

Package Dependencies Analyzer



User Guide

| | |
|----------------------|---|
| Project | ISPDA |
| Document File | WiaPackageDependencies UG (EN) v1.2.doc |
| Version | 1.2 |
| Version Date | 29-JUL-2016 |
| Status | FINAL |
| Author | António Abreu |

1 INTRODUCTION

The **Package Dependency Analyzer** is a tool to analyze existing and missing dependency declarations of **Integration Server Packages**.

It is intended to answer the following questions:

- If I install these packages at an **Integration Server**, which packages do I also have to have installed?
- Which packages can I safely remove from the **Integration Server** because no other package needs them?

Indirectly, the analysis of the package dependencies also uncovers nodes (*services, documents, triggers, etc.*) and **DSPs** that have references to nodes that do not exist in the server¹.

1.1 Document Objectives

This document serves the objective of providing a **User Guide** to the tool, currently in version **1.7**.

The tool is continuously under development² and therefore it is certain that this document will change in the future. The core analyzer is quite complete, but the graphical interfaces are a long way from completion:

- There are no setting or analysis options available;
- The analysis report could be completely reworked.

1.2 Installation

Just install the `WiaPackageDependencies IS` package through the **Integration Server** package management page.

Mind that additional packages need to be installed beforehand, in order:

- `WiaRoot v1.2`;
- `WiaUtilities v1.12`;
- `WiaServiceProfiler`, on any version, with a valid license key.

1.3 Access the analysis functionality

Access the package functionality through the package's home page and following the link(s) defined in there (*see [Figure 3](#), in page 3*).

¹ This may be due to missing packages or packages that have been disabled. If the dependencies are declared, upon installation the **Integration Server** will validate the dependency, issue error message and abort the installation. Package disabling and deleting is also guarded by the dependency declarations.

² ... because we actively use it in our projects and customers.



Figure 1 - Go to the package home folder

A direct access link to the tool is made available in the **Service Profiler** menu if you have installed **v1.3.4** or above.



Figure 2 - The tool at Service Profiler's menu

At the `WiaPackageDependencies` package homepage, there is a link to the analysis functionality, and some useful information about the tool.

1.3.1 Generating Reports from Analysis at Remote Servers

The dependency analysis can only be performed at the server where the **Package Dependencies Analyzer** is installed.

However, if the tool is installed at all required servers and these are accessible in the network the analysis can be submitted and viewed from a single **Integration Server**.

As stated in the tools homepage, by defining **Remotes Aliases** with names suffixed with `_WIA_PKG_ANALYZER`, the tool automatically enables remote analysis on those servers. If such configuration exists, upon trying to access the functionality a server selection panel is presented first.

That panel lists all the defined remote aliases stripped of the name suffix and identified as **Logical Name**. These names are as descriptive as the definitions you created as **Remote Alias**.

1.3.2 Do a Package Dependencies Analysis and interpret the results

A package dependency analysis starts by selecting the package(s) to analyze (*see Figure 4, in page 3*): these are the input packages, and will appear listed in the analysis report.

There are more possible options, but these are not currently settable through this user interface.

All packages are listed and are present at any time. The **Regular Expression** filters the list to facilitate the selection. You can apply any number of **Regular Expressions** to select or unselect package for the report before submitting. Before submitting your selection to generate the report, you can check the complete list of packages you have selected by clicking the link [Show all selected packages](#).

Purpose

This package serves the purpose of extending the **webMethods Admin Page** package management. It implements a tool for **analysis & report of package dependencies**.

The **analysis & report of package dependencies** tool does more than just report the dependencies. It

- Discriminates dependencies that are defined by **design**, from those only detectable in **runtime**. To have runtime dependencies analyzed, it is required to have the **Service Profiler** installed, an **Design-time (D)** dependencies are those that are not declared in the package definition, but between the nodes.

Figure 3 - Go to the package analysis tool's page

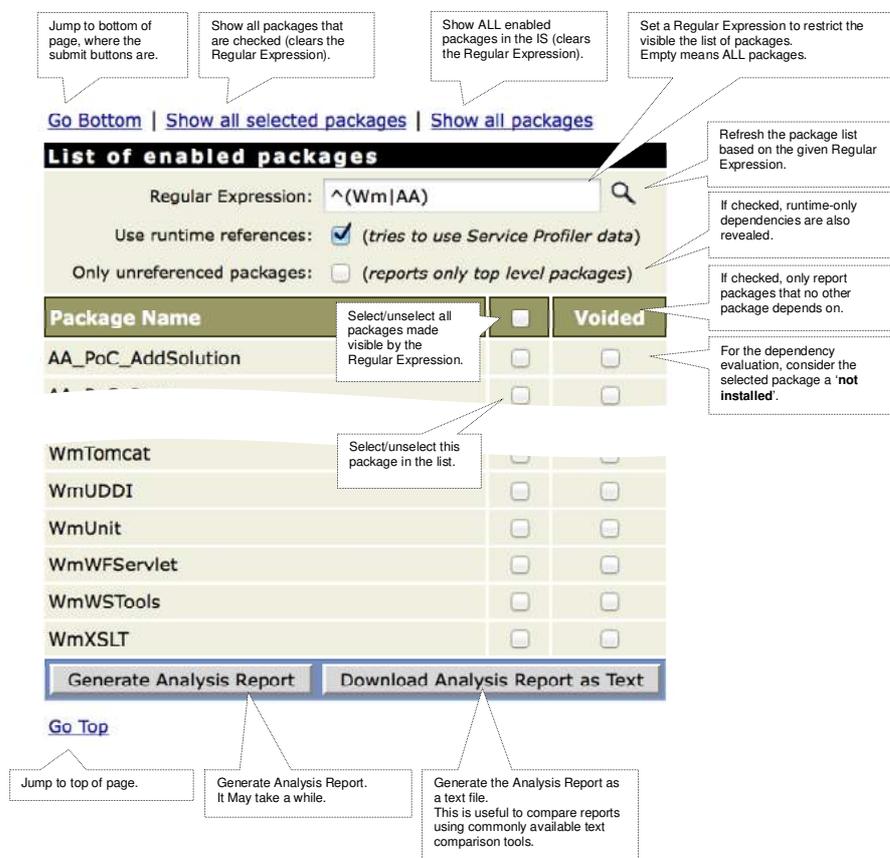


Figure 4 - Give input for the package dependencies analysis

The report may, and most certainly will, present more packages than those provided as input. This is because the internal analyzer option **Show All Scanned Packages** is currently being always used as `true`. During the analysis process, the tool will simulate a package loading situation checking every loaded package for references into other packages and loading them in turn, until all references are resolved. This behavior in analysis allows its use in a practical case:

- Select only one (or a very strict list of) package(s).

The result is a list of all packages the selected package(s) required to completely fulfill its (their) functionality.

If there are unresolved nodes, are they may be missing because:

1. Their package is not installed?
2. Their package is currently disabled?
3. Those nodes have simply not been deployed in the required package version³?

Because the report can be very extensive, when it is presented, only a list of headers identifying the packages is visible. The package analysis details are collapsed, but the header already provides very useful information (*see Figure 5, below*).

The **Use runtime references** option is checked by default. However, it may report more dependencies and circular references than normal, because some of those dependencies and references are either intentional or unavoidable. Checking or not this option is a matter of experimentation, environment, personal experience, etc. For instance, in one occasion an entire **Integration Server** was migrated to a newer version by installing the new platform from scratch and installing-and-copying everything from the older server to the new. Everything was equal and worked. However, every time the server was restarted there was one package that always reported an error on the startup sequence. The problem was caused by the fact that:

- The package load order is not guaranteed as being the same on disparate servers;
- The startup sequence called a service that called a service that was at a package that was not yet loaded.

Because the calling service was a **Java Service**, design time dependencies were not revealed. But by checking the **Use runtime references** option this dependency was detected and reported; and it was just a matter of declaring the missing dependency to guarantee the correct package load order and make the error disappear.

³ This may happen when a certain node is removed, but the references were not automatically updated or removed by the developer.

• [Go Home](#)

Expand all package analysis details.

Collapses all package analysis details back to only the headers.

Jumps to the bottom of the page.

Reveals a report summary only with counters and pertinent lists.

[Expand All](#) | [Collapse All](#) | [Go Bottom](#) | [Show Report Summary](#)

| | |
|-----------------------------------|------|
| AA_PoC_ACLS v1.0 | D |
| AA_PoC_AddSolution v1.0 | D |
| AA_PoC_Docs v1.0 | D |
| AA_PoC_ErrorHandling v1.0 | D |
| AA_PoC_ErrorHandlingStrategy v1.0 | DU |
| AA_PoC_JavaServicesDev v1.0 | |
| AA_PoC_ModelReprocessing v1.0 | DU |
| AA_Test_WebServices v1.1 | |
| AA_Tests v1.0 | A DU |
| AA_Tests_EmptyFolder v1.0 | |
| AA_Tests_JDBC v1.0 | A |
| AA_WebServices v1.0 | |
| WmWSTools v6.0.1 | |
| WmXSLT v6.0.1 | |

[Save Changes](#)

[Go Top](#)

Jumps to the top of the page.

If any missing declaration has been selected for update, this option will store it in the package. The analysis is resubmitted and an updated report is presented. The saved changes cannot be undone through this tool.

Package state summary:

- D:** Undeclared dependencies have been found.
- C:** Circular references have been found.
- U:** References to unresolved/undefined nodes have been found.

This package is completely defined, i.e., has no missing declaration of dependencies, etc.

Click on this header bar to show/hide the details about this package.

Packages marked with **A** are found to have direct dependencies from an Adapter Package.

Figure 5 - Package Dependencies Analysis report

| Package Name | Kind | Version | Buttons |
|--|-------------|---------|--------------------------|
| AA_Test_VfProperties v1.0 | DU | | |
| AA_Test_VfProperties2 v1.0 | D | | |
| AA_Test_WebServices v1.1 | U | | |
| AA_Tests v1.0 | A DU | | |
| (D) AA_PoC_PipelineSize | | 1.0 | <input type="checkbox"/> |
| (D) BawTestFrameworkByWia | | 1.0.10 | <input type="checkbox"/> |
| (D) SampleFileUpload | | 1.0 | <input type="checkbox"/> |
| (D) WiaChronometer | | 1.0 | <input type="checkbox"/> |
| (D) WiaUtilities | | 1.9.1 | <input type="checkbox"/> |
| WmART | | ** | |
| (D) WmBrokerAdmin | | 6.1 | <input type="checkbox"/> |
| WmJDBCAdapter | A | 6.0.3.1 | |
| WmPublic | | ** | |
| WmRoot | | ** | |
| Referenced by these packages | | | |
| AA_PoC_SubscribeDoc | | | |
| AA_ServiceProfiler_Regression | | | |
| References to undefined nodes | | | |
| (U) (AdapterType)WmA4AbreuantTest | | | |
| referenced by: amabr.test.connection:ABREUANT | | | |
| (U) ir.pub.util.flow:getThisServiceName | | | |
| referenced by: amabr.test.service:loopCallingService | | | |
| (U) IRShared_RMSResource.docs:brokerStats | | | |
| referenced by: amabr.test.broker.info:getBroker | | | |
| AA_Tests_EmptyFolder v1.0 | | | |
| AA_Tests_JDBC v1.0 | A | | |
| AA_WorkflowSample v1.0 | | | |

In parenthesis, the kind of analysis that found the dependency:

- (D) Design-time;
- (R) Runtime.

Dependencies not declared in the package, but determined during the analysis. Checking it causes the declaration to be created when the [save changes] button is pressed. The package version can also be edited.

Packages marked with an **A**, are **Adapter Packages**.

Dependencies already declared in the package, which cannot be changed, through this tool.

A list of packages that depend on this one. If listed as link, click it to jump to its details in this the current report (causes all details to be collapsed with exception to the target one).

A list of unresolved nodes referenced within the package, with its name and the name of the node that tries to reference it.

Figure 6 - Report detail on declared dependencies, found dependencies and unresolved references

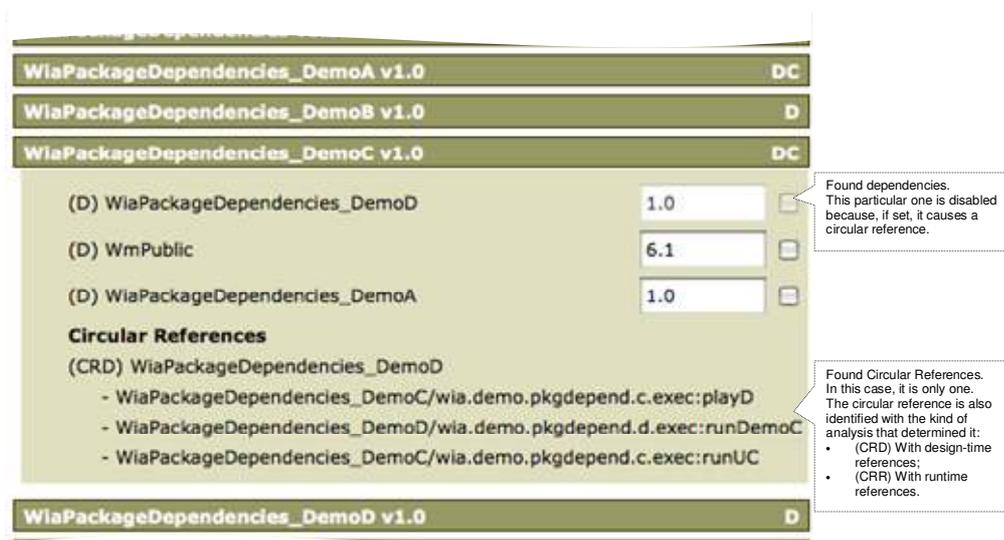


Figure 7 - Details on Circular References reporting

1.4 Known issues & Words of caution

There are some already known situations and they will be addressed in the next versions, either with refined functionality or/and increased validation and protection:

- The package analysis does a package scanning that tries to simulate the **Integration Server** loading sequence but that goes beyond it and that cannot guarantee the same loading sequence. The packages are loaded in the order they are fed as input, *i.e.*, alphabetical order. The issue is that once a package is scanned it is thus considered loaded. To avoid infinite recursive loading, this causes the detection of circular references to be non-reflexive, *i.e.*, if a non-declared circular reference is found from package A into E, it should be reported also a circular reference from E into A, but it doesn't. This is something that is being evaluated, because it may indicate that saving some dependencies is **OK** and it isn't... because a circular reference could be generated.
- Because of the issue above, be very careful when checking the creation of a declared dependency. There may be some cases where the missing declaration is known, intentional and justified. *E.g.*, the `WiaPackageDependencies` is missing a dependency on the `WiaServiceProfiler` package, but it is intentional: the service code will use the **Service Profile** services, but only if they are available... no error will be generated. However, there may be some found dependencies that are quite straightforward, and that can safely be set.
- Some of those straightforward dependencies will be, for instance, on the `WmPublic` and `WmRoot` packages.

It is usually assumed as implicit by developers that these dependencies are not required to be declared.

Ideally, all dependencies should be declared, because that rules and improves the startup and shutdown times. However, this is not supported by the **Integration Server** on a very practical

manner⁴. So, only a subset of situations require having the dependencies to be declared as mandatory:

- ▶ Always declare dependency from a package containing services that are being called by services in this package's startup or shutdown sequence;
Failing to do this may cause errors on **Integration Server** startup or shutdown.
- ▶ On packages that contain **Adapter Connections**, always declare dependency from the adapter package;
This is clearly stated in any of the adapters **User Guide** and even in the **ADK** documentation. This is intended to avoid instability at the **Integration Server** due to internal management of objects by the adapter itself: if the dependency is not declared, when the adapter releases global object, rebuilds pool, *etc*; the dependent packages should reload and they do not... because they will be referencing object that are no longer valid, this may incur in a surge of adapter and service errors... from which the server may recover or not (*sometimes a server restart is the only solution*). Declaring the dependency is sometimes enough to eradicate all these situations.
- ▶ On packages that contain **Adapter Services**, always declare dependency from the package that contain the **Adapter Connection**;
- ▶ On packages that contain **Adapter Listeners**, always declare dependency from the package that contain the **Adapter Connection**;
- ▶ Packages that publish a **Publishable Document** to Broker that is defined at another package should declare a dependency from the package containing that document definition (just to make sure it exists) because the **Dispatcher** requires its presence to validate the signature against its representation at the broker.

However, the dependency is not actually mandatory. But you have to be sure it exists when a publishing of that document happens, or a runtime error will occur.

⁴ The problem is that by having a dependency declared automatically it means that when a package is reloaded **all** depending packages are also reloaded and you can only disable or delete packages that others do not depend on. If all dependencies were declared it would mean that the installation of a simple package patch could force the reload of (almost) all packages on the **Integration Server** and that could take a very long time. The dependencies should be classifiable in a way that indicates whether a reload is implied or not... but they are a *one-size-fits-all*.

APPENDIX A GLOSSARY

| Item | Definition |
|------|--------------------------|
| ADK | Adapter Development Kit. |
| | |