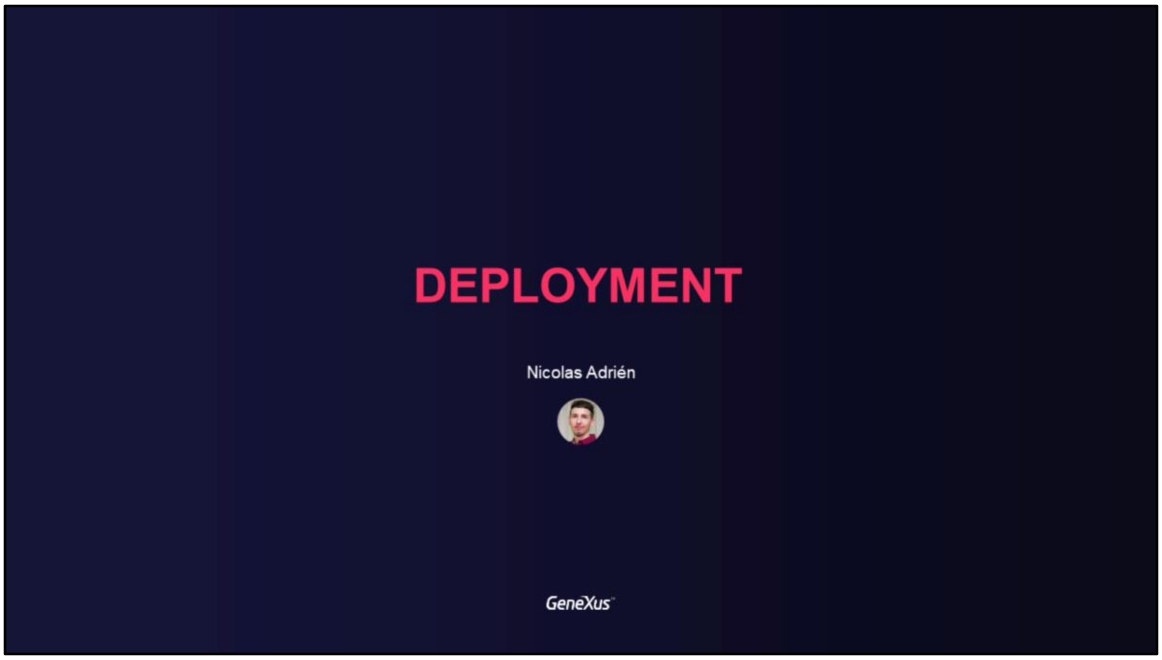
GeneXus™

by Globant



DEPLOYMENT

Nicolas Adrién

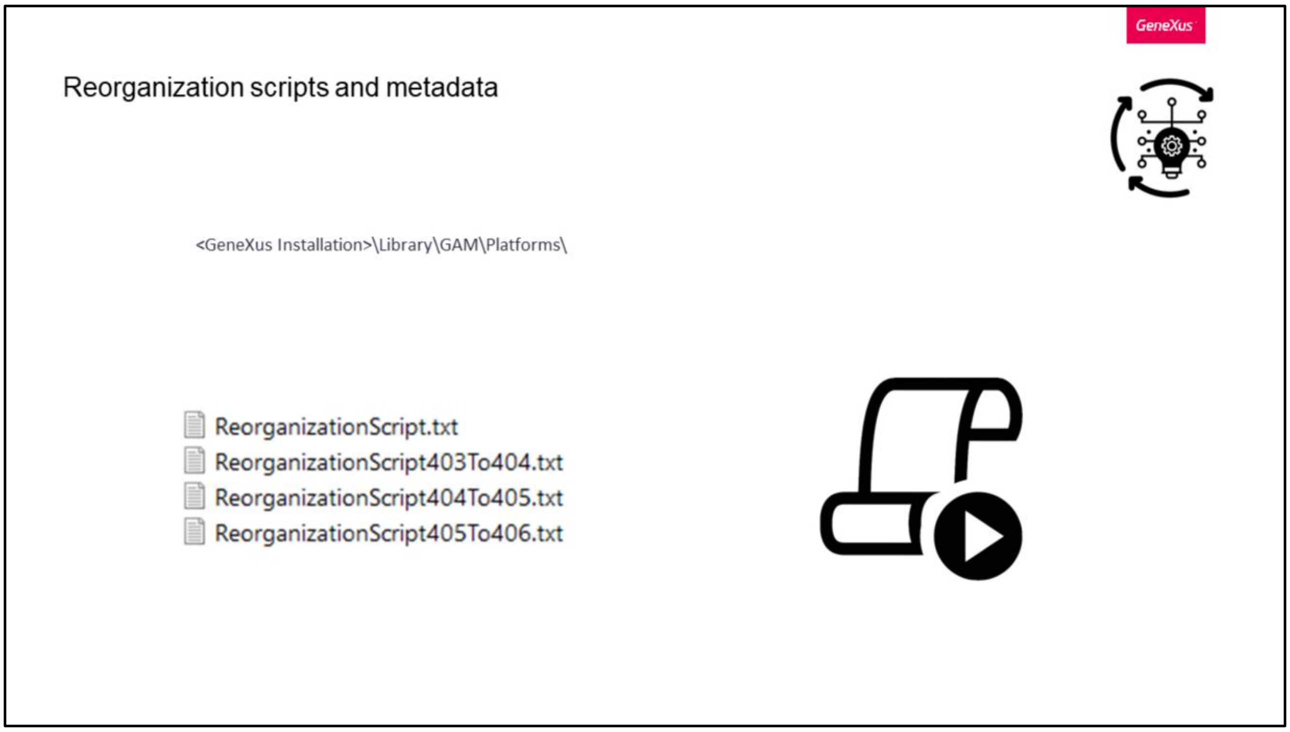
GeneXus

DEPLOYMENT

Application deployment

GeneXus™

In this video, we will talk about the transition between environments and upgrades, and common problems with it, in addition to learning how to use the GAM Deploy Tool.



Reorganization scripts and metadata

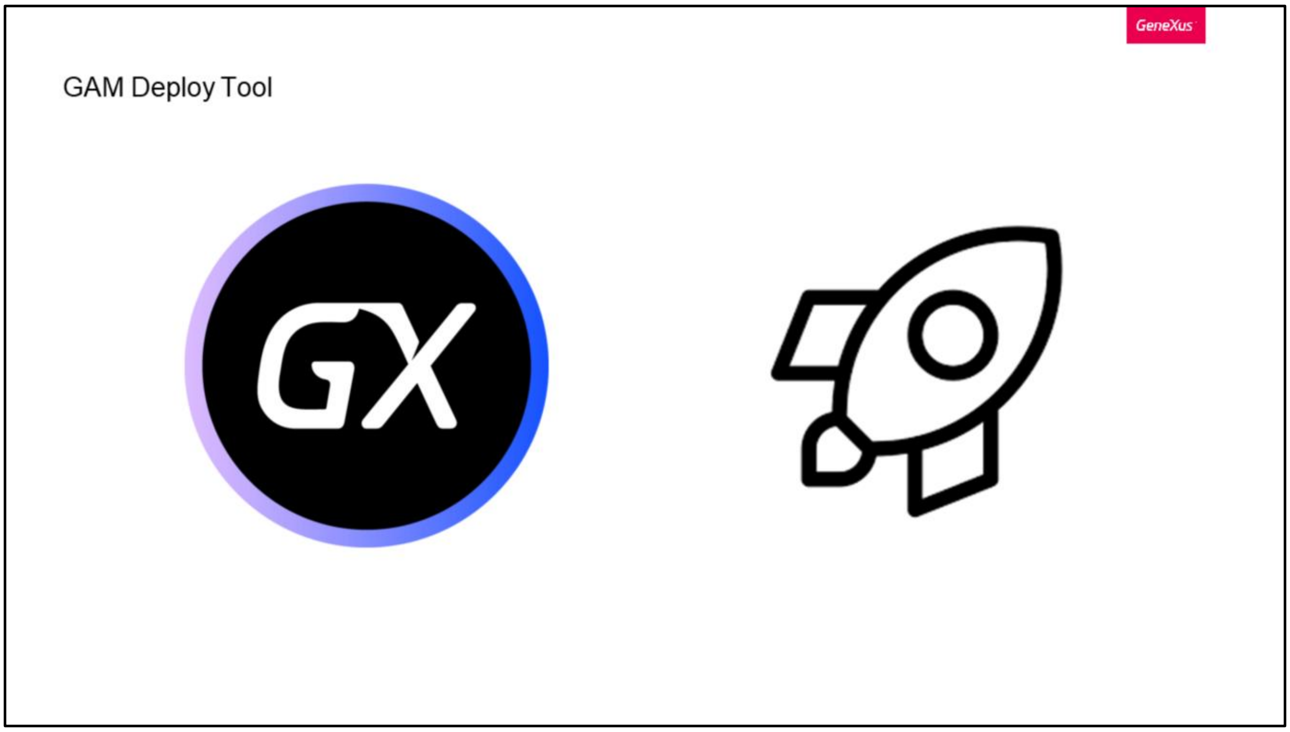
<GeneXus Installation>\Library\GAM\Platforms\

- ReorganizationScript.txt
- ReorganizationScript403To404.txt
- ReorganizationScript404To405.txt
- ReorganizationScript405To406.txt

Sometimes it may happen that a developer doesn't have access to the database and wants to perform a GAM pass between environments.
Here we may wonder how to perform the reorganizations and metadata impacts.

To solve this problem, GAM provides a number of scripts that are already included in the GeneXus installation folder.
Inside the Platforms folder, you should find the environments that we have defined in our KBs.
There we will find the following files.

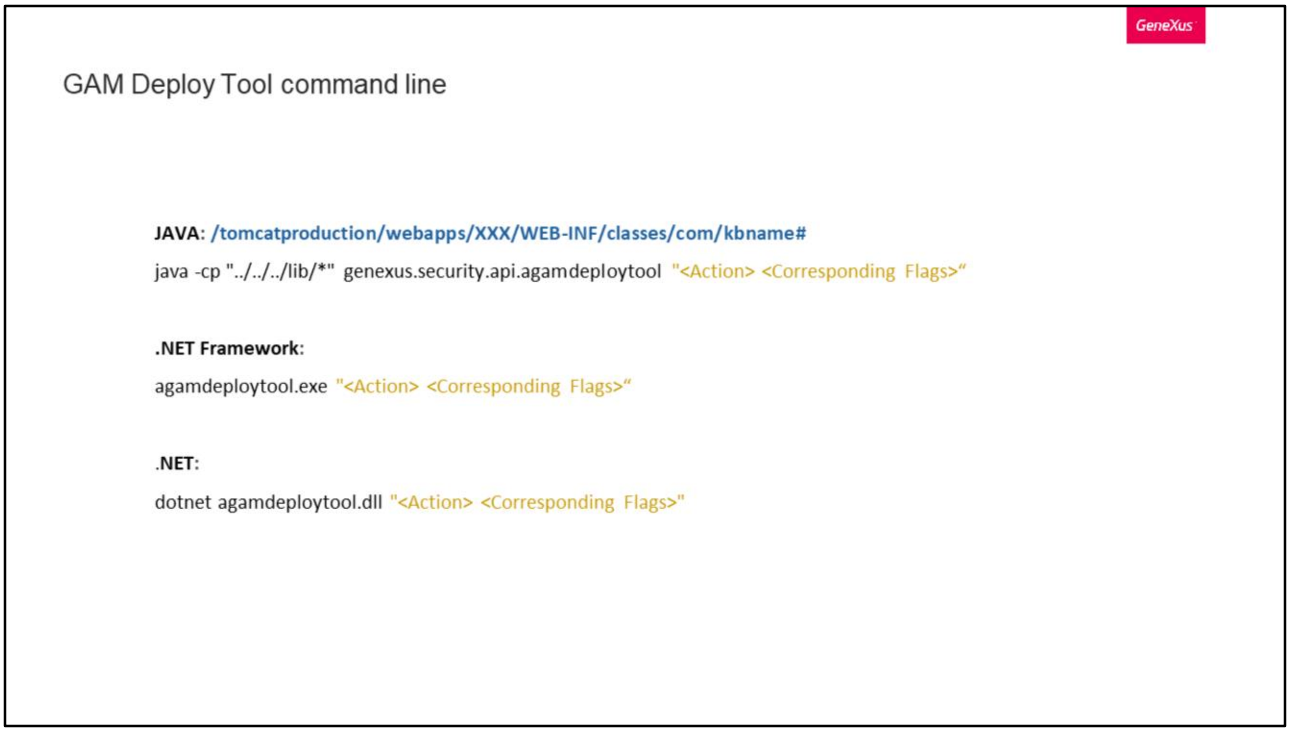
With the first file we will be able to create the entire GAM database.
The remaining ones consist of migrations between the different versions of GAM, as their names indicate. Please note that if these migrations are used, they must be executed for everything to work correctly.
This is part of the database structure, where those files will contain all the corresponding SQL statements that must be executed in a GAM database client.


GeneXus

GAM Deploy Tool

The GAM Deploy Tool's purpose is to help us deploy applications using GAM, since it contains knowledge of the structure and relationships of the GAM database, which allows us to gradually import data while maintaining the consistency of the data model. The tool is intended to take to production the information that needs to be updated in the GAM Database, and is available from GeneXus 16 U5.

We will focus on the command line version.



GAM Deploy Tool command line

JAVA: **/tomcatproduction/webapps/XXX/WEB-INF/classes/com/kbname#**

```
java -cp "../../../lib/*" genexus.security.api.agamdeploytool "<Action> <Corresponding Flags>“
```

.NET Framework:

```
agamdeploytool.exe "<Action> <Corresponding Flags>“
```

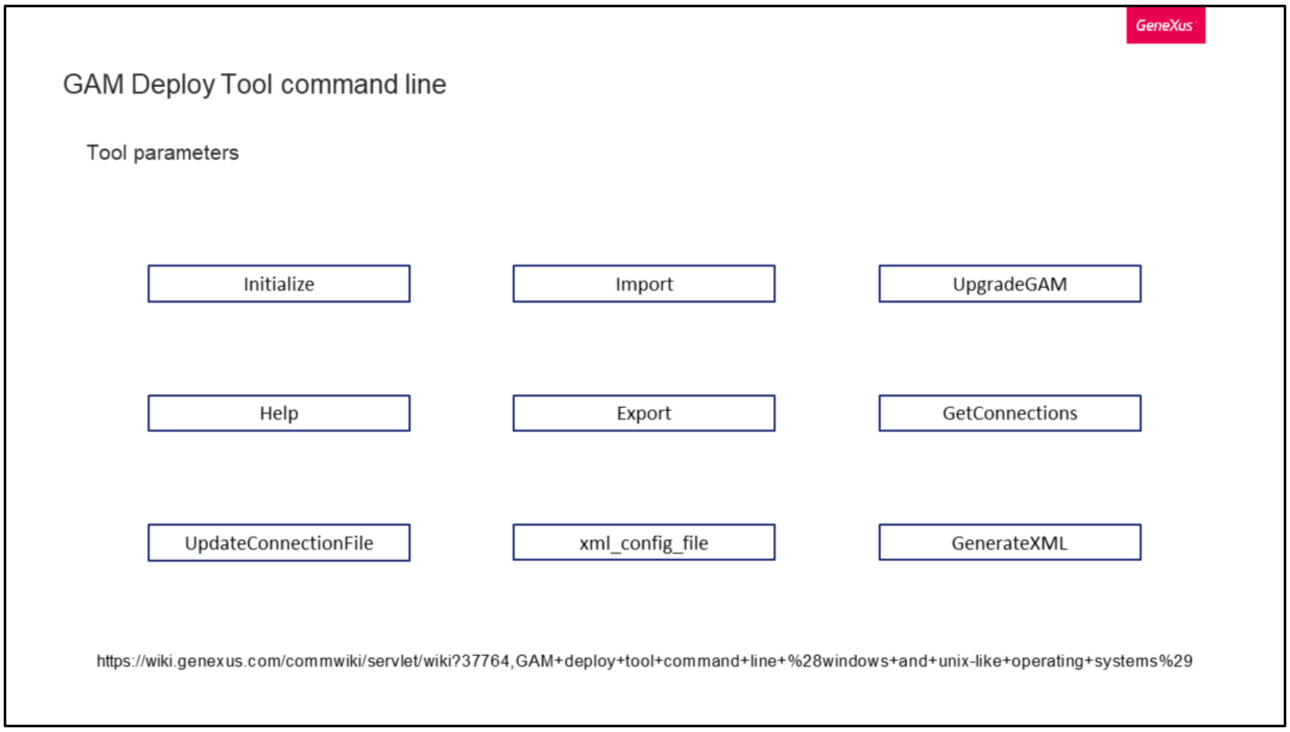
.NET:

```
dotnet agamdeploytool.dll "<Action> <Corresponding Flags>"
```

The calls to the tool should have the following format, depending on the environment in which they are located.

Ideally, the tool should be run below the deployment environment. In the case of .NET, in the \bin virtual directory. In the case of Java, below its webapp as we can see in the command.

The tool will not request the connection configuration of the GAM database, such as server, port, user, or password, because this information is taken from the configuration files such as client.cfg in the case of Java and web.config in the case of .NET.



GAM Deploy Tool command line

Tool parameters

Initialize
Import
UpgradeGAM
Help
Export
GetConnections
UpdateConnectionFile
xml_config_file
GenerateXML

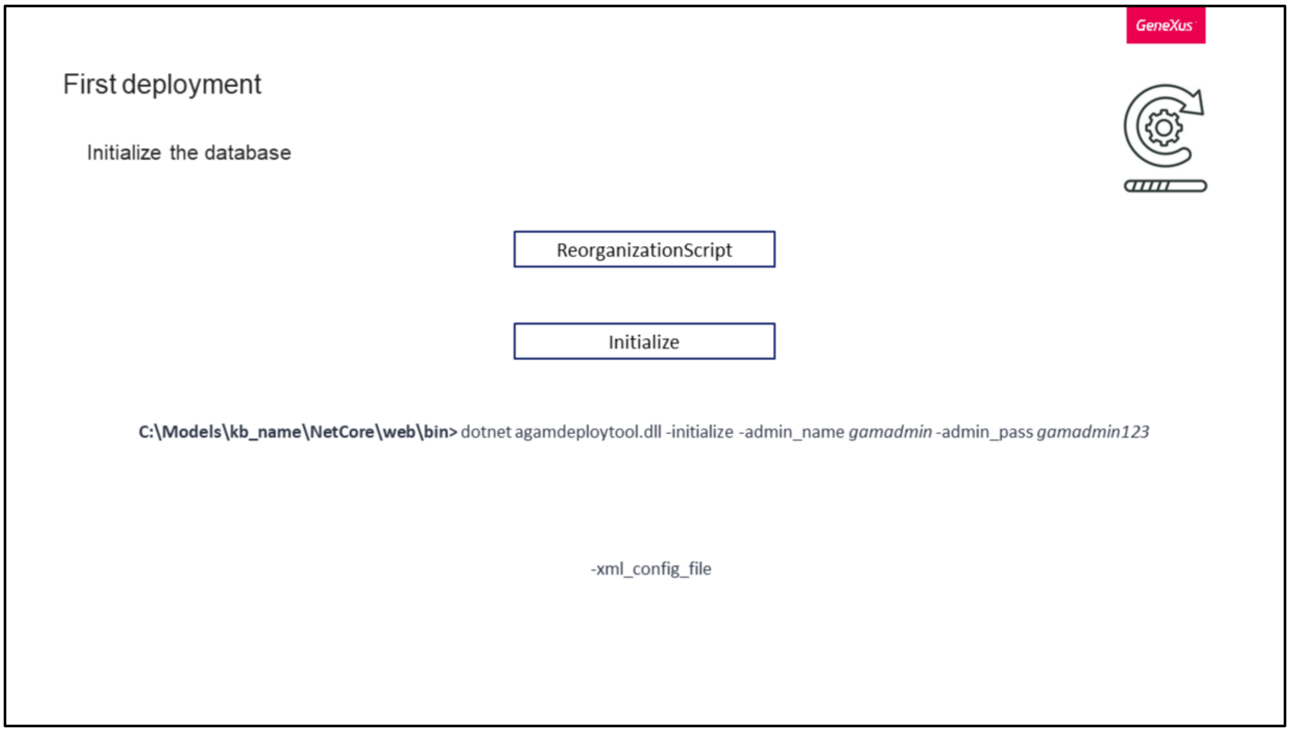
https://wiki.genexus.com/commwiki/servlet/wiki?37764,GAM+deploy+tool+command+line+%28windows+and+unix-like+operating+systems%29

This tool provides a variety of actions that we can do with it. Let's see them in detail.

- Initialize: Allows us to initialize the GAM database with its metadata (repository 1 only).
- Import: Imports a package with its configurations and data.
- Update GAM: Updates the version of the GAM database. If a reorganization is needed, it must be done first to then run this action of the tool.
- Help: Like any command, with this we will see the actions available in the tool.
- Export: Exports the data of a GAM database and stores it in a .gpkg package.
- Get Connections: Gets the connections grouped by Repository. This function is useful to obtain the GUIDs of our repositories and the connection names that are sent as parameters in the UpdateConnectionFile option.
- This last one updates or creates the connection.gam file with the connection data that it obtains.
- xml_config_file: This is a special flag that only receives an XML file in which all the parameters to be entered in the tool are loaded, including the desired action.

Finally, we have Generate XML, which generates an example XML and displays it in the standard output. This XML can be used as input for the tool, changing the corresponding tag values.

Inside each action of course there are several flags to include. You can find more details about them and all the required documentation in the GeneXus wiki.


First deployment

Initialize the database

ReorganizationScript

Initialize

```
C:\Models\kb_name\NetCore\web\bin> dotnet agamdeploytool.dll -initialize -admin_name gamadmin -admin_pass gamadmin123
```

-xml_config_file

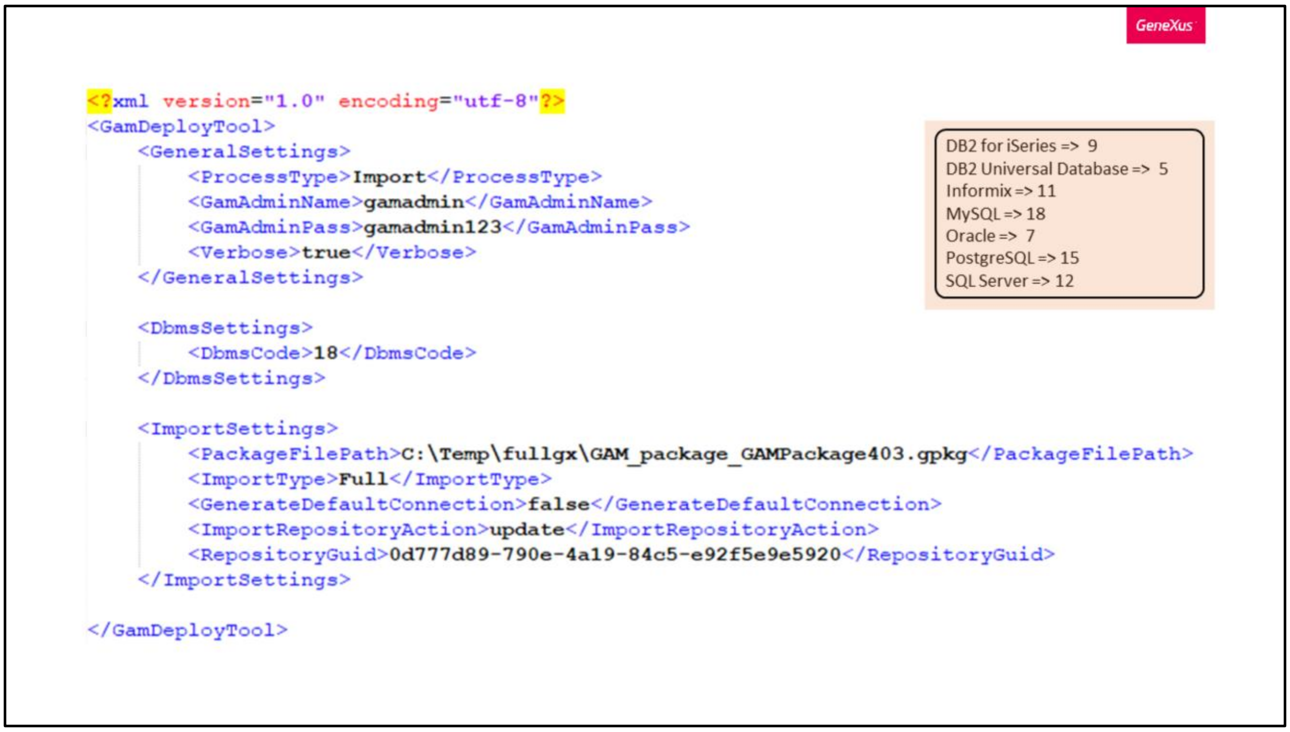
Now let's see how we should deploy GAM and its data for the first time and for subsequent deployments.

Let's start with the first deployment.
Of course, we will use the GAM Deploy Tool.

At the beginning of the video, we saw that GAM provided different scripts in the GeneXus installation folder. Since in this case we want to initialize the database, we will run the ReorganizationScript. With it, we will create the empty GAM database.
We also saw the different options provided by the tool we are looking at, where one of them is Initialize. With this we initialize the GAM database as follows.
When executing it, we must indicate the username and password of GAM.

As a side note, this and all the other examples are based on .NET, which is what will be used in the practice exercises.

In addition, we can use the following flag, where we can indicate the path to an XML file that has all the parameters configured. If we set this flag, all the others are automatically ignored and only the parameters of the file are taken into account. The following is an example of a file for this flag.



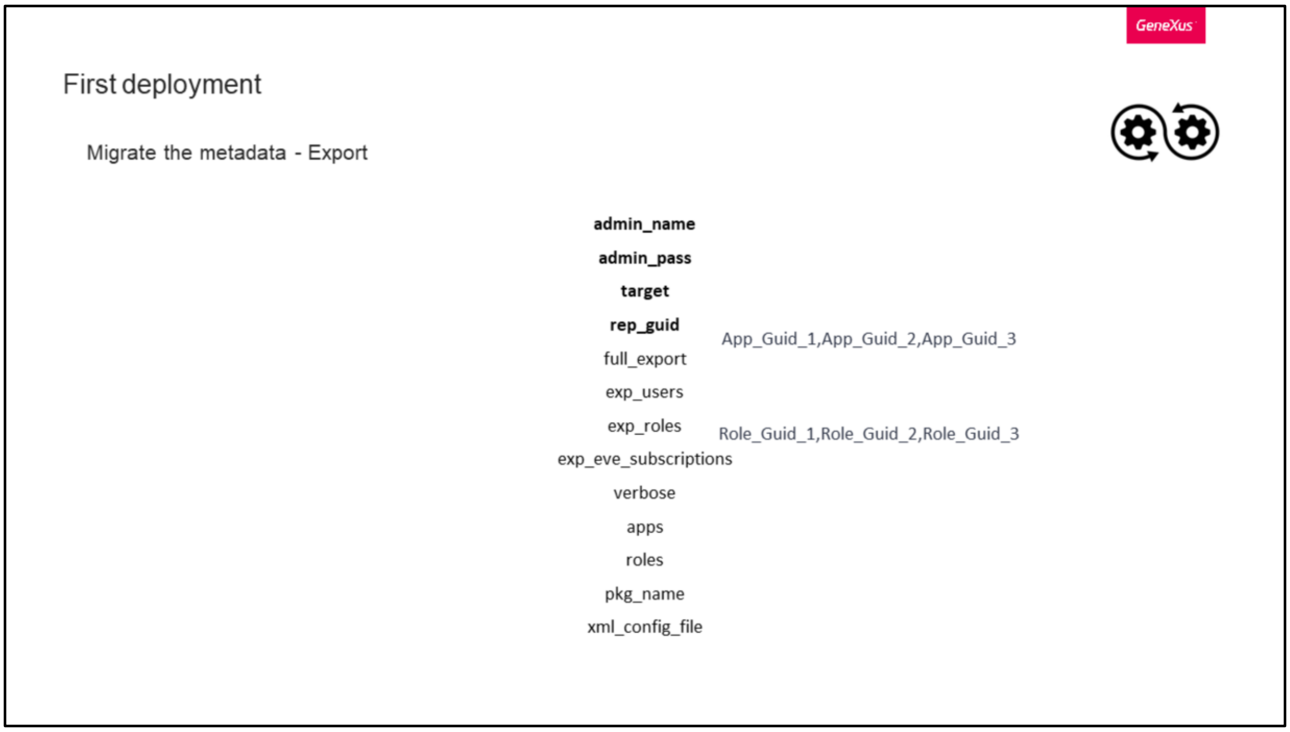
```xml
<?xml version="1.0" encoding="utf-8"?>
<GamDeployTool>
    <GeneralSettings>
        <ProcessType>Import</ProcessType>
        <GamAdminName>gamadmin</GamAdminName>
        <GamAdminPass>gamadmin123</GamAdminPass>
        <Verbose>true</Verbose>
    </GeneralSettings>

    <DbmsSettings>
        <DbmsCode>18</DbmsCode>
    </DbmsSettings>

    <ImportSettings>
        <PackageFilePath>C:\Temp\fullgx\GAM_package_GAMPackage403.gpkg</PackageFilePath>
        <ImportType>Full</ImportType>
        <GenerateDefaultConnection>false</GenerateDefaultConnection>
        <ImportRepositoryAction>update</ImportRepositoryAction>
        <RepositoryGuid>0d777d89-790e-4a19-84c5-e92f5e9e5920</RepositoryGuid>
    </ImportSettings>

</GamDeployTool>
```

DB2 for iSeries => 9
DB2 Universal Database => 5
Informix => 11
MySQL => 18
Oracle => 7
PostgreSQL => 15
SQL Server => 12

In the ProcessType tag we have these values: Import, Export, GenerateConnectionFile, RestartGamDb, and UpdateSchema.

The database connection settings contain the DBMS code. The following values are there according to the database system being used.

Finally, we have the configuration to execute an Import. Obviously, this is only taken into account if ProcessType has the Import value.
Inside here we first find the path to load the deploy package; then the type of import (if it is Full or Custom); a Boolean to indicate whether to generate a default connection for the repository, and a flag that can indicate if you want to update an existing repository (as it is now with the value update; if we want to create a new repository, we should set the value CreateNew). Finally, we have a repository identifier that would apply to the repository that we want to update if the previous flag is set with the update value.



First deployment

Migrate the metadata - Export

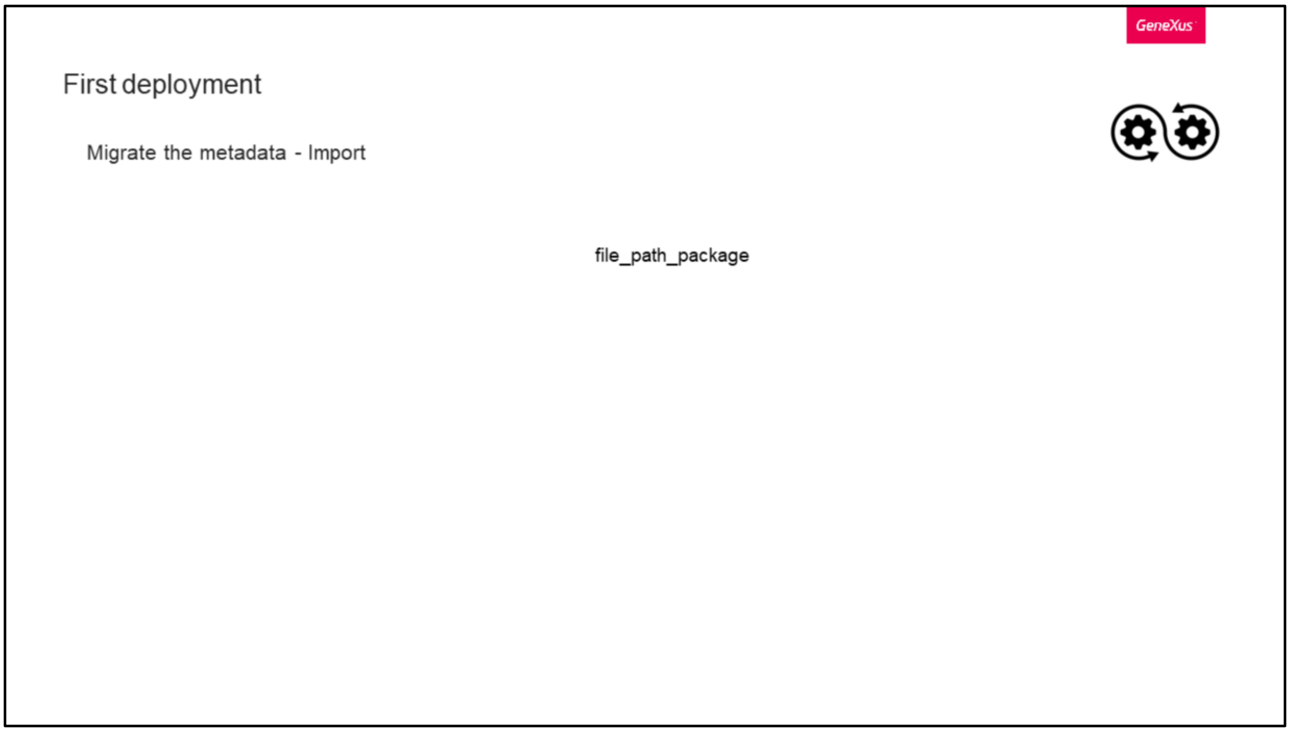
admin_name	
admin_pass	
target	
rep_guid	App_Guid_1,App_Guid_2,App_Guid_3
full_export	
exp_users	
exp_roles	Role_Guid_1,Role_Guid_2,Role_Guid_3
exp_eve_subscriptions	
verbose	
apps	
roles	
pkg_name	
xml_config_file	

Once Initialize is run, we will have the default repository created. But now we have to pass our own repositories, together with users, roles, etc.
To do this, we have to first export the data and then import it into the Production or Pre-Production environment to which we want to migrate the data.

In addition to the username and password, in target we will specify the export destination directory, and in rep_guid we will enter the identifier of the repository to export.
Let's look at the following fields:

- full_export: Boolean; when true, it indicates that the export is complete, including users, roles, and applications.
- exp_users, roles, and eve_subscriptions: will indicate if we want to export users, roles, and event subscriptions, respectively. This flag is only taken into account if full_export has the value False.
- apps and roles: will contain the list of applications and roles that we want to export. They must have the following format.
- pkg_name: is the name of the package we are exporting,
- and xml_config_file, as we said, is an XML with all these parameter configurations.

Something to highlight is that the flags in bold are mandatory to make the export.



First deployment

Migrate the metadata - Import

file_path_package

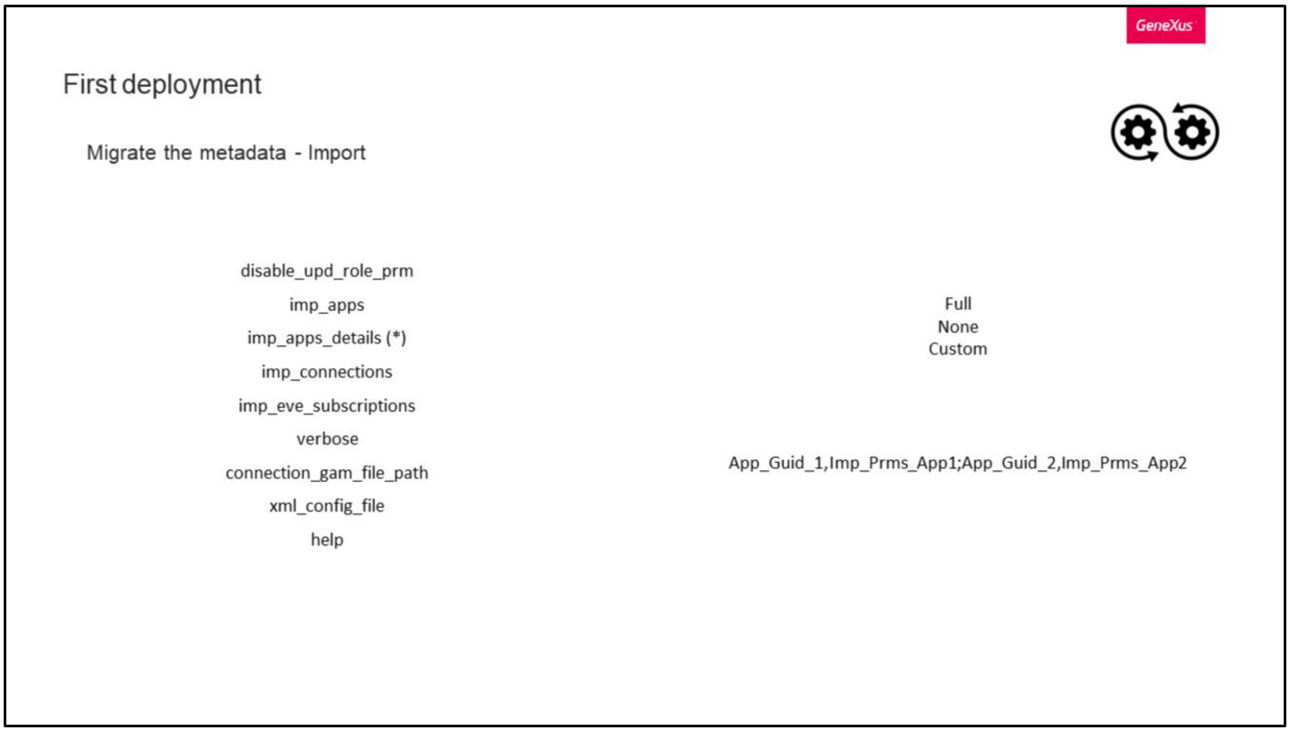
After doing the Export, we have to do the Import. Let's see the available flags for this.

In file_path_package we must enter the path of our previously exported package.
Then we have the administrator data; in this case, in addition to the name and password, we can enter the username and role identifier of the administrator.
upd_rep is a Boolean to indicate if we are going to update the existing repository; if so, in the following flag we indicate its identifier. Otherwise, we use the following flags:

- new_rep_create is a Boolean that indicates if a new repository is going to be created. Then we have the name, namespace, identifier, name, and password of the new repository administrator, and username and password of the repository connection.

Also, there are all those that indicate what we want to import, which correspond to the types of authentication, security policies, users, and roles. All of them are Boolean. In case we want to import everything, we can use the following flag which is imp_full. (It is only valid from GeneXus 16 U6).

As in the previous slide, the flags in bold are mandatory.
The ones with an asterisk are also mandatory but only when the new_rep_create flag has the value True.



First deployment

Migrate the metadata - Import

disable_upd_role_prm

imp_apps

imp_apps_details (*)

imp_connections

imp_eve_subscriptions

verbose

connection_gam_file_path

xml_config_file

help

Full
None
Custom

App_Guid_1,Imp_Prms_App1;App_Guid_2,Imp_Prms_App2

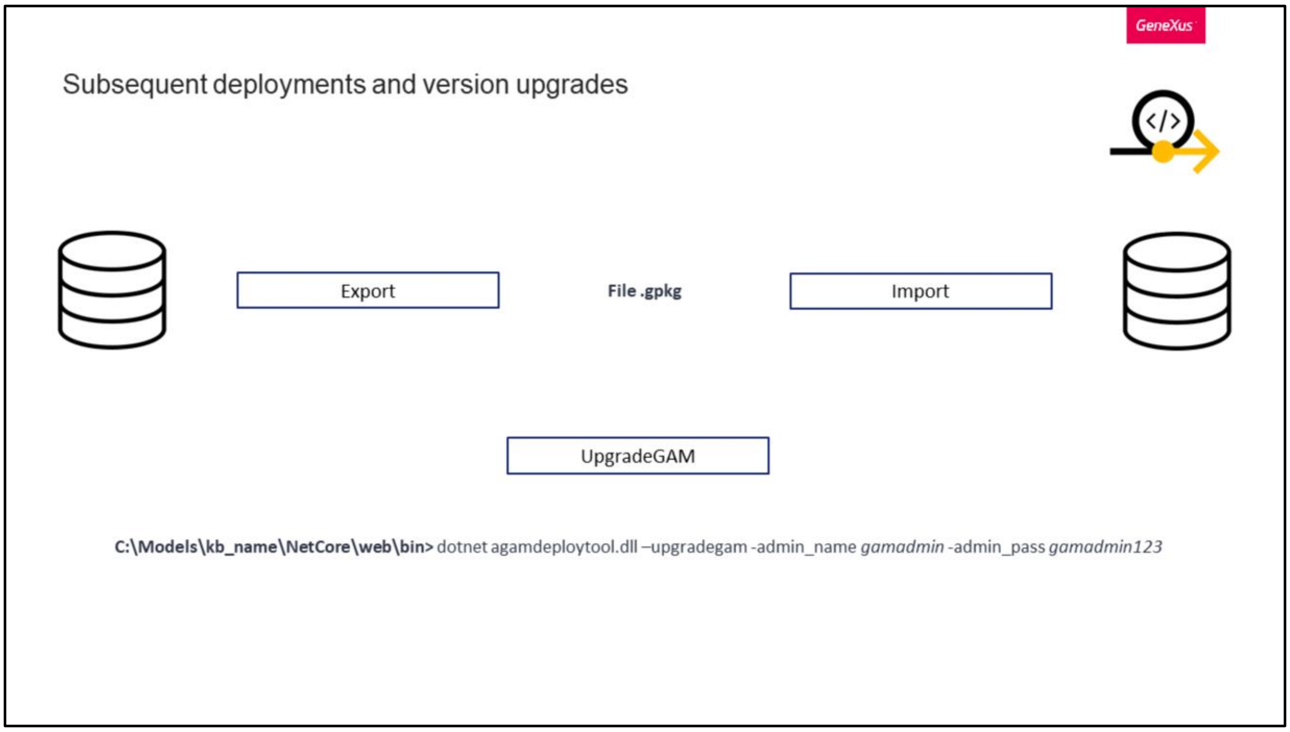
Continuing with flags, with disable_upd_role_prm we indicate if the permissions of the roles that already exist in the Database should be imported; it is set to False by default.
imp_apps is the level with which the applications are imported, and for this we have the following values:

- Full, where all applications are imported with all permissions.
- None, which indicates that nothing is imported in relation to the applications.
- Or custom, where they are configured according to the imp_apps_details flag.

This last flag will represent the list of application identifier and Boolean pairs indicating if the permissions of that application are imported.
Then we have imp_connections, which imports the connections of the package. For new repositories, a new connection is always created regardless of the value of this flag; however, in case of an update, the connections are not updated unless they have the same connection username as an existing connection.
To close the import part, we find imp_eve_subscriptions, which will import event subscriptions.

Of the rest of the flags, we can highlight connection_gam_file_path, with which we will indicate the destination where the connection.gam file will be generated.

The imp_apps_details flag has an asterisk because it is mandatory only when the imp_apps flag has the Custom value.



Subsequent deployments and version upgrades

```
C:\Models\kb_name\NetCore\web\bin> dotnet agamdeploytool.dll –upgradegam -admin_name gamadmin -admin_pass gamadmin123
```

Once the first deployment is done, let’s see how to do the next ones, or how to migrate from one GAM version to another.

If we only need to update our own metadata, we must perform the same process as before: first Export, which will bring everything we need from the database and generate a .gpkg file, and then Import, obviously selecting what we want to migrate through the different options provided by the functionality.

If we only want to update the GAM version, it is enough to execute the UpgradeGAM action, which will update GAM together with its metadata. Its execution is similar to the initialize action, where the administrator's name and password are required.

GeneXus™
by Globant

training.genexus.com
wiki.genexus.com