Aras Innovator 12

Package Import Export Utilities

Document #: 12.0.02019053801

Last Modified: *1/3/2020*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810

Phone: 978-806-9400
Fax: 978-794-9826

E-mail: Support@aras.com

Website: https://www.aras.com

Table of Contents

Send Us Your Comments

Aras Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for future revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where and what level of detail?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, indicate the document title, and the chapter, section, and page number (if available).

You can send comments to us in the following ways:

Email:
Support@aras.com
Subject: Aras Innovator Documentation

Or,

Postal service:
Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Attention: Aras Innovator Documentation

Or,

FAX:
978-794-9826
Attn: Aras Innovator Documentation

If you would like a reply, provide your name, email address, address, and telephone number.

If you have usage issues with the software, visit https://www.aras.com/support/

Document Conventions

The following table highlights the document conventions used in the document:

Table 1: Document Conventions

Convention	Description
Bold	This shows the names of menu items, dialog boxes, dialog box elements, and commands. Example: Click **OK**.
`Code`	Code examples appear in `courier` text. It may represent text you type or data you read.
`Yellow highlight`	Code with yellow highlight is used to draw attention to the code that is being indicated in the content.
`Yellow highlight with red text`	Red color text with yellow highlight is used to indicate the code parameter that needs to be changed or replaced.
Italics	Reference to other documents.
Note:	Notes contain additional useful information.
Warning	Warning contains important information. Pay special attention to information highlighted this way.
Successive menu choices	Successive menu choices may appear with a greater than sign (-->) between the items that you will select consecutively. Example: Navigate to **File** --> **Save** --> **OK**.

1 Overview

The main implementation method for a new functionality in Aras Innovator is creation of a set of items that encapsulate the desired functionality - some items represent business objects (ItemTypes and their instances), some support user interaction (Forms, etc.), some may implement business logic (Methods), etc. Because all Aras Innovator items are stored in the database it creates numerous problems with identifying differences between different installations of Aras Innovator, keeping track of them, merging the differences, and upgrading to new releases. In order to implement a better mechanism for solving these problems, Aras Corporation has created a set of tools called 'Package Import Export Utilities', whose purpose is for it import, export and management of AML packages. Outlined here are some of the main ideas that this set of tools based on.

First, the idea of representing each item in a form of a separate AML files, which in its turn would allow:

- The use of a visual comparison between different versions of the same AML file.
- The use of third party visual merge tools for merging differences between different versions of the same AML file.
- Keeping custom solutions or any set of related items including those that implement core Aras Innovator functionality in XML form in a file structure and keep track of their changes.

Second, the idea of organizing a set of logically related items into a package.

Together these concepts allow simplifying the process of importing and exporting Aras Innovator core, BRS's, and custom solutions to and from the database. This allows administrators to keep track of modifications, merge differences between packages, and migrate these changes between databases.

The main goals for Package Import Export Utilities set are:

- The ability to create and modify AML packages in the database.
- The ability to export the some or all of the components of an AML package from a database to the file system in a form of a hierarchal set of AML files.
- The ability to import a hierarchy of AML files that represent a package to a database.
- If the package already exists in the database, the import process must provide an ability to automatically merge the differences between items into the database from corresponding imported AML.

2 Data Model

Conceptually each package is a collection of item IDs. There are a certain ItemTypes that were introduces to support package functionality. The following represents the common set of items that defines an AML package:

- `PackageElement` – The Package element represents ID number of the item that has been defined in the database.
- `PackageGroup` – This represents the type of ItemType (Method, List, etc.) that the Package Elements are added from. This ItemType also defines the name of the folder the Package Element is exported to in the file structure.
- `PackageDefinition` – This is the package itself. It represents the collection of package groups that makes up a package, and allow for the grouping of one package separate from the next.
- `PackageDependsOn` – This is a relationship type that establish dependencies between packages
- `PackageReferencedElement` – This is a relationship on `PackageDependsOn` type that defines what exact package elements from the 'related' package the 'source' package references. This could be useful when packages are exported (check-box '`Export Referenced Items`' in Export tool; see section Export Tool for more details).

Note: No Package Element may belong to more than one AML package. This is to prevent conflicts when importing the packages, if these two elements are not identical in each package. The import would have no way of knowing which AML was the correct AML to apply otherwise.

A newly installed Aras Innovator database contains Package Definitions of two types:

- **Core Packages** – These packages are used to define the basic structure of every Aras Innovator database, regardless of what solutions are used in the database.
- **Solution Packages** – These packages define the elements that comprise the definition and functional rules of different BRS's data models.

3 Using Package Import Export Utilities

3.1 AML Packages

The first step in understanding the use of the Package Import Export Utilities is to understand the structure of package AML files on disk and the corresponding manifest file.

3.1.1 The File Structure

The folder structure of a core package can be defined by careful use of the Package Definition name. Let's use the core Dashboard package as an example. Note that fully qualified package name of this Package Definitions is com.aras.innovator.dashboard. When exported, this package is exported to a hierarchal structure as such:

Innovator/
  Imports/
    Com/
      Aras/
        Innovator/
          Dashboard/

Any new packages are treated as such, and allow for the export of packages in this manner.

Non-core packages do not use this same rule for export despite the fact that solution package have a fully qualified names. I.e. this name is not translated into a directory hierarchy of 'com\aras\...' Instead, the three solution packages always export to the predefined folders:

Solutions/
  PLM/
    Import/
  Project/
    Import/
  QP/
    Import/

3.1.2 The Manifest File

The manifest file contains information about what packages can be processed by the utilities, dependencies between packages and where to find package's AML files.

Here is an example of a manifest file:

```
<imports>
   <package name="com.aras.innovator.solution.PLM" path="PLM\import" />
   <package name="com.aras.innovator.solution.QP"  path="QP\import" >
       <dependson name="com.aras.innovator.solution.PLM" />
   </package>
   <package name="com.aras.innovator.solution.Project"
path="Project\import">
       <dependson name="com.aras.innovator.solution.PLM" />
```

```
   </package>
</imports>
```

package tag

Attribute `name` of `package` – unique fully qualified name of the package

Attribute `path` of `package` – path to the directory that contains the package AML files.

Note: Path to the package's AML files could be either absolute or relative; in case of relative path, it's relative to the location of the manifest file itself.

For all non-core packages, this path is the path to the directory where the folders for the different AML types are stored. From our example in the previous section The File Structure, if the manifest file for the com.aras.innovator.solutions.PLM package is placed in the Solutions folder then the path to the PLM solution AML points to the \PLM\Import folder where the \ItemType, \Form, etc. folders are (path relative to the location of the manifest file itself is used), and the package tag would be written so:

```
<imports>
   <package name="com.aras.innovator.solution.PLM" path="PLM\import" />
</imports>
```

For core packages (admin, core, dashboards, and preferences), the path to package AML files is calculated based on the specific package name. For core packages the '.' in the fully qualified name of a core package is replaced with a '\' when calculating the file path based on these specific package names. From our example in the previous section The File Structure, if the manifest file for the com.aras.innovator.dashboards package is placed in the \Innovator\Imports folder, then the package tag would be written so:

```
<imports>
   <package name="com.aras.innovator.dashboards" path=".\" />
</imports>
```

depenedson tag

This tag contains the information about packages that the package defined in the `package` tag depends on. This also populates `the Package Depends On` relationship of the `Package Definition` item in the database. This information is used for creating in the database dependencies between packages. If the package referenced in the `dependson` tag is one of packages imported in the import session, then it is loaded prior to the package that depends on it; otherwise it's assumed that the package referenced by the `dependson` tag already exists in the target database

Note: If it doesn't then the import might fail because imported package might contain references to some items from the `dependson` package.

3.2 Export Tool

The Export tool allows user to select Package Elements to export to the file system as XML. These package elements can be exported individually, as part of a Package Group, or as part of a Package Definition.

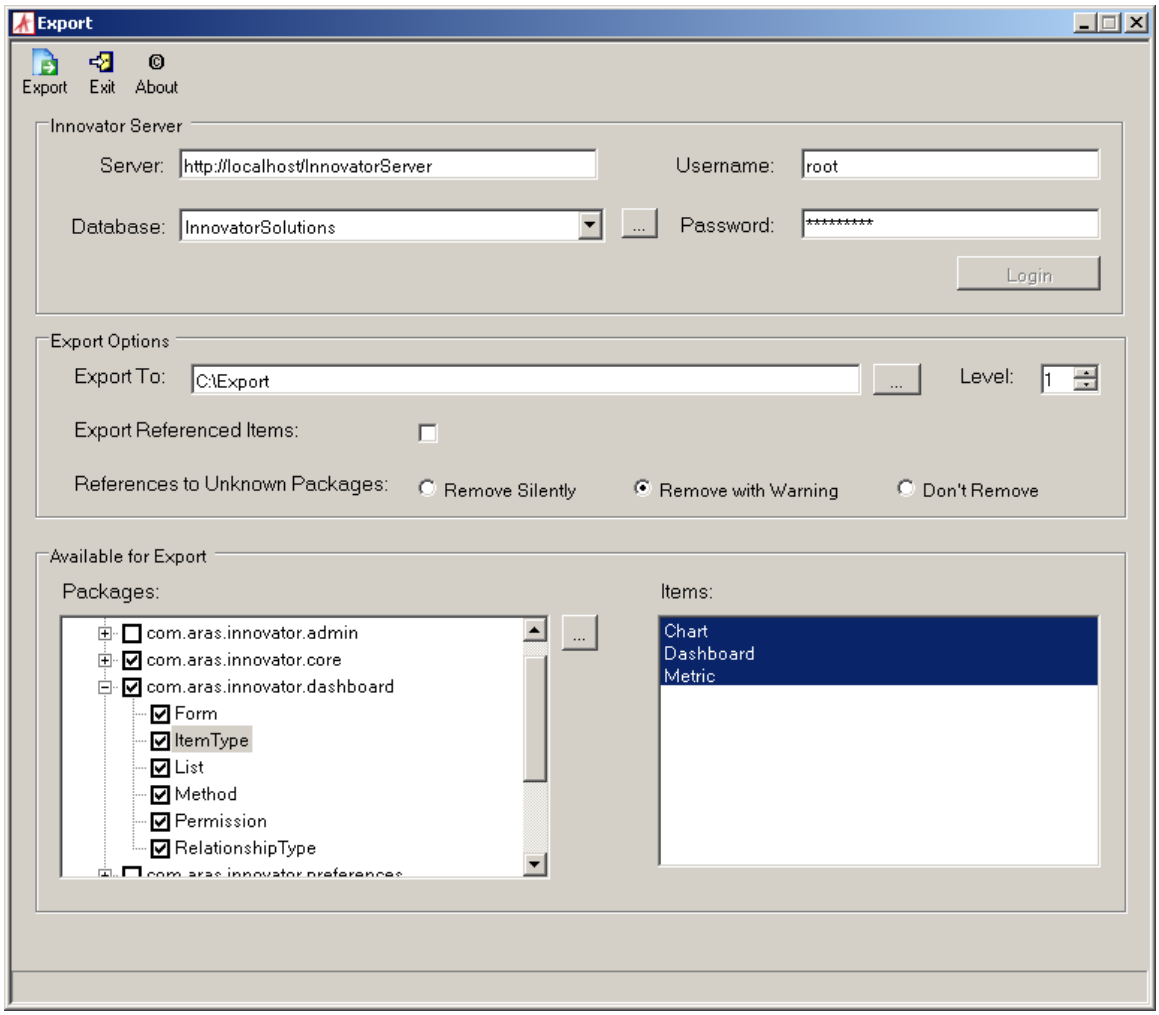

Figure 1.

1. The first step in using this utility is connecting to the Aras Innovator Server. To do this, we must first define the fields for doing so:
   - **Server**: The URL used to login to Aras Innovator. A default install uses a URL such as `http://localhost/InnovatorServer`
   - **Database**: The database you wish to export the package from. This filed is populated by selecting the ellipse to the right of it, after filling out the Server URL field.
   - **Username**: The login of the user that is used log into Aras Innovator. This is usually the root or admin logins when working in a non-production or upgrade environment.
   - **Password**: This is the password for the login filled out in the Username field.

   After all of this information is filled out, select the Login button to connect to the Aras Innovator Server.
2. The second step in running the utility is the Export Options.
   - **Export To**: The location in the file system where you wish to export the AML packages to.
   - **Levels**: This field is used to specify the levels attribute of the query in a limited number of ItemType queries not pre-defined by the tool.

- **Export Referenced Items**: This option is used to export Items explicitly defined in the package definition as a referenced Item.
- **References to Unknown Packages**: by 'Unknown Package' it's meant here a package that is neither a core package nor a package that the exported package depends on ('depends on' means that there is a dependency of type `PackageDependsOn` between exported package and the other package). If an item has references to items in the 'unknown' package, these references are normally removed during the export. One example of this is the 'Quality Planning' Identity. The 'Quality Planning' Identity is used in the 'New Part' Permission, and because the Quality Planning solution depends on the Product Engineering solutions exporting this relationship could create a circular reference. Therefore, the reference must be handled.
  - **Remove Silently**: References to unknown packages are removed. (The 'New Part' Permission contains no reference to the 'Quality Planning', and this removal is not logged.)
  - **Remove with Warning**: References to unknown packages are removed, and a warning is given to allow the user to resolve this reference separately. (The 'New Part' Permission contains no reference to the 'Quality Planning'.)
  - **Don't Remove**: References to unknown packages are not removed (The 'New Part' Permission contains reference to the 'Quality Planning'. This can cause import errors if the 'Quality Planning' Identity does not exist in the database when the created package is imported.)

3. The last step is to choose what Package Elements should be exported in the Available for Export section. Use the ellipse next to the left panel to refresh the list of available elements for export. It is possible to export only a part of a package by expanding the package in the list of packages on the bottom left of the form and selecting a particular package groups in the tree and/or particular items in the list of items on the bottom right of the form for export.

3.3 Import Tool

The Import tool allows user to select predefined manifest files, and import the corresponding package AMLs into a database.

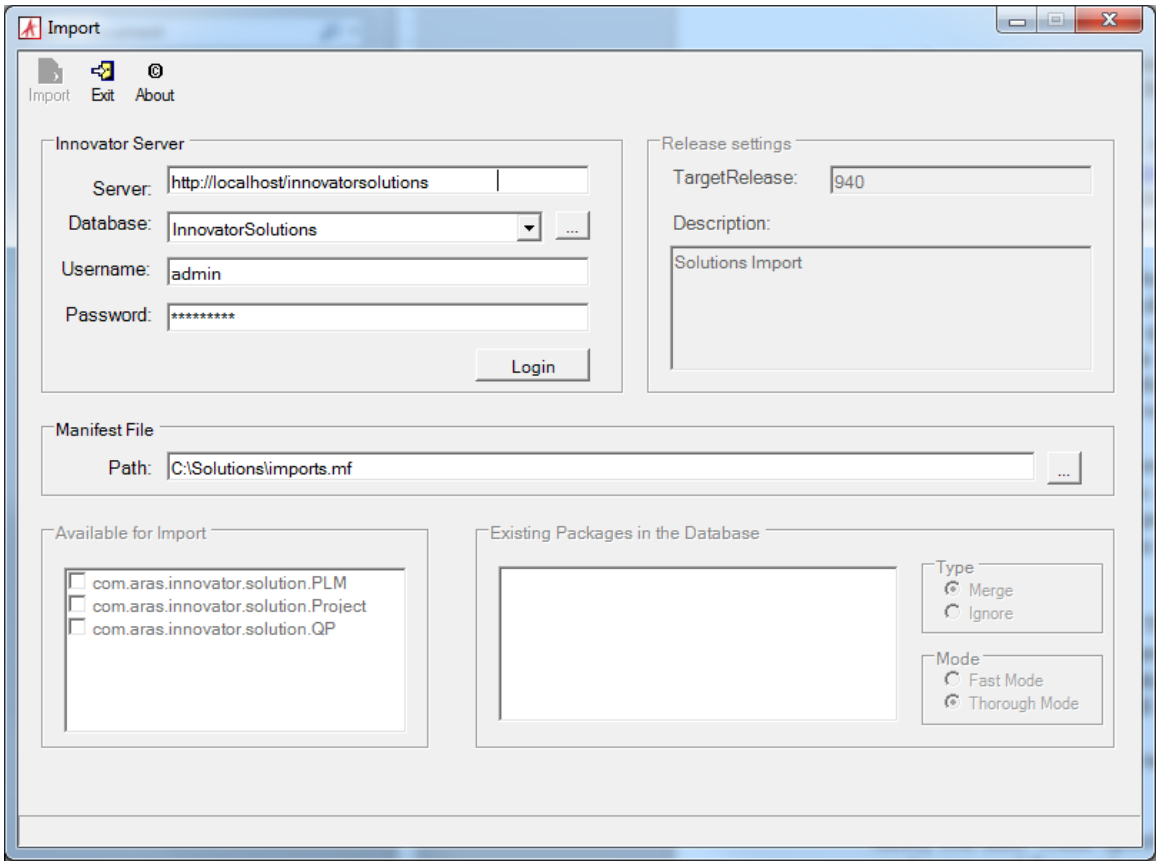

Figure 2.

1. The first step in using this utility is connecting to the Aras Innovator Server. To do this, we must first define the fields for doing so:
   - **Server**: The URL used to login to Aras Innovator. A default install uses a URL such as `http://localhost/InnovatorServer`
   - **Database**: The database you wish to import the package to. This filed is populated by selecting the ellipse to the right of it, after filling out the Server URL field.
   - **Username**: The login of the user that is used log into Aras Innovator, and apply the package AML. This is usually the root or admin logins when working in a non-production or upgrade environment.
   - **Password**: This is the password for the login filled out in the Username field.

   After all of this information is filled out, select the Login button to connect to the Aras Innovator Server.
2. The second step in running the utility is the Release Settings. This defines the target database release version number of the package being imported and the description of the purpose of the import. Release Settings are also used by Aras Innovator classes that allow user to obtain information about database upgrades applied to the database.
   - **Target Release**: The target release number of the database after the import is complete. When migrating packages, this is the current version number of the database (12.0 to 12.0 imports). When upgrading a database, this is the upgrade target version of the database (9.4.0 to 12.0 imports).
   - **Description**: A brief description of the imports purpose.

3. Next, the user must specify a manifest file that contains information about what packages and from where on disk should be imported into the database.
4. Fourth, the user must select what packages from the manifest to import to the database from the Available for Import section.
5. Last, a choice must be made as to how conflicts with existing packages in the database should be resolved.

   The main display of this section shows what packages are already present in the database.

   - Type: Determines how existing element in a package is handled.
     - Ignore - Skip imported items if they already exist in the database
     - Merge - Update package items in the database with the new AML

     If item action specified in AML is not 'add' it always used as is

     If item action specified in AML is 'add' then it's replaced by 'edit' if item with the ID already exists in the database.
   - Mode: Determines the level of error checking that is performed before an AML is imported to the database.
     - Fast - No additional verifications of the imported AML is done during the import process.
     - Thorough - During the import additional checks about the AML's dependent items are checked for existence in the database before applying. (e.g. making sure all properties exist with the correct ID before attempting to apply an ItemType)

Aras recommends that imported AML files always contain item action 'add' for every imported item unless it is specifically required otherwise (e.g. the item with a particular ID must be removed from the database). Another important thing to understand is how the import processes versioned items. Package Elements always contains a `config_id` of a versionable item. When a versionable item is imported its first made an attempt to find its `config_id` and then use the ID in the Package Element. Correspondingly, exporting a versionable item always writes its `config_id` as id in the resulting AML file (see Export above).

3.4 Console Upgrade Tool

The Console Upgrade Tool is a command line version of both the **Export Tool** and **Import Tool** described above. The command line parameters can be obtained by typing '/?' as the command line parameter.

```
Required Parameters :

        server=url      server's URL (e.g. server=http://localhost/InnovatorServer)
        database=db     database's name (e.g. database=dbWithoutSolution)
        login=uname     user's login (e.g. login=root)
                        NOTE: login must have root or admin privileges
        password=pw     user's password (e.g. password=xxxx )
        release=rel     release's name used by import only (e.g.
                        release=rel11.0)
        mfFile=path     explicit path to .mf file (Not required for export.  Default
                        behavior is "export all")
Optional Parameters  :

        import          import or export  (by default export)
```

```
merge             used by import only  (by default uses non-merge option)

fastmode          used by import only  (by default uses thorough mode that
                  provides more through verification of the applied AML)
verbose           used by import and export (by default uses non-verbose)

dir=d1            export: output directory
                  import: location of the manifest file
                  NOTE: the specified path must exist. If the parameter
                        is not specified at all, user is prompted for
                        the path
log=logpath       full path to the log file. If specified file already
                  exists it's appended.
description=desc  release description used by import only
                  (e.g description="SolutionsUpgrade")
vlog              save the resulting log file on the vault
                  (don't save if the argument wasn't specified)
level=n           request attribute 'level' that is used for
                  non-dictionary item types (used by export only)
                  default: level=1
```

3.5 Package Definition Tool

The Package Definition Tool allows creating an instance of the Package Definition in the database. This is a temporary utility that is created only for the situations when Aras Innovator has a set of items that makes up a Package Definition, but these items are not included as part of a Package Definition. Generally, this only occurs when the database was upgraded from a version of Aras Innovator that pre-dated the corresponding use of Package Definitions for this solution.

Note: The Package Definition Tool does NOT import anything into the database. It creates a Package Definition in the database that later can be used for managing the Package Elements.

3.5.1 The Package Definition Tool GUI

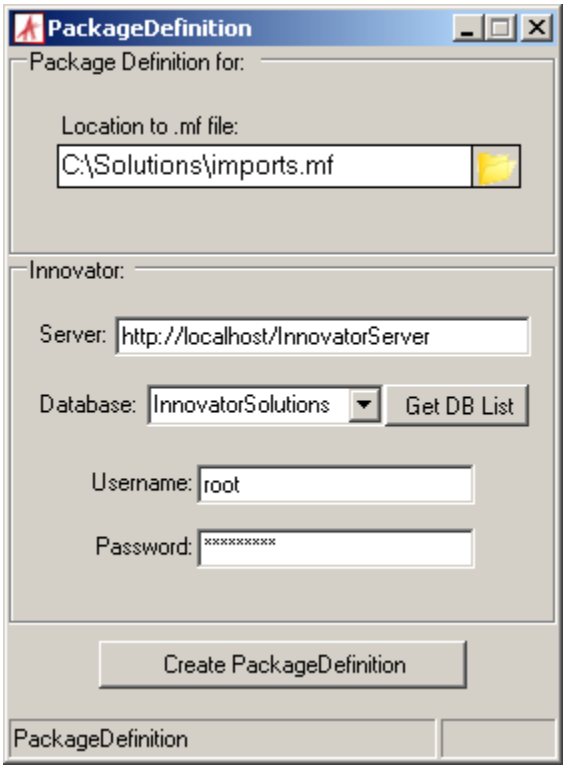

Figure 3.

1. The first step in using this utility is specifying a manifest file that contains information about what packages and from where on disk should be defined in the database.
2. The next step is to establish the connection parameters:
   - **Server**: The URL used to login to Aras Innovator. A default install uses a URL like http://localhost/InnovatorServer
   - **Database**: The database you wish to define the package in. This filed is populated by selecting the 'Get DB List' button, after filling out the Server URL field.
   - **Username**: The login of the user that is used to log into Aras Innovator. This is usually the root or admin logins when working in a non-production or upgrade environment.
   - **Password**: This is the password for the login filled out in the Username field.

   After all of this information is filled out, click **Create Package Definition** to begin.

3.5.2 The Package Definition Tool Command Line

Alternately, the Package definition tool can be executed from the command line. The command line parameters can be obtained by typing '/?' as the command line parameter.

Usage:

```
PackageDefinition.exe {server url} {user} {password} {db} {path to manifest
file}
 server url              server's URL (e.g. http://localhost/InnovatorServer)
```

```
user                    user's login
                        NOTE: login must have root or admin privileges
password                user's password
database                database's name
path to manifest file   location of the manifest file
                        NOTE: the specified path must exist
```

EXAMPLE:

```
C:\PackageImportExportUtility\PackageDefinition\PackageDefinition.exe
http://localhost/InnovatorServer root innovator InnovatorSolutions
C:\Solutions\core_imports.mf
```

3.6 Creating a Package Definition from the Aras Innovator UI

Package Definitions can also be created for new solutions using administrator features in the Aras Innovator user interface. These packages should be created with a specific data model in mind, and you should be careful to review your package for any referenced items that may not be in the next database you import to. Lists, for instance, are commonly forgotten when creating a package definition, but causes errors when trying to import ItemTypes that reference them.

In order to create a package definition, you must be an administrator in Aras Innovator.

1. From the main search grid right click the Item you wish to add to a package. A menu similar to the following appears:

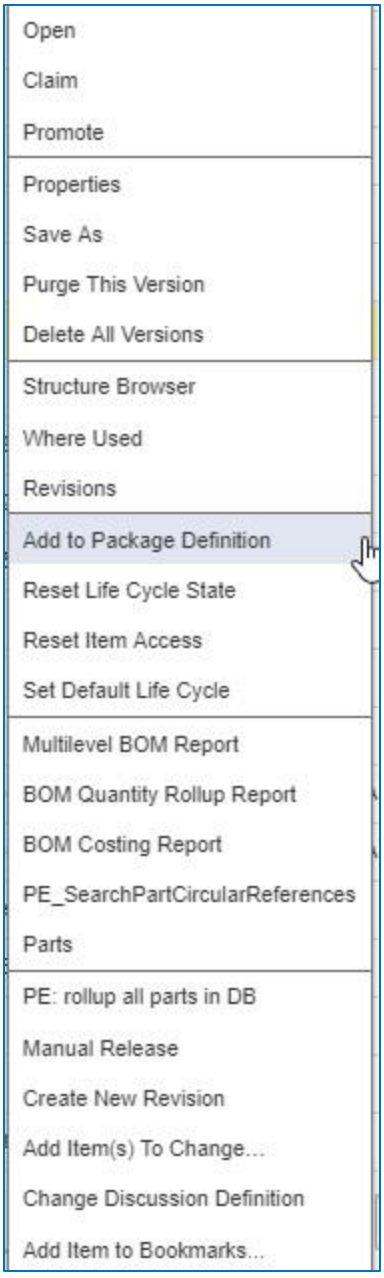

Figure 4.

2. Click **Add to Package Definition**. The following dialog appears.

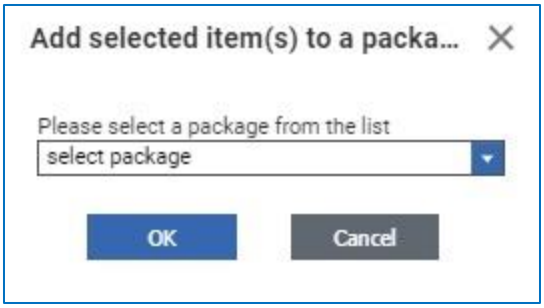

Figure 5.

3. From the dialog, ether select an existing package or 'create new'

   If you select an existing package, the item is added to the existing package, and you are done.

4. If you select 'create new' you are prompted to define the new package.

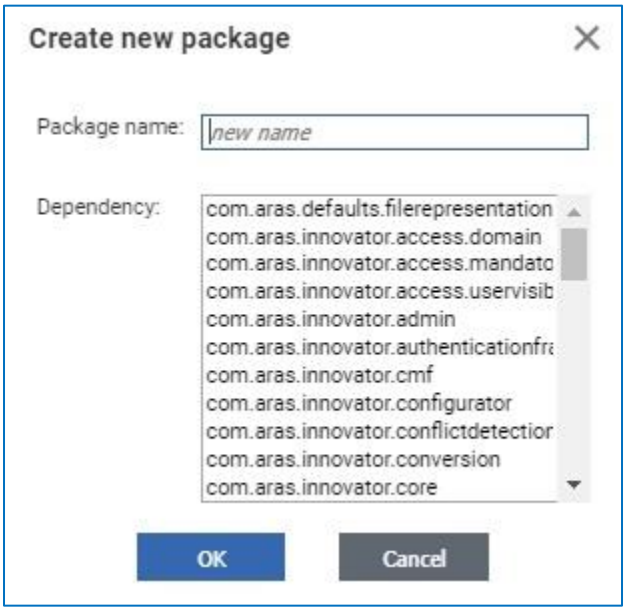

Figure 6.

   - Package name should be alphanumeric without spaces, as a best practice. The package name is used to create file folder names on export.
   - Dependency should be the packages that are required to exist in the database before the defined package can be imported.

After you have created your package, you may review it and other existing package definitions by selecting Administration\Package Definitions in the TOC.

3.6.1 What to include in a Solutions AML Package

There is no fixed list for what Items to include in a solutions AML package, because the list would change based on the version of Aras Innovator or solution you are working with. However, there is a basic list of the Core ItemTypes that make up the basic metadata of a database. This is NOT a definitive list, but should act as a helpful guideline when creating your packages.

- Actions

- CommandBar… (all items)
- Conversion Rules
- Derived Relationship Fmilies
- E-Mail Messages
- File Types
- Forms
- Grids
- Identities
- ItemTypes (Exclude ItemTypes with is_relationship=1)
- Life Cycle Maps
- Lists (Exclude Lists associated with PolySources like ‘Change Controlled Item’ and ‘Deliverable’)
- Mac Policies
- Methods
- Permissions
- Presentation Configurations
- Query Definitions
- RelationshipTypes
- Reports
- Sequences
- SQLs
- User Messages
- Variables
- Workflow Maps
- xClassification Trees
- XML Schemas
- XML Schema Elements
- xProperty Definitions