CBS Secrets Unveiled – Understanding Broken and Dirty DCs

Applies to:

SAP NetWeaver development infrastructure (installed as usage type DI). This article is valid for SAP NetWeaver releases ’04 and 2004s.

Abstract

The SAP NetWeaver Development Infrastructure (NWDI) offers an automated build server (CBS) for Java-based development. This article helps developers and administrators to better understand the component build. With the CBS the build problems are typically reported as “broken” components. Reasons that lead to broken components are explained and solutions to this type of problems are shown.
Some basic experience with CBS and knowledge about SAP’s component model will help to understand it. For more information see NWDI Knowledge Center.

Author(s): Dr. Wolfram Kleis, Wolf Hengevoss

Company: SAP AG

Created on: 6 December 2006

Author Bio

Dr. Wolfram Kleis joined SAP in 1999 and worked on CRM Mobile Application Repository, Java Build Infrastructure and Java Component Model. Wolfram is a member of the NWDI development team.

Wolf Hengevoss graduated in Natural Sciences from the University of Kaiserslautern. In 1999, he joined SAP AG working in product management, starting in the Basis group. Since the early stages of SAP NetWeaver Exchange Infrastructure, he has been working on the Java environment. Today, Wolf's focus is on the rollout of SAP NetWeaver development infrastructure.

Table of Contents

Broken DCs Explained

From time to time, SAP development support receives problem messages from customers who are worried about "broken" components reported by the CBS (Component Build Service). The CBS comes as part of SAP NetWeaver Development Infrastructure (NWDI). This article explains what exactly a "broken" Development Component (DC) is in CBS and also gives some hints for build administrators and developers how to deal with broken DCs.

Broken DC != Broken NWDI

Assume you are responsible for the CBS of your development team. You come to your office in the morning, and - like every morning - you open the CBS Web UI to check the status of your buildspace. But what is this? There are 10 "broken" DCs! Broken? Is my NWDI blown up? Is it time to contact SAP support and create yet another problem message? No. Wait a moment!

Having "Broken DCs" *usually* does not mean that there is something wrong with the NWDI software. In many cases, these are just errors in your modifications or in your custom development. Think of a developer who writes a small Java program and runs Javac to compile it. If the compiler reports some errors, the developer would not immediately blame the compiler. He or she would first check if there are errors in the code or if the class path is not complete.

But what are broken DCs? How can this happen? There are two reasons why your DC may get broken:

- Your DC was activated forcefully even though the build failed during activation
- There is something wrong with another DC that is used by your DC. For example, there may be an incompatible change in a DC you are using (for example, a method was deleted from a Java interface of the used DC).

Broken after Forceful Activation

When you activate a change (the activity that contains the changed files to be more precise), the CBS analyzes the affected files and determines the DCs to which these files belong. These are the DCs that are directly modified by the change. CBS first tries to build these DCs. If the build fails, the activation request is rejected.

But sometimes you want to activate the change even if the build fails. This makes sense, for example, if you are absolutely sure that the build error is not caused by your change. It could be caused by a problem in a used component. If you activate your change "*by force*", it will be available in the active DTR workspace and, after the used DC is fixed, your component will be built automatically without any further action from your side. (DTR or *Design Time Repository* is the NWDI's WebDAV/DeltaV based source code versioning system. Changes are first done in a virtual location called the "inactive workspace". By activating an activity the changes are promoted into the active workspace, from where they can safely be used by others.)

If you activate a change to a DC even if build failed, the DC is marked by CBS as broken. This indicates that the latest build results on CBS (built before the change) do not match the current content of the active workspace in DTR. This is one way how a DC gets broken.

Broken after Automatic Rebuild

The second reason why a DC can get broken is because of a failing attempt to rebuild the DC after a *used* DC was changed. How can this be possible? Does not NWDI prevent activation of code that leads to build errors? Yes and no. NWDI prevents activation of code that cannot be compiled. But it does not prevent activation of code that breaks some other components that depend on your DC. If the modified DCs can be built successfully, the change gets activated.

Other DCs that use the modified DCs are rebuilt automatically. This is necessary in two cases:

- The using component assembles results of the modified DCs. If the content of the public part with purpose "assembly" has changed, the new results must be assembled.
- The using component depends on a public part with purpose "compilation", and the content of the public part was changed. The using DC must be rebuilt because input for some generators may have changed and in order to check if there are incompatible changes in interfaces published by the used DC.

If the using DC was rebuilt successfully, the CBS starts the next "wave" of rebuilding: It may be necessary to rebuild the using DCs of that DC as well. If the content of the public parts was not changed, the process stops here. If the content of the public parts *was* changed, the rebuild process continues with the next "wave". But this can only be decided after the first rebuild is complete.

Example: For public parts of type "assembly" the change in a Java DC could lead to the rebuild of a Web Module DC that assembles the JARs from the Java DC. The rebuild of the Web Module DC would trigger the rebuild of an Enterprise Application DC that assembles the Web archive.
The same can happen with public parts for compilation. Assume DC1 exposes some XML file in a public part. DC2 declares a dependency to this public part. In the content of DC2, the XML files exposed by DC1 are used as input for generating some Java classes. In DC2 these generated Java classes are exposed in a public part for compilation. The exposed classes are imported in Java classes in another component DC3. If the public part of DC1 is modified, CBS has to rebuild first DC2 and then DC3. This way, one change can trigger an avalanche of rebuilds.

Asynchronous Rebuilds and Dirty DCs

It is clear that for non-trivial projects the rebuilds can take a lot of time. The design goal was to make the build results of a change available as soon as possible, and to support parallel work in a team.

This could only be achieved by executing the rebuild of used components in an asynchronous way.

CBS activates the modified DCs immediately. It does not wait until all using components are rebuilt. During activation of the modified DC, it only schedules requests for rebuilding the using DCs. DCs that are scheduled for rebuild are called *dirty* DCs. For dirty DCs, CBS creates *internal* build requests. These requests are put into the request queue together with those activation requests that are created by the developers. The dirty DCs are rebuilt later when CBS fetches the internal build request from the queue and executes it.

Note that the number of dirty DCs may increase during rebuild. This happens, for example, if the 2nd level of using DCs contains more DCs than the first level. In this case, the "2nd wave" will show an increased number of dirty DCs. Figure 1a and 1b show an example:

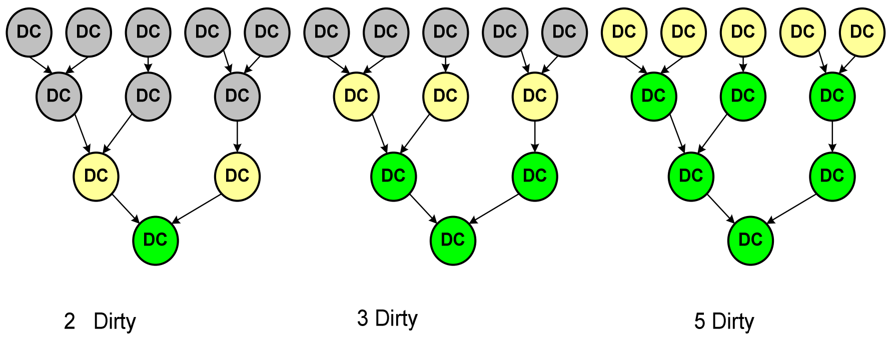
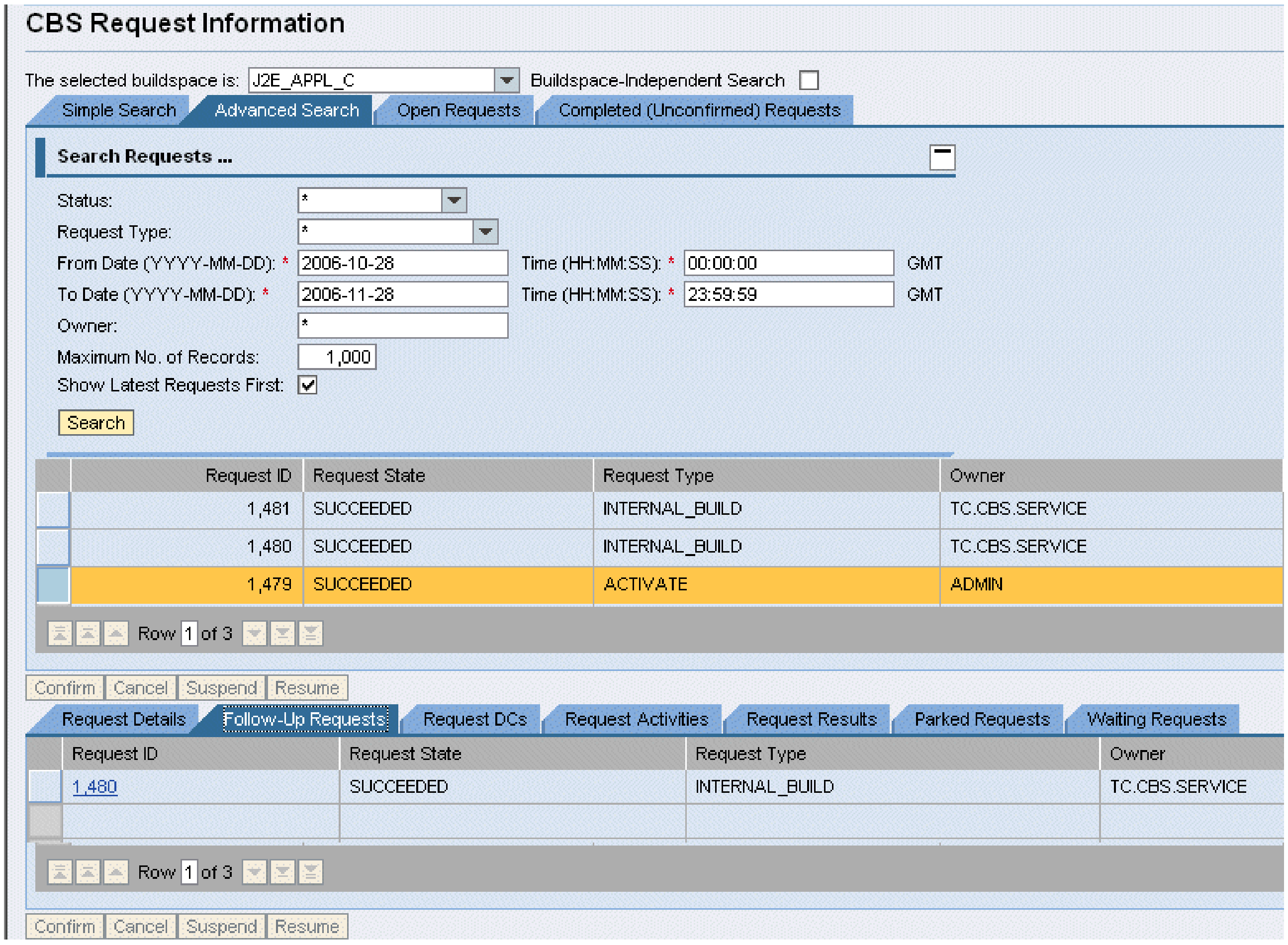


Figure 1a: Rebuild of DCs "in waves"

By the way: Understanding the rebuild mechanism of CBS also shows how you can improve build performance: Always create dependencies to specific public parts and not to the whole DC. If you create a dependency to the whole DC, your DC is rebuilt whenever any of the public parts of the used DC was changed. Unnecessary rebuilds are avoided if you create dependencies only to those public parts you really need. It may also be a good idea to create multiple public parts for different functionalities offered by your DC, so the users of your DC can select the public part with the functionality they actually need.

You can look for *follow-up requests* using the request details view in the CBS Web UI:


CBS Request Information
Simple Search | Advanced Search | Open Requests | Completed (Unconfirmed) Requests
Search Requests ...
Search
Confirm | Cancel | Suspend | Resume
Request Details | Follow-Up Requests | Request DCs | Request Activities | Request Results | Parked Requests | Waiting Requests

The selected buildspace is: J2E_APPL_C
Buildspace-Independent Search: ☐

Status:	*
Request Type:	*
From Date (YYYY-MM-DD): *	2006-10-28	Time (HH:MM:SS): * 00:00:00 GMT
To Date (YYYY-MM-DD): *	2006-11-28	Time (HH:MM:SS): * 23:59:59 GMT
Owner:	*
Maximum No. of Records:	1,000
Show Latest Requests First:	✓

Request ID	Request State	Request Type	Owner
1,481	SUCCEEDED	INTERNAL_BUILD	TC.CBS.SERVICE
1,480	SUCCEEDED	INTERNAL_BUILD	TC.CBS.SERVICE
1,479	SUCCEEDED	ACTIVATE	ADMIN

Row 1 of 3

Request ID	Request State	Request Type	Owner
1,480	SUCCEEDED	INTERNAL_BUILD	TC.CBS.SERVICE

Row 1 of 3

Figure 1b: Follow-Up Request in CBS: Requests no. 1480 and 1481 were initiated by CBS itself as follow-up requests of no. 1479.

And What if the Rebuild Fails?

The consequence of the asynchronous rebuild is this: Rebuilds of using components may fail long after the modification of the used DC was activated. And if this happens, what should CBS do in such a case? Assume that you change your DC "my/library" and the activation succeeds. Fifteen minutes later, CBS tries to rebuild DC "my/bean" which uses "my/library". But you made an incompatible change to some Java interface in "my/library" and so the build of "my/bean" fails. CBS cannot reject the change anymore because it is already active. However, it must somehow signal that there is a problem. The solution: CBS marks DC "my/bean" as broken. It is broken because it should have been rebuilt but the attempt failed.

One important guideline that helps to avoid broken DCs is to be careful with changes to public parts. It is a good idea to establish some development rules like "only compatible changes are allowed in public parts".

Propagation of "Broken" State

If component "my/bean" was rebuilt successfully, the next step would be to rebuild component "my/application" which assembles "my/bean". But this makes no sense if the rebuild of "my/bean" failed and "my/bean" is broken. Instead of rebuilding "my/application", the CBS marks "my/application" also as broken.

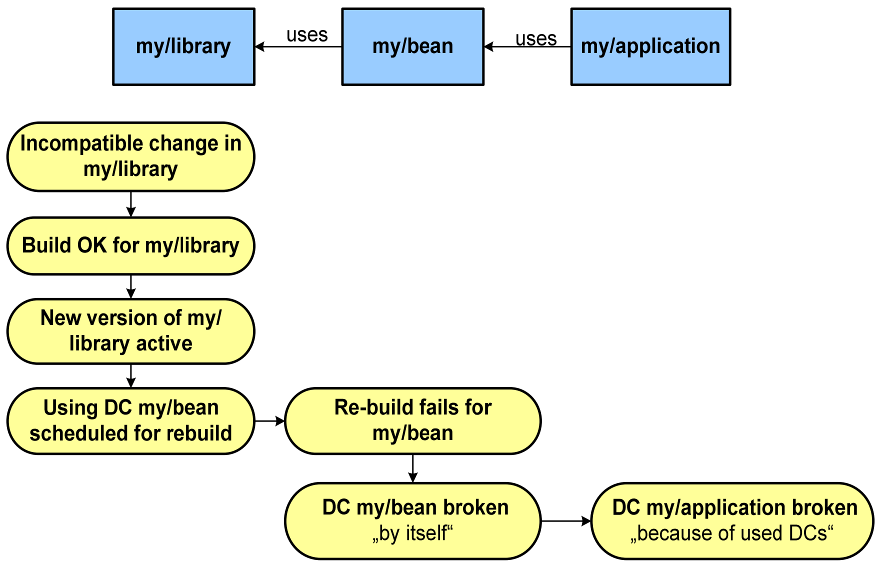

my/library
uses
my/bean
uses
my/application
Incompatible change in my/library
Build OK for my/library
New version of my/ library active
Using DC my/bean scheduled for rebuild
Re-build fails for my/bean
DC my/bean broken „by itself"
DC my/application broken „because of used DCs"

Figure 2: Broken used DCs and propagation of "brokenness"

The example shows that there are two types of broken DCs:

- DCs which are broken because the build failed for this DC. In CBS Web UI these DCs are named "broken by itself". DC "my/bean" is an example.

- DCs which are broken because a used component is broken. Brokenness was propagated without even trying to build this DC. In CBS Web UI these DCs are named "broken because of used DCs". DC "my/application" is an example.

How to Analyze and Fix Broken DCs

The Strategy

Now you understand what broken DCs are and how the broken state is propagated and how easily one change can cause dozens of DCs to be broken. The good news is: One fix can repair dozens of broken

DCs. This idea leads to the strategy for fixing broken DCs:

1. Use the CBS Web UI to find out which DCs are “broken by itself “ (To find it in a browser, type *http://<host:port of CBS server>/webdynpro/dispatcher/sap.com/tc.CBS.WebUI/WebUI* or use the overview page type *http://<host:port of CBS server>/devinf* to navigate to the CBS.)
    a. Locate your build space.
    b. Open the “compartments” view.
    c. Select one compartment. At the bottom of the page, you see a list of DCs. There you can filter for DCs that are “broken by itself”.
    d. If there are DCs “broken by itself” find out why these DCs are broken and fix them
    e. Repeat for other compartments.

2. After the “broken by itself” DCs are fixed, CBS tries to rebuild the DCs using them again. If you are lucky, everything is green now. But not necessarily. Of course the rebuild could reveal new errors (caused by incompatibilities etc). If this happens for some DC its state changes from “broken because of used DC” to “broken by itself”. And you continue investigation with that DC…

Analyzing a Single DC “broken by itself”

If a DC is broken “by itself”, there are two alternatives:

- The DC is broken because of DC problems with meta data that are checked by CBS before the actual build is started. In this case, there is no build log and only the request log exists. The request log contains the details about the error. You find a link to the request log in the CBS Web UI in the DC details page. (see SAP Help Portal →CBS Web UI: CBS Development Components DC Details). Examples for typical errors that appear in request log are:
    - A used DC or public part does not exist
    - Using and used DC are in different SC compartments but there is no usage dependency between the two SCs
    - The “black box” rule for child DCs was violated. A DC tries to use the child DC of some other DC, which is not allowed.

- The DC is broken because of build errors. Here the CBS actually started the DC build, which is executed by the DC build tool in a separate VM. The build failed and you should see the error messages in the build log. The build log is created for each DC separately. It looks very much like a normal ANT build log. You find a link to the build log in the CBS Web UI in the DC details page.

  Examples for errors in the build log are:

    - The used DC is protected by an access control list.
      Example: Someone changed the ACL for a used DC in an incompatible way.

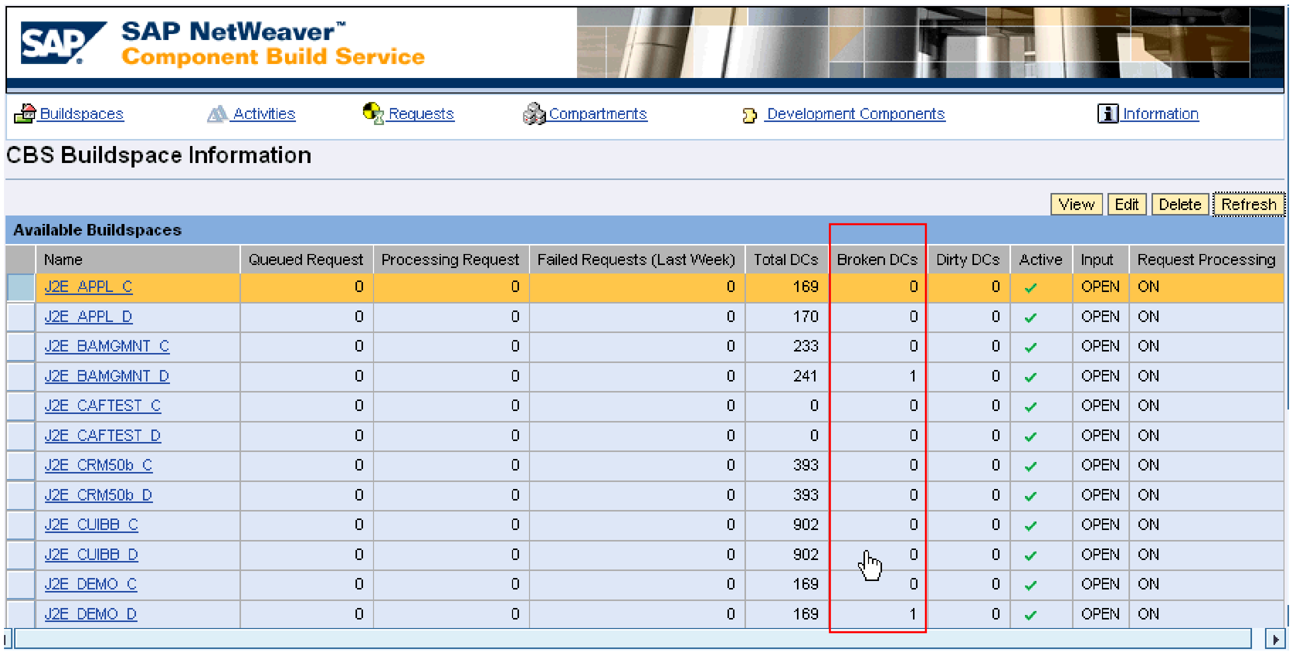
    - Errors reported by Java compiler (class missing, method not found and so on).
      Example: Someone removed a class from a public part, removed a method and so on.

    - Error reported by generators like Web Dynpro generator.
      Example: Some Web Dynpro component interface changed in incompatible way.

    - Technical problems with generators, packaging steps and so on.
      In this case there should be an error text or stack trace in the build log.

Example: Analyzing a Single DC “broken due to its MetaData”

Use the CBS Web UI to find broken DCs. In our example, we assume the public part of a Web Module DC has been deleted. The Enterprise Application DC using this public part is in state “broken” after the automatic rebuild.

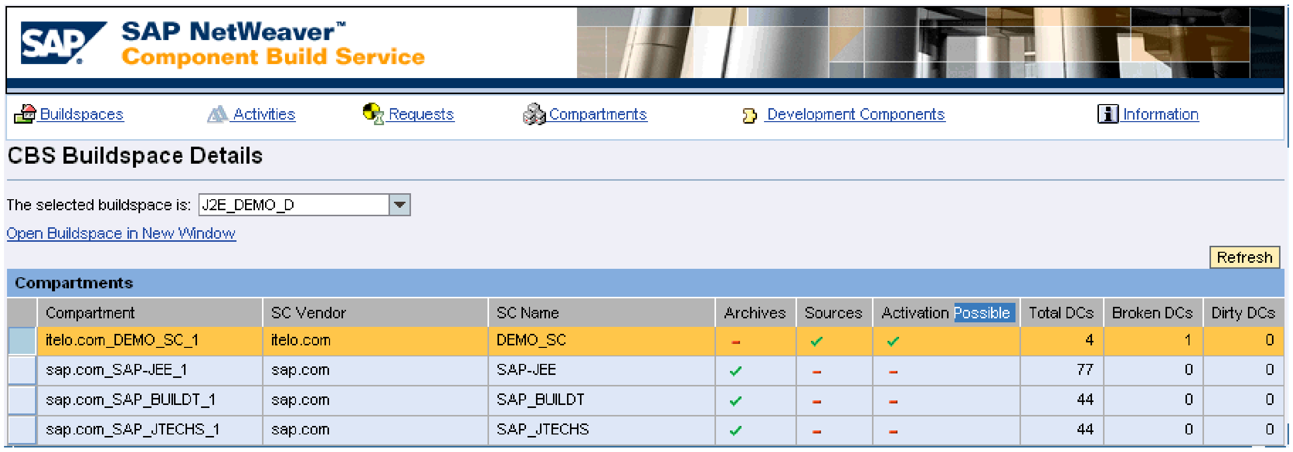


CBS Buildspace Information

Available Buildspaces

Name	Queued Request	Processing Request	Failed Requests (Last Week)	Total DCs	Broken DCs	Dirty DCs	Active	Input	Request Processing
J2E_APPL_C	0	0	0	169	0	0	✓	OPEN	ON
J2E_APPL_D	0	0	0	170	0	0	✓	OPEN	ON
J2E_BAMGMNT_C	0	0	0	233	0	0	✓	OPEN	ON
J2E_BAMGMNT_D	0	0	0	241	1	0	✓	OPEN	ON
J2E_CAFTEST_C	0	0	0	0	0	0	✓	OPEN	ON
J2E_CAFTEST_D	0	0	0	0	0	0	✓	OPEN	ON
J2E_CRM50b_C	0	0	0	393	0	0	✓	OPEN	ON
J2E_CRM50b_D	0	0	0	393	0	0	✓	OPEN	ON
J2E_CUIBB_C	0	0	0	902	0	0	✓	OPEN	ON
J2E_CUIBB_D	0	0	0	902	0	0	✓	OPEN	ON
J2E_DEMO_C	0	0	0	169	0	0	✓	OPEN	ON
J2E_DEMO_D	0	0	0	169	1	0	✓	OPEN	ON

Figure 3a: Buildspaces Overview. You can check the state of a complete buildspace



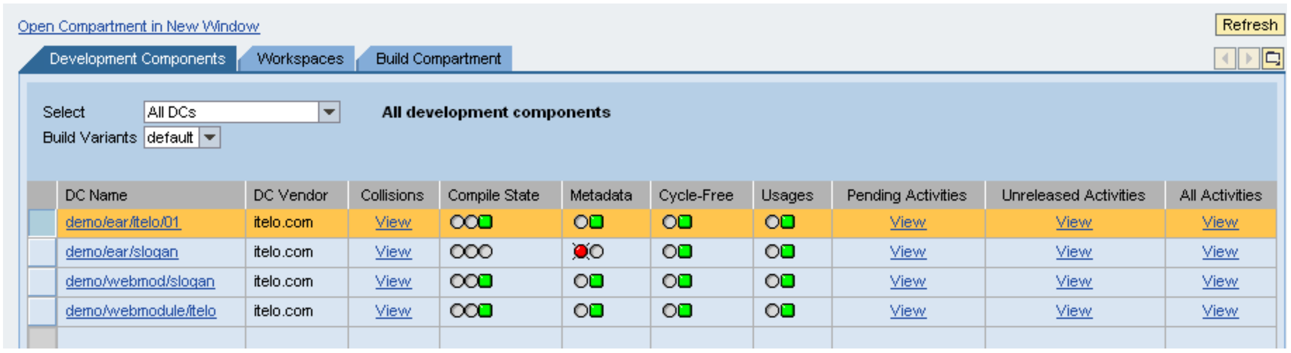
CBS Buildspace Details

The selected buildspace is: J2E_DEMO_D

Compartments

Compartment	SC Vendor	SC Name	Archives	Sources	Activation Possible	Total DCs	Broken DCs	Dirty DCs
itelo.com_DEMO_SC_1	itelo.com	DEMO_SC	–	✓	✓	4	1	0
sap.com_SAP-JEE_1	sap.com	SAP-JEE	✓	–	–	77	0	0
sap.com_SAP_BUILDT_1	sap.com	SAP_BUILDT	✓	–	–	44	0	0
sap.com_SAP_JTECHS_1	sap.com	SAP_JTECHS	✓	–	–	44	0	0

Figure 3b: Buildspaces details.



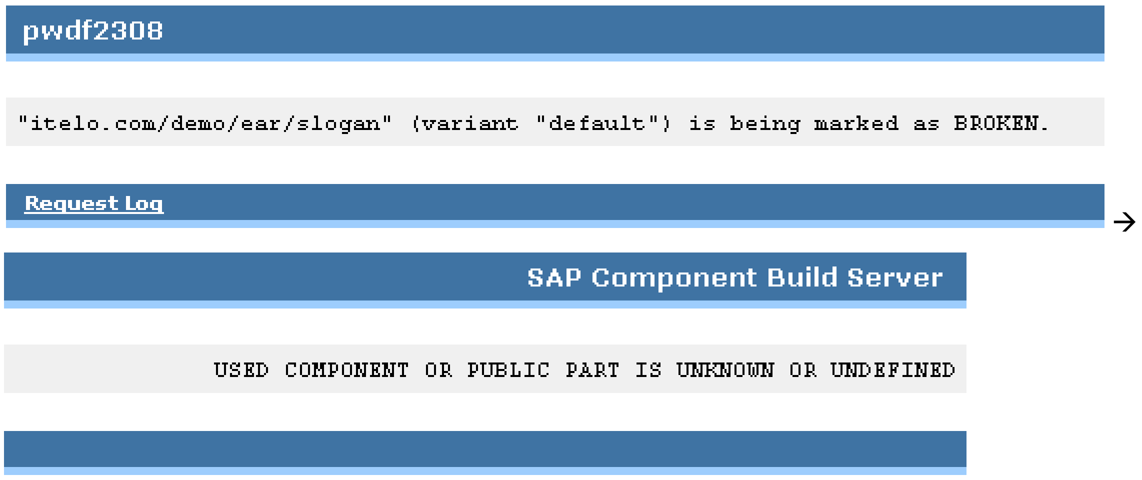
Select All DCs

Build Variants default

All development components

DC Name	DC Vendor	Collisions	Compile State	Metadata	Cycle-Free	Usages	Pending Activities	Unreleased Activities	All Activities
demo/ear/itelo/01	itelo.com	View	○○🟩	○🟩	○🟩	○🟩	View	View	View
demo/ear/slogan	itelo.com	View	○○○	🔴○	○🟩	○🟩	View	View	View
demo/webmod/slogan	itelo.com	View	○○🟩	○🟩	○🟩	○🟩	View	View	View
demo/webmodule/itelo	itelo.com	View	○○🟩	○🟩	○🟩	○🟩	View	View	View

Figure 3c: Broken DC.



```
pwdf2308

"itelo.com/demo/ear/slogan" (variant "default") is being marked as BROKEN.

Request Log

SAP Component Build Server

USED COMPONENT OR PUBLIC PART IS UNKNOWN OR UNDEFINED
```

Figure 3d: Build log for broken DC.

How to Repair Broken DCs

There are several ways to repair a broken DC, depending on the reason why it got broken:

- **Activate fixed version of broken DC**
  If the DC itself contains the error you should check out the affected files from DTR, fix the problem in Developer Studio and then check-in and activate the new version. As a result of the activation the DC will directly go from "broken" state to "OK" state.
  Example: As announced before a deprecated interface was deleted by the provider. You change your code and use a new interface that was specified as an alternative.

- **Activate/Import fixed version of used DC**
  If the DC was broken because of a problem in a used DC or build plug-in, the solution is to fix the used DC. This can be done by editing and activating a new version of the used DC (if it is available with its sources) or by importing a patched version of the used SC into CMS (if it is available as archives). In both cases, the broken DC will become dirty first (scheduled for rebuild). After successful rebuild, the DC will get the "OK" state.
  Example: The owner of some DC has accidentally deleted a public part you are using. This DC was built somewhere else and it was imported into your NWDI track as part of some used software component. After you reported the problem, the owner of the DC has provided a fixed version. Your NWDI administrator imports the software component archive with the fixed version into your track. CBS automatically rebuilds all dependent components including your DC. Because the used public part is now found, your DC can be built successfully and is no longer broken.

- **Administrative Rebuild**
  Sometimes a DC gets broken if a rebuild fails because of resource problems. In these cases, it is enough to fix the resource problem and then trigger a rebuild. Therefore, it is not necessary to change anything in the DCs themselves.
  Example: There is not enough memory or the disk is full in the CBS server for some time. After the server administrator has fixed the resource problem, NWDI administrators can use CBS Web UI to rebuild individual DCs or whole SC compartments. After the successful rebuild, the DCs will go directly from broken to OK state.

- **Initialize Compartment**

  In some rare cases internal caches of the CBS may be out of sync with the DTR.
  Example: Think of a situation where you have no broken DCs or any pending requests but you have a mismatch between the content of the files in the active workspace of DTR and the DC meta data or

build results on CBS. Maybe the content of DTR had to be restored from a backup that does not contain the latest changes.

NWDI administrators can use the “initialize compartment” operation to repair this kind of inconsistency. When initializing a compartment, the CBS discards all the build results and the cached source code for all the DCs in that compartment. The compartment is now empty on CBS and must be filled again from scratch. All source files will be fetched from DTR again and all DCs must be rebuilt. For a compartment that contains several hundred DCs this may take hours. So this feature should only be used if you have strong reason to believe that there is some inconsistency, which can be cured by re-initialization.

Related Content

Please include at least three references to SDN documents or web pages.

- NWDI Knowledge Center.

Copyright