
                  keep="true"
                  ignoreErrors="true"
                  outputApplication="AsyncSource-Deployed.ear"
                  workingDirectory="."
                  noJ2eeDeploy="true"
                  freeform="true"/>
      </target>

</project>
```


As stated earlier, serviceDeploy can also be used in ANT Tasks. Within an ANT task, you will specify the appropriate task definition or taskdef value and then for the servicedeploy task, you will specify the appropriate attributes to customizing running the application.



ServiceDeploy can complete with no input other than the archive to deploy. There are a variety of other parameters that can be used to control how serviceDeploy runs.

-workingDirectory dir

serviceDeploy will create a temporary eclipse workspace (for example, 107538b694f) in the directory where the command was launched. The directory where the workspace is created can be controlled using the –workingDirectory argument.

-outputApplication ear

serviceDeploy by default will export the ear in the directory where the command was launched, and named according to the service module (Xapp.ear). The outputApplication argument specifies both the path and the name of the resulting application.

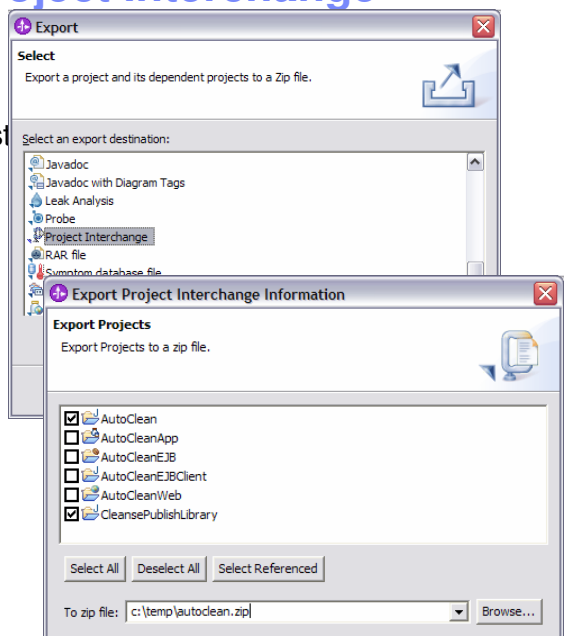
-noJ2eeDeploy

By default, serviceDeploy will run all registered deployers in order to transform the application into something that can successfully run on a WebSphere Process Server. This includes the two deployers that are inherited from WebSphere Application Server: EJB and web service deployers. The noJ2eeDeploy argument instructs serviceDeploy to bypass those deployers. This is useful when you need more control over how the base deployers are run.

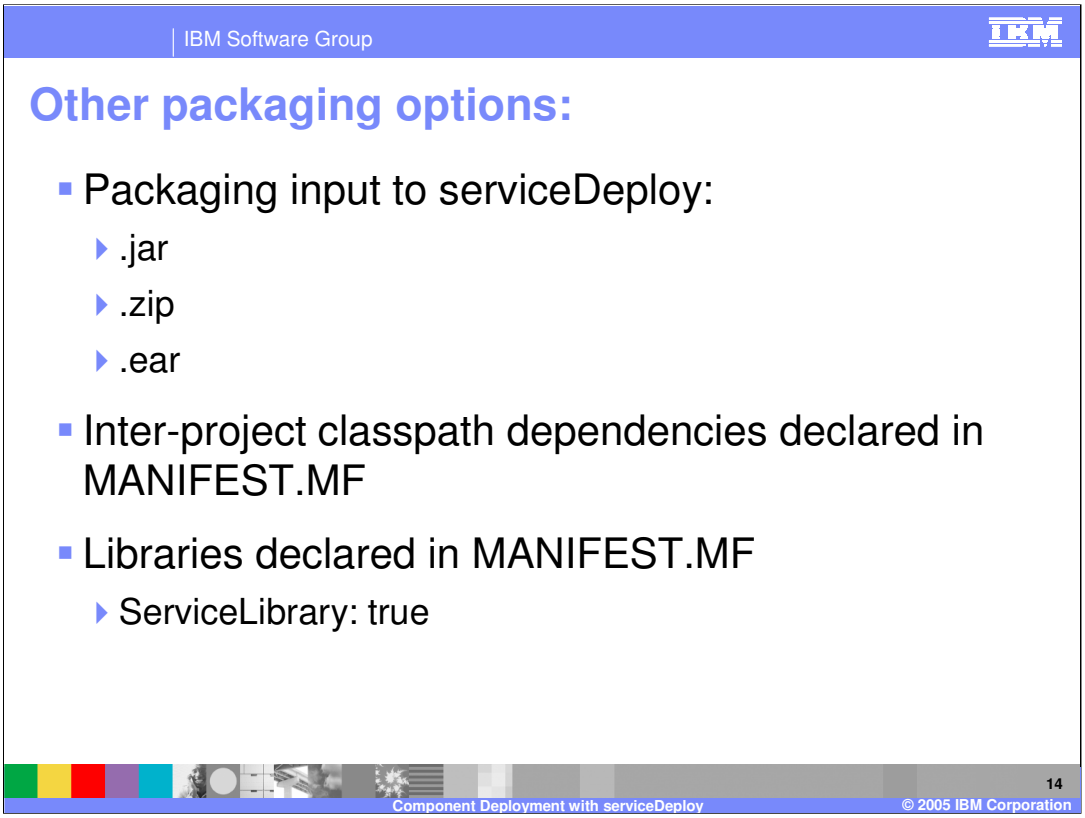
-freeform

Archive Packaging - Project Interchange

- Archives which are project interchanges from WebSphere Integration Developer are easiest to use
  - Recommended
- Must contain only a single module project
- Staging modules (EJB, EJBClient, Web, and EAR projects) are not required in Project Interchange
  - Projects will be replaced with new versions when serviceDeploy is created



serviceDeploy accepts as input different types of archives. The preferred archive type is project interchange, which can be exported directly from WebSphere Integration Developer. This archive contains the projects in a workspace, including all project metadata. The only restriction that serviceDeploy places on the archive is that it contain just a single business module. It must also contain all dependencies, including libraries and J2EE modules. Externally referenced classpath contributions will need to be passed in using the –classpath parameter to serviceDeploy if the referenced .jar file or zipped file does not reside at the configured path in the system on which serviceDeploy is run. The project interchange does NOT need to contain the automatically generated staging modules.



serviceDeploy also supports three other archives. It can handle a single jar that is a service module. It can contain a zipped file of jar files and other J2EE archives (zips, wars, and rars), all of which will be included in the resulting .ear file. It can also be run against an existing ear file.

Because none of these archives are closely associated with WebSphere Integration Developer, project interchange is preferred. There are a variety of restrictions. In all cases, there must be exactly one jar that contains the sca.module file. The sca.module file must be at the root of its jar. Jar files that represent libraries must be declared as such in the manifest so that serviceDeploy will know to configure the corresponding project with the right set of natures, builders, and so on. Classpath dependencies must also be declared in manifests, so that serviceDeploy will know to configure project classpath dependencies during deployment.

Anatomy of a workspace

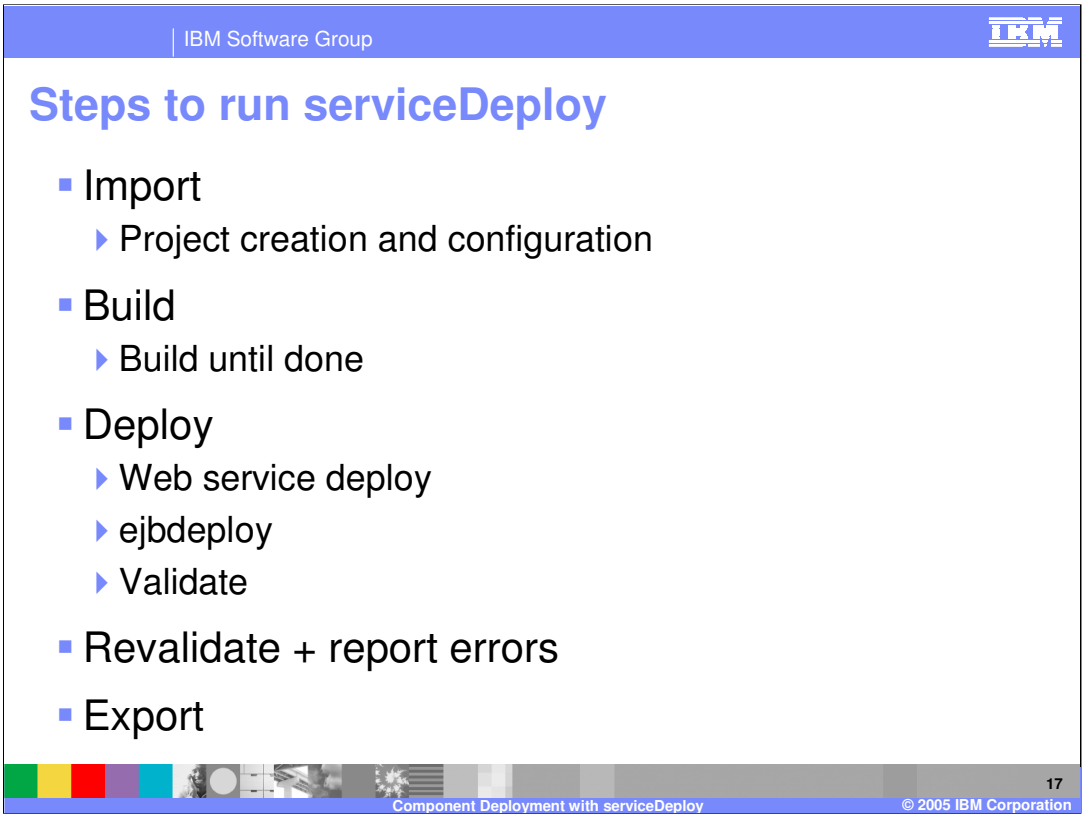
- Project creation
  - Primary module project
    - sca.module
    - .project has natures:
      - <nature>com.ibm.wbit.project.generalmodulenature</nature>
      - <nature>com.ibm.ws.sca.rapiddeploy.style.SCAProjectNature</nature>
  - Library (optional)
    - .project has nature:
      - <nature>com.ibm.wbit.project.sharedartifactmodulenature</nature>
    - Staging projects:
    - Other projects:

Project interchange preserves all the necessary metadata. Round tripping is easy. Projects can be stored in CVS, extracted and assembled (zipped) for production deployment, or extracted and imported directly into WebSphere Integration Developer. The other three archives do not import back into WebSphere Integration Developer in any natural way, and they must include enough information to enable the correct configuration of the project metadata.

Resource Adapter Files (RARS)

- ${WPS}/bin/serviceDeploy X.jar –classpath y.rar
  - XApp.ear:
    - X.jar
    - XEJB.jar
    - XEJBClient.jar
- ${WPS}/bin/serviceDeploy X.zip
  - XApp.ear:
    - X.jar
    - XEJB.jar
    - XEJBClient.jar
    - Y.rar

Resource Adapter (RAR) files may be installed onto the system as a service to multiple applications, or they may be included with and on behalf of a single application. When using project interchange, the inclusion or exclusion of the connector project is explicitly configured in the service project's .project file using WebSphere Integration Developer's dependency editor. RAR files included in the other zip format will be included in the EAR file. RAR files passed using the –classpath argument will be imported into the workspace, but they will not be exported with the application.



As serviceDeploy runs it performs a number of steps. serviceDeploy will first inspect the deployment argument to determine what kind of archive it is dealing with. If it is a project interchange, it unzips the archive into the workspace. Missing classpath entries are cleared; otherwise, the build will stop. If the archive is an EAR project, serviceDeploy will import it as a J2EE application, then fix up the project to transform it according to standards. Classpaths dictated by the manifest are applied if necessary; externally passed classpaths are also added.

serviceDeploy will then build the workspace. Eclipse first calculates the build order depending on declared dependencies across projects. serviceDeploy then builds the projects, one at a time. Circular dependencies are managed by building until the resources in the projects cease to change and the reported errors no longer decrease.

The deploy operation is started next if you have not requested that it be disabled. If there are errors in the project from the build, and you have not passed the –ignoreErrors argument, then the deploy operation will be skipped. Note that errors may cascade.

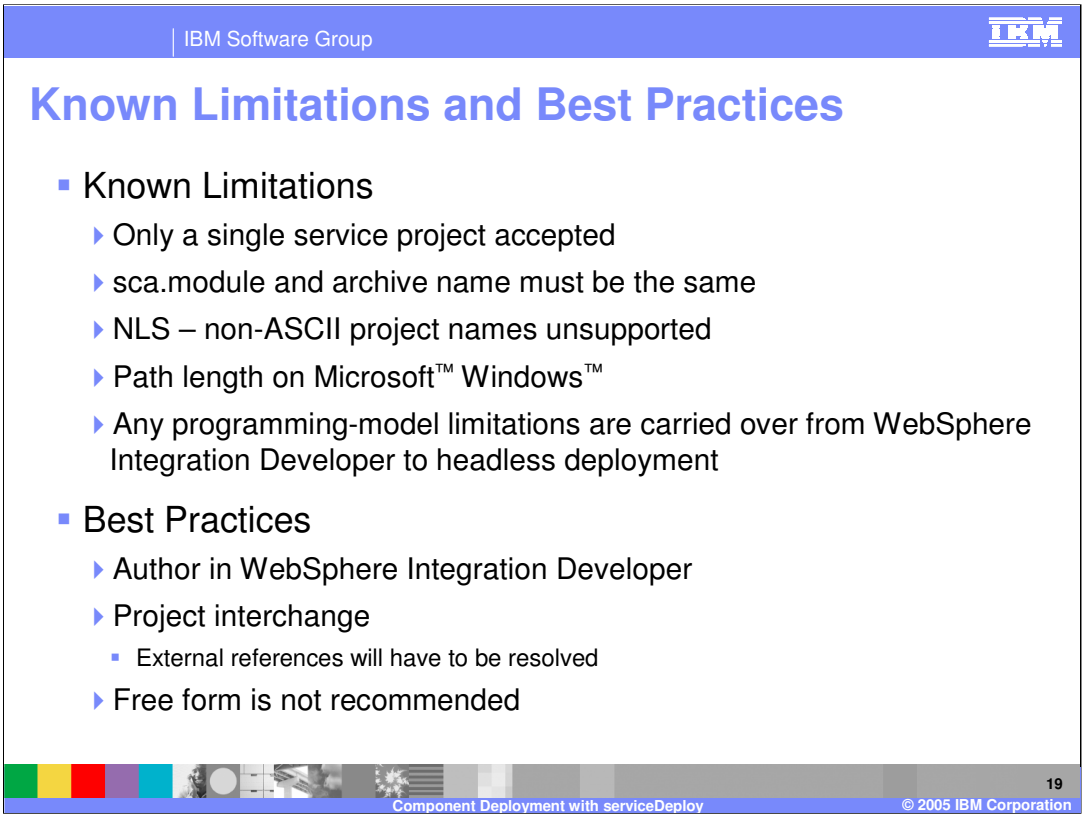
Finally, the validators are run once again and informational, warning, and error messages are printed to the screen. If there were no errors, or if the you have instructed serviceDeploy to ignore errors, then the application will be exported.

Errors in the deployment framework will prevent exporting an EAR file.

Agenda

- serviceDeploy Overview
- serviceDeploy Command and Details
- Troubleshooting serviceDeploy
- Summary

This section will cover problem determination for the serviceDeploy command.

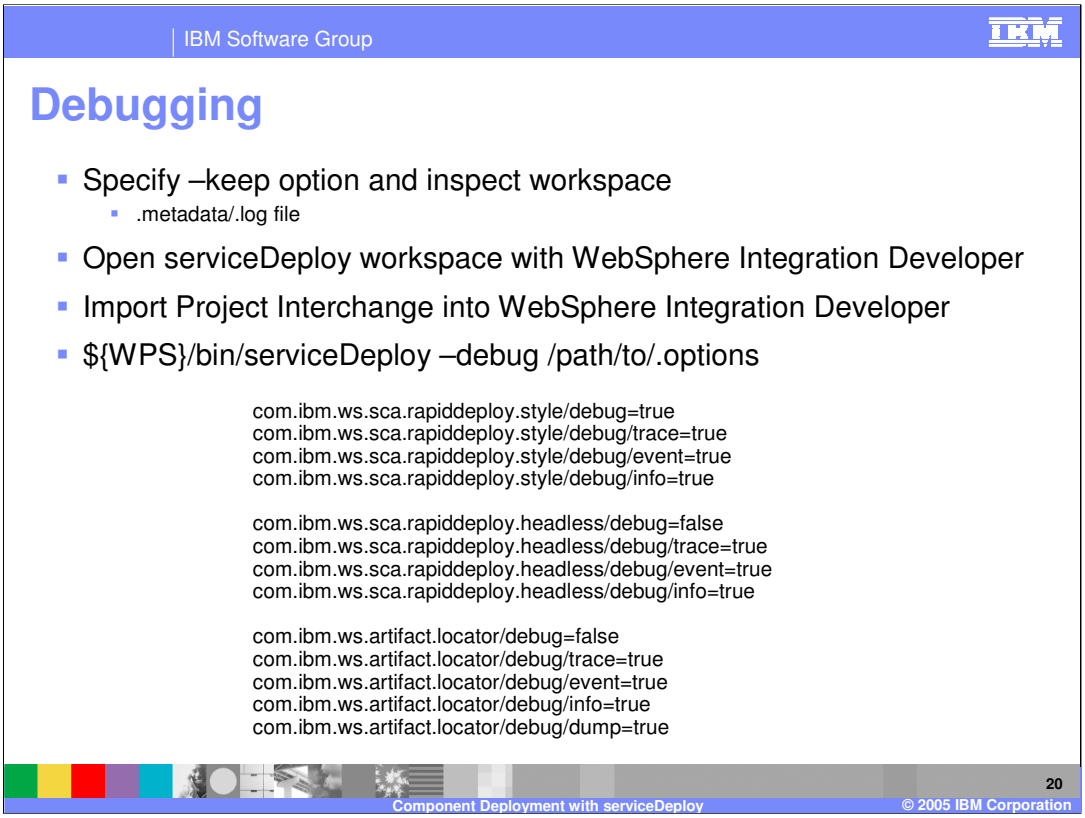


There are a number of known limitations. Only a single service project can be deployed with serviceDeploy. The sca.module and archive name must be the same. These limitations are by design.

Projects cannot be named with non-ASCII characters. Directory path length limitations on Windows at 256 characters can cause problems at any time while serviceDeploy runs, because files are constantly being written to the file system. Deploy close to the root of the file system to avoid these problems.

Finally, any programming restrictions in WebSphere Integration Developer are inherited by serviceDeploy. If the module will not build correctly in WebSphere Integration Developer, it will not build correctly with serviceDeploy.

There are a number best practices you can follow when using serviceDeploy. First, use WebSphere Integration Developer to create artifacts and define the service components.Second, use project interchanges so that applications can be explicitly configured for both serviceDeploy and WebSphere Integration Developer. Third, do not use free form; instead, create independent J2EE projects in WebSphere Integration Developer and use the dependency editor to associate them with the application.



If a problem arises, there are a number of debug steps you can perform to determine the problem. Usually, the workspace provides all the data necessary for debugging. Specifying the –keep option will prevent the deletion of the temporary workspace after serviceDeploy finishes running. Opening the workspace in WebSphere Integration Developer might clarify what are the issues when viewed in the editors. Importing the project back into WebSphere Integration Developer can help determine whether the problem is unique to the “headless” or “no user interface” environment. The log will usually contain enough information to find the component to begin troubleshooting with if it is not clear that the problem is rooted in the application.

The headless plug-in drives the mechanics of procedural deployment. The locator plug-in provides the implementation for the artifact loader. These are the only two plug-ins that ship with and run in headless only. The style plug-in controls much of the mechanics of project configuration; this plug-in runs in WebSphere Integration Developer as well. Errors specific to builders, validators, and deployers will require knowledge of trace enablement by specific components. Many plug-ins use the eclipse logging facility, as above. Others use native java logging facilities; those logs are generally at the root of the worskpace/.metadata directory (for example, MedValidator.log)

Agenda

- serviceDeploy Overview
- serviceDeploy Command and Details
- Troubleshooting serviceDeploy
- Summary

This section will provide summary.

Summary

- Provides a command-line interface for building archives with service components into installable artifacts
- Part of an automated service component deploy and installation process

In summary, ServiceDeploy is a command-line interface for building archives with service components into installable artifacts. Used in conjunction with other command-line tools for preparing the archive and installing the application, serviceDeploy is an important part of an overall automated deploy and install process.

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.