Aras Innovator 2024 Release

Package Import Export Utilities

Document #: *D-008119*

Last Modified: *5/20/2024*

Copyright Information

Aras Corporation
100 Brickstone Square
Suite 100
Andover, MA 01810
Phone: 978-691-8900

E-mail: support@aras.com

Website: https://www.aras.com/

Table of Contents

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. 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 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.

1 Data Model

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

- `PackageElement`: it represents the ID number of the item that has been defined in the database.
- `PackageGroup`: it 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`: it 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`: it is a relationship type that establishes dependencies between packages
- `PackageReferencedElement`: it is a relationship on the `PackageDependsOn` type that defines what exact package elements from the 'related' package the 'source' package references. This could be useful when packages are exported (checkbox '`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 the correct AML was to apply otherwise.

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

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

2 Using Package Import Export Utilities

2.1 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.

2.1.1 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 the 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 even though the solution packages have a fully qualified name. 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/

2.1.2 The Manifest File

The manifest file contains information about what packages can be processed by the utilities, dependencies between packages, and where to find the package 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>
```

The `package` tag

The `package` tag can have the following attributes:

- `name`: a unique, fully qualified name of a package.
- `path`: a path to the directory that contains the package AML files.

Note: A path to the package AML files could be either absolute or relative; in the case of a relative path, it is relative to the location of a 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 The File Structure section, if the manifest file for the **com.aras.innovator.solution.PLM** package is placed in the **Solutions** folder, the path to the PLM solution AML points to the **\PLM\Import** folder where the **\ItemType**, **\Form**, etc. folders are.

A 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 The File Structure section, 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>
```

The `dependson` 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 does not, the import might fail because an imported package might contain references to some items from the `dependson` package.

2.1.3 Multilingual Packages

The Aras Innovator platform includes a “Multilingual String” datatype that enables users to view and edit specific properties in the supported language of their choice. These multilingual properties may be managed with the Import and Export Utilities.

By default, exported packages contain only the English values for multilingual properties.

```
<AML>
 <Item type="ItemType" id="0BB5B81FEB37475BB9C779408080DB61" action="add">
  <allow_private_permission>1</allow_private_permission>
  ...
  <is_versionable>0</is_versionable>
  <label>Field</label>
  <label_plural>Fields</label_plural>
  <large_icon>../images/Properties.svg</large_icon>
     ...
```

However, packages may optionally be exported with multilingual property values for additional languages. The following example shows the same package item with French, German, and Japanese values in addition to the default English. Each multilingual property will be represented with a tag for each language value present in the database.

```
<AML>
 <Item type="ItemType" id="0BB5B81FEB37475BB9C779408080DB61" action="add"
xmlns:i18n="http://www.aras.com/I18N">
  <allow_private_permission>1</allow_private_permission>
  ...
  <is_versionable>0</is_versionable>
  <label xml:lang="en">Field</label>
  <i18n:label xml:lang="fr">Champ</i18n:label>
  <i18n:label xml:lang="de">Feld</i18n:label>
  <i18n:label xml:lang="ja">フィールド</i18n:label>
  <label_plural xml:lang="en">Fields</label_plural>
  <i18n:label_plural xml:lang="fr">Champs</i18n:label_plural>
  <i18n:label_plural xml:lang="de">Felder</i18n:label_plural>
  <i18n:label_plural xml:lang="ja">フィールド</i18n:label_plural>
  <large_icon>../images/Properties.svg</large_icon>
     ...
```

Note: Multilingual property tags are indicated using the standard internationalization or “i18n” language codes.

2.2 Export Tool

The Export tool allows the 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.

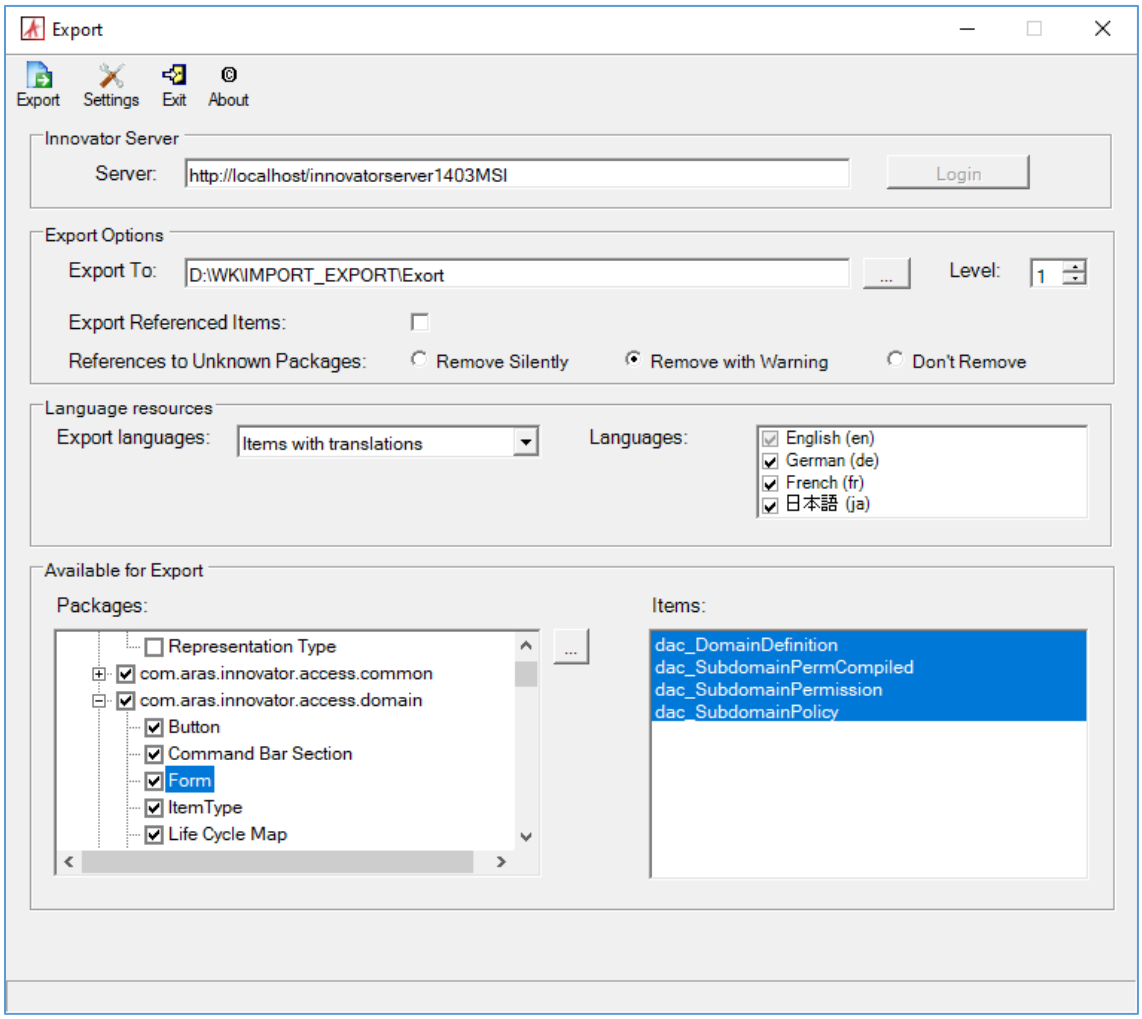


Figure 1.

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`

   Then click Login button and use opened embedded browser window to connect to the Aras Innovator Server.

Note: The Export tool supports Client Certificate Authentication. If Aras Innovator server is configured for CCA, a dialog for certificate selection can be shown.

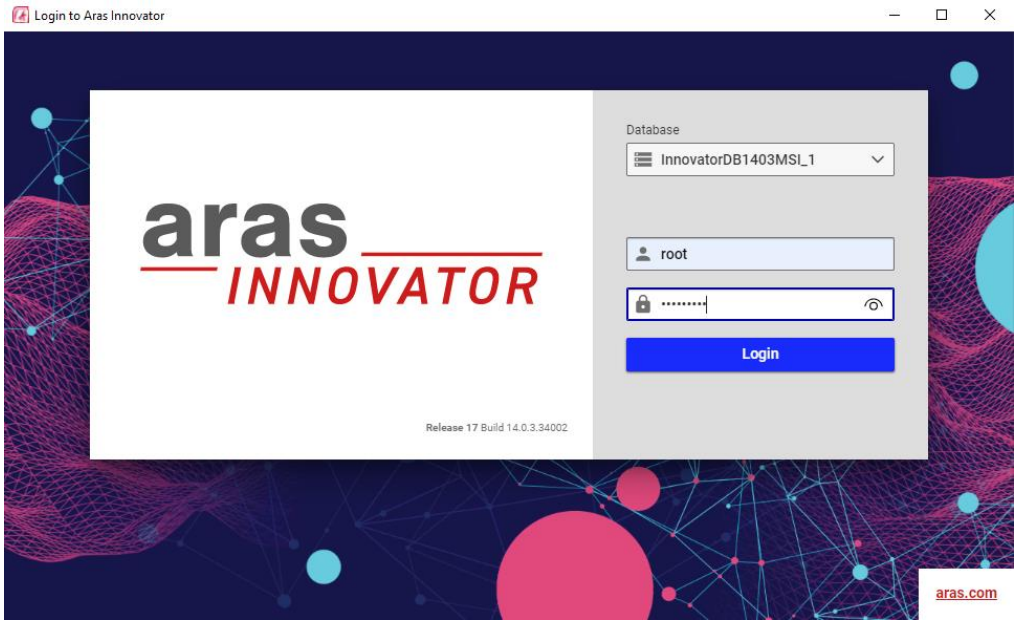


Figure 2.

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 third step is choosing which language resources to export. The Export tool supports three modes of exporting multilingual package data:
    - **None**: All properties are exported. For multilingual properties, only the default language is exported (English). This is the default setting for the Export utility.
    - **Items with translations**: All properties are exported. The selected languages are included for multilingual properties.
    - **Translations only**: Only multilingual properties with translations for the selected languages are exported. Properties of all other datatypes are excluded from the export (string, date, Item, etc.).
4. 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 group in the tree and/or items in the list of items on the bottom right of the form for export.

2.3 Import Tool

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

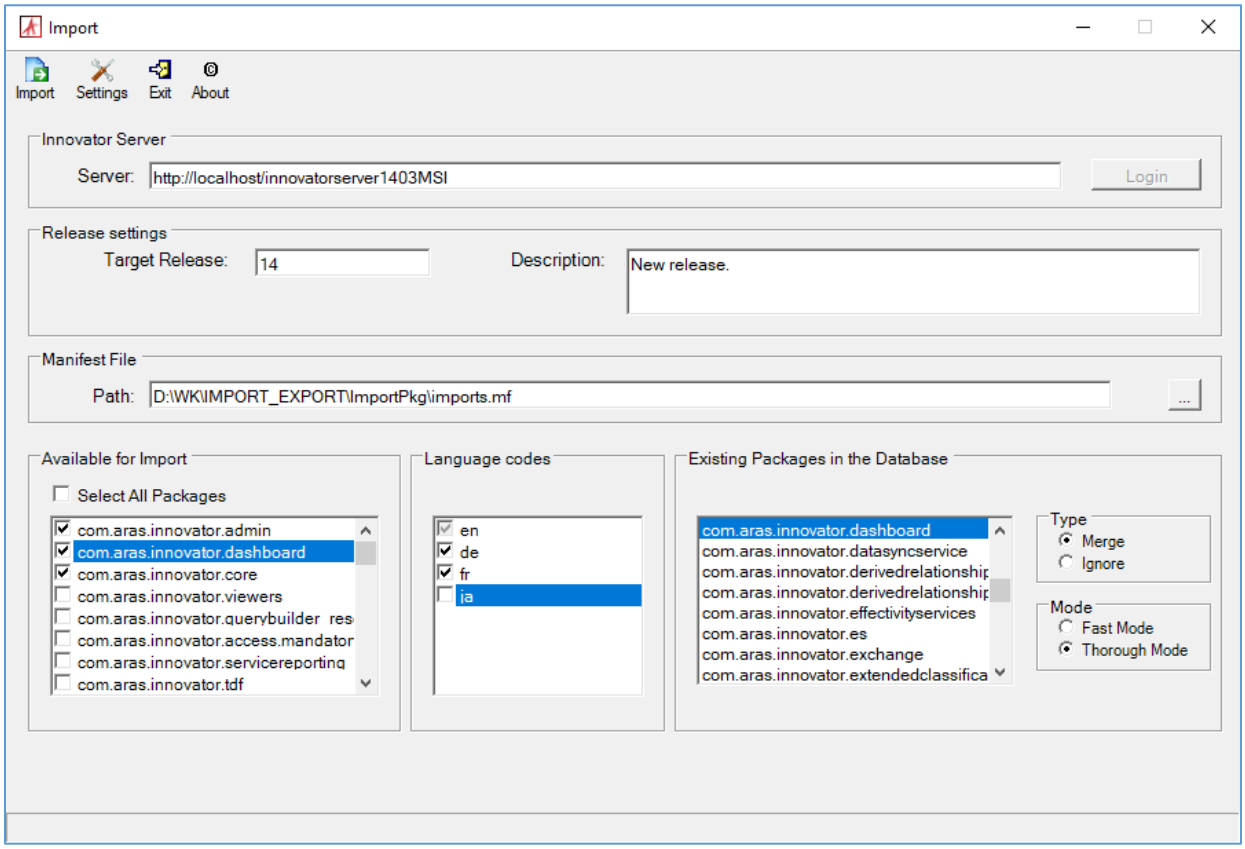


Figure 3.

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`

   Then click Login button and use opened embedded browser window to connect to the Aras Innovator Server.

Note:	Export tool supports Client Certificate Authentication. If Aras Innovator server is configured for CCA, a dialog for certificate selection can be shown.

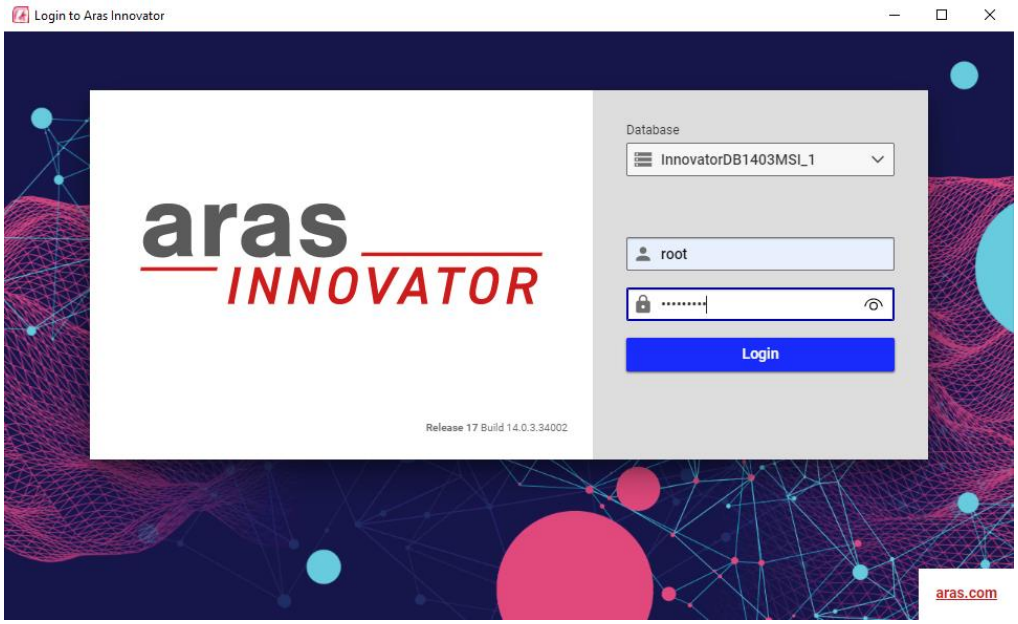


Figure 4.

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 (14 to 14 imports). When upgrading a database, this is the upgrade target version of the database (9.4.0 to 14 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. If a package contains multilingual properties, the tool indicates the available languages in the “Language codes” list. The user may choose to import only the default English values, or they may choose to include one or more additional languages.
6. 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 are 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 tried 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).

2.4 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)
        export-with-translations exports packages in “Items with translations” mode.
        export-only-translations exports packages in “Translations only” mode.
        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
        langs       List of language codes for import/export. 'en' code is
                    required. '*' value can be passed to import/export all
                    available languages.
        crtloc      Certificates store location (default: CurrentUser)
        crtstore    Certificates store name (default: My)
        crtftype    Find type for certificate search (default: FindBySubjectName)
        crtfvalue   Find value for certificate search
        tcrtvalid   Certificate should be valid (default: true)
```

Please note that 'crtstore', 'crtloc', 'crtftype', 'crtfvalue', 'crtvalid' options are required for Client Certificate Authentication. Using these options, the tool can use appropriate certificate for authentication.

You can pass the data about client certificate by ConsoleUpgrade.exe.config file. Below you can find an example of the configuration section:

```
<Aras>
    <Net>
      <RequestProvider>
        <providers>
          <provider uriPattern=".*">
            <ClientCertificate>
              <certificate sourceType="CertificateStore"
storeLocation="CurrentUser" storeName="My" findType="FindByThumbprint"
findValue="71f942b0710faa873d1d3de2a9815b3321604d5a"/>
            </ClientCertificate>
          </provider>
        </providers>
      </RequestProvider>
    </Net>
  </Aras>
```

2.5 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.

2.5.1 The Package Definition Tool GUI

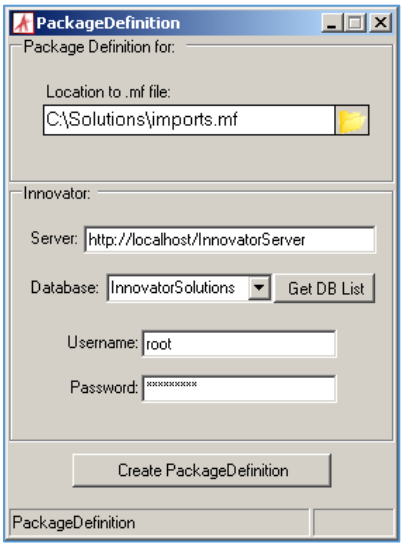


Figure 5.

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 field 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.

2.5.2 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
```

2.6 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.

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 like the following appears:

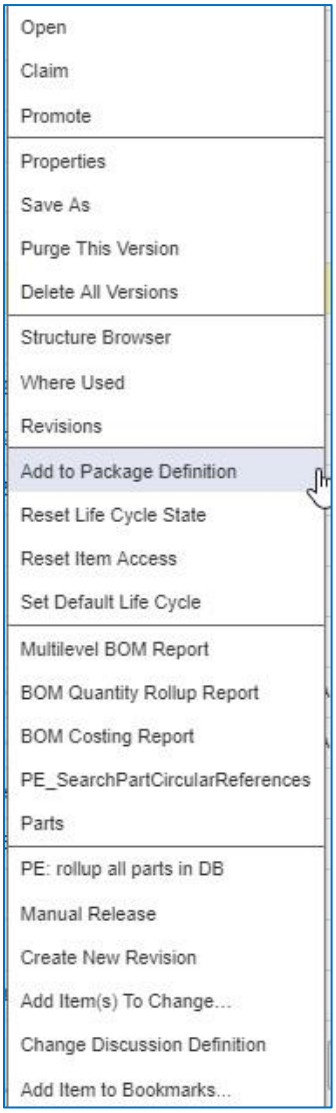


Figure 6.

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

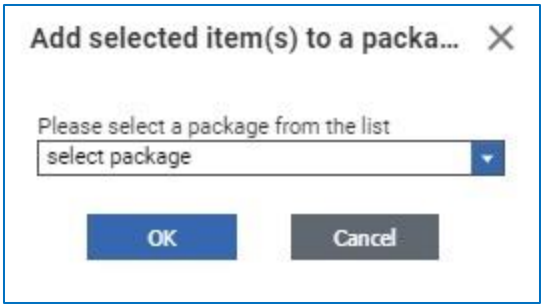


Figure 7.

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.

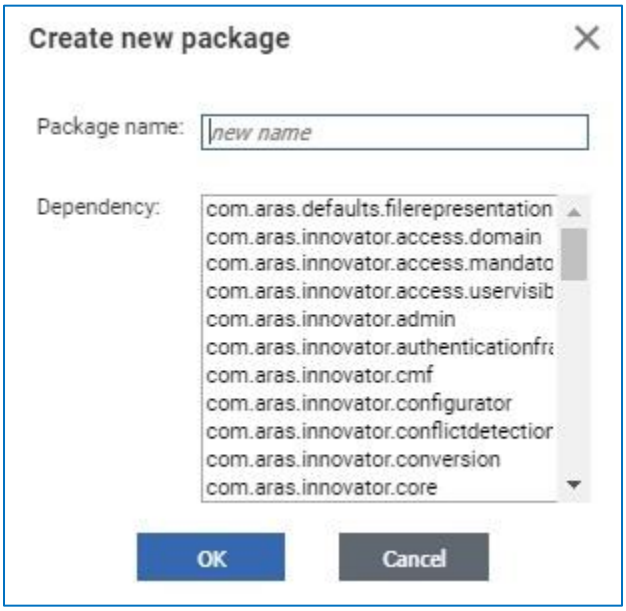


Figure 8.

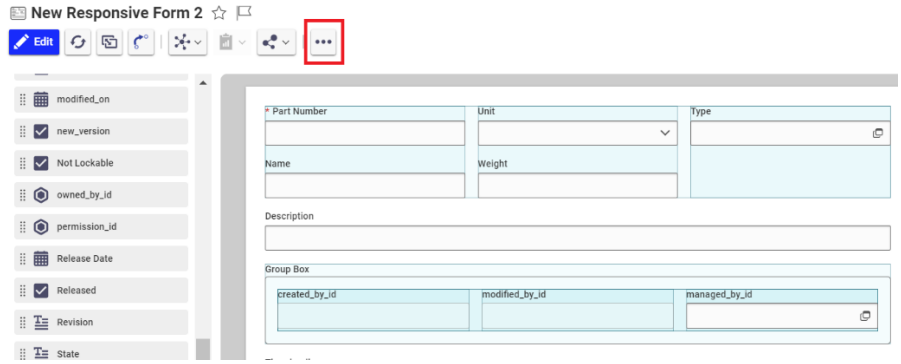
- Package name should be alphanumeric without spaces, as a best practice. The package name is used to create file folder names on export.
- Dependency should be the packages that are required to exist in the database before the defined package can be imported.

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

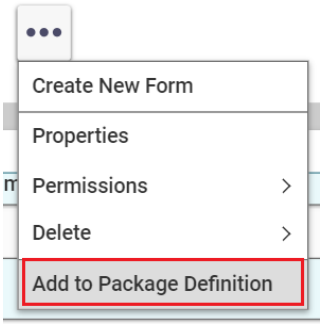
2.6.1 Add form to a package via the Responsive Form Designer

The following steps outline the process of adding a form to a package via Responsive Form Designer:

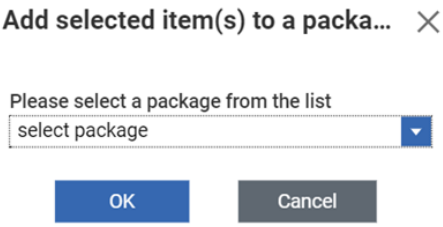
1. On **Responsive Form Designer**, click on the **More** icon.



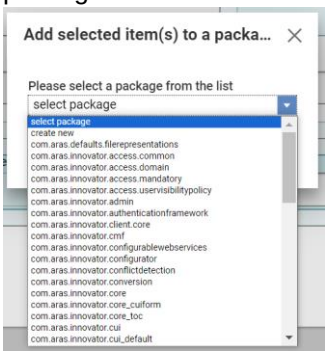
2. Select **Add to Package Definition**.



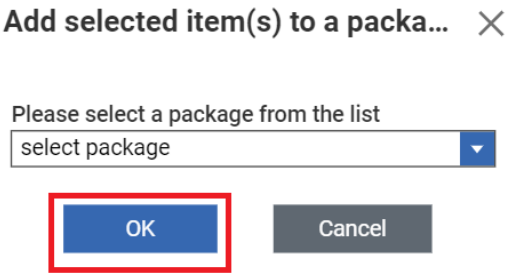
The **Add selected item(s) to a package** dialog box appears.



3. In the **Select Package** dropdown, either select the existing required package or create a new package.



4. Click **Ok**.



2.6.2 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