

Integracja systemów

Laboratorium 10
Prowadzący: Marek Kowal
(M.Kowal@issi.uz.zgora.pl)

Using Sequence Containers to Organize a Package

Sequence Containers provide a simple and easy method for organizing the flow of a package and can help you divide a package into more manageable pieces. When you first begin exploring the **Sequence Container**, you may think organization is the only benefit it provides. However, to the creative developer, this container's uses go far beyond simple organization. If you know how to use it, it can also grant you the following capabilities:

- **Grouping tasks** so that you can disable a part of the package that's temporarily not needed
- **Narrowing the scope** of a variable to just the container
- **Collapsing and expanding** the container to hide the tasks within
- **Managing the properties of multiple tasks** in one step by setting the properties of the container
- **Using one method to ensure that multiple tasks execute successfully** before the next task executes
- **Creating a transaction** across a series of data-related tasks, but not on the entire package
- **Creating event handlers** on a single container so that you can send an e-mail if anything inside one container fails, and perhaps even page yourself if anything else fails

To add a **Sequence Container** to a **package**, drag and drop the **Sequence Container** in the design pane just like you would any other task. To have a task as part of the container, just **drag the task** within the outlined box. Once tasks have been placed inside the **Sequence Container**, they can be connected by precedence constraints only to other tasks within the same container. If you attempt to connect a task inside the container to one outside, you receive an error.

1. Cel ćwiczenia

You explore how **Sequence Containers** can be used inside a package. Create a package with **Sequence Containers** and test different uses of the container. Just use **Script Tasks** to test inside the containers because **Script Tasks** do not require any configuration.

2. Przebieg ćwiczenia

- 1) Create a new package

- 2) Drag **three Sequence Containers** onto your designer and then place **three Script Tasks** inside each container. Connect the precedence constraints from each container to make your package look like Fig. 1. Feel free to run this package as is because the Script Task requires no further configuration. The package will complete successfully without changing any data.

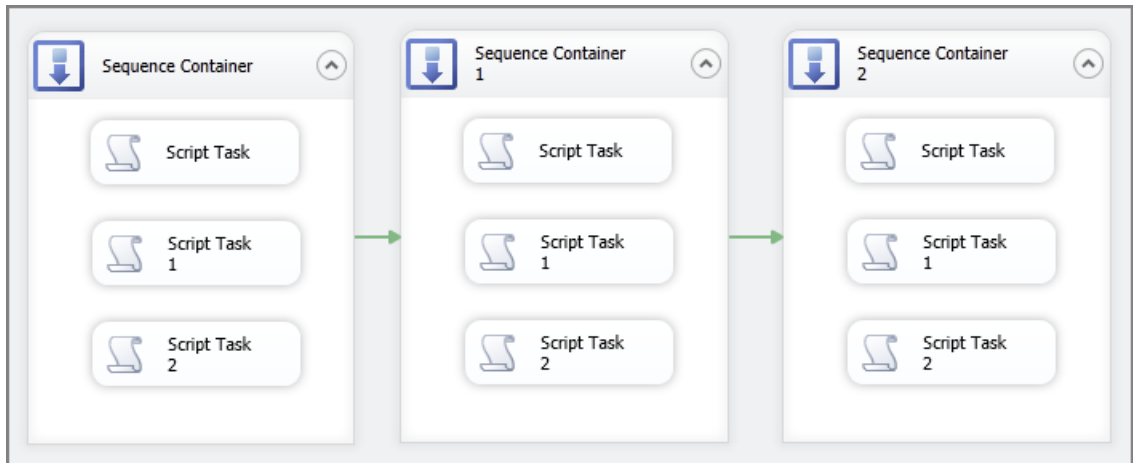


Fig. 1

- 3) The next several steps will help you better understand the benefits and limitations of this container. First, attempt to connect the precedence constraint from any **Script Task** inside **Sequence Container 1** to any other object in the package. You will receive the error shown in Fig. 2 because objects inside a **Sequence Container** cannot be connected to any component outside the container.

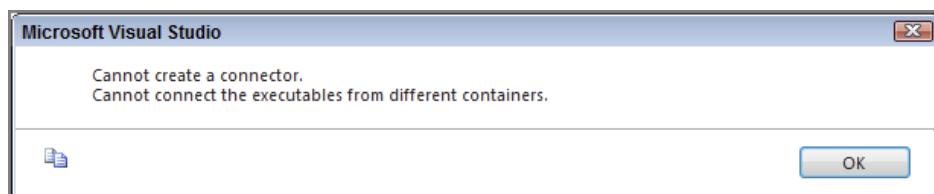


Fig. 2

- 4) With individual tasks, you can execute a single task while excluding the rest of the package. You can also do this with entire containers. **Right-click** the **first container** and **click Execute Container** to execute just the tasks that are inside the container. Notice that just the first container executes while the rest of the package remains inactive. Once you're ready to go to the next step, click the **stop debugging button** to continue.
- 5) **Containers** also enable you to scope variables exclusive to the contents of the container. Click once on **Sequence Container 2** and open the **Variables window**. **Create a variable** called **Lesson** that has a scope set to **Sequence Container 2**.

- 6) Next, **right-click and disable Sequence Container 1** (the container in the middle) and then run the package. The results in Fig 3 demonstrate how **Sequence Containers** enable you to disable entire sections of a package with the container. The outer two containers should have green checkmarks and the middle container is gray, indicating it is disabled. Stop debugging the package.

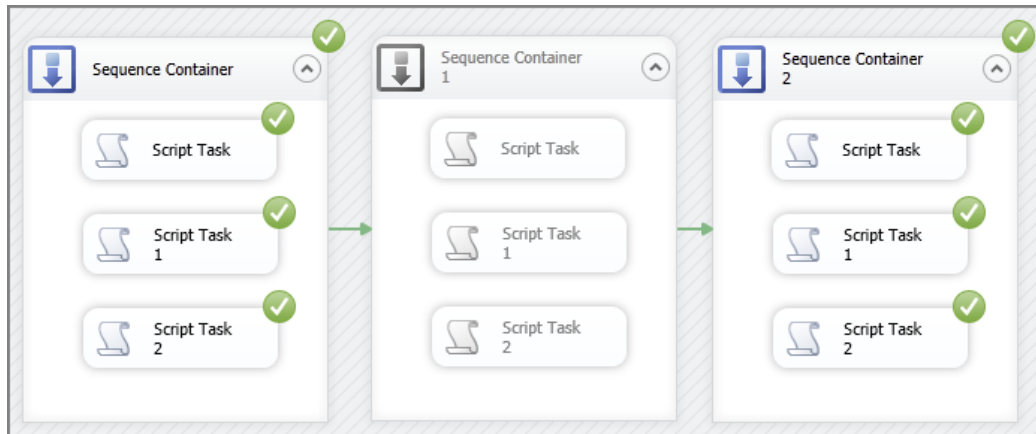


Fig. 3

- 7) Finally, you can collapse a container simply by clicking the arrow pointing upward next to the container name. Fig. 4 shows all three containers collapsed. This action does not change how the package runs, but just hides the content. To expand the containers again, click the arrows that are now pointed down.

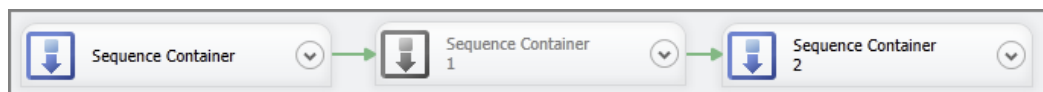


Fig. 4

Using For Loop Containers to Repeat Control Flow Tasks

Loops are a great tool that can be used in many programming languages. They provide a way to **iterate over selected code** repeatedly until it is **conditionally told to stop**. For example, in T-SQL a **While loop** is constructed so that a statement is repeated until a **boolean expression** evaluates as false. The **For Loop Container** in SSIS works much the same way.

A common reason that you might use the **For Loop Container** is if you have a set of **Control Flow Tasks** that need to be run **until a condition has been met**. For example, in this lesson, you use a **For Loop Container** to check to see if a table has any rows of data. If the table has no records, the loop continues to iterate until it finds that at least one row is present.

When you use a **For Loop Container**, you first **create a variable** that the **container** can use to **store an expression value** of **how many times the container has completed a run**. The container runs **three separate expressions** at run time to formulate the loop. These three expressions require only one variable because the expressions change the same variable value each time they evaluate.

To configure the **For Loop Container**, drag it on the design surface and **double-click the top** portion of the container **to open the editor**. Here you see the ***InitExpression***, ***EvalExpression***, and ***AssignExpression*** properties:

- ***InitExpression***—This expression sets the initial value of the variable, so if you want the package to perform the loop a set number of times, you will likely start the variable at 1.
- ***EvalExpression***—This expression tells the loop when to stop. This must be an expression that always evaluates to True or False. As soon as it evaluates to False, the loop stops.
- ***AssignExpression***—This expression changes a condition each time the loop repeats. Here you tell the loop to increment or decrement the number of runs the loop has completed.

1. Cel ćwiczenia

You explore how the **For Loop Container** uses expressions to determine how many times to run **Control Flow** items. The package you create continuously loops until data has been loaded in a table called **ForLoop**. After this lesson, you will understand how the **For Loop Container** is used to iterate through **Control Flow Tasks** a set number of times.

2. Przebieg ćwiczenia

- 1) Open **SQL Server Management Studio** and create the ***ForLoop*** table in the ***AdventureWorks2017*** database with the following statement:

```
CREATE TABLE ForLoop
(
    ID int NOT NULL IDENTITY (1, 1),
    Name varchar(50) NULL
) ON [PRIMARY]
```

- 2) After this table is created, create a **new package**.
- 3) Create a new **OLE DB Connection Manager** that uses the **AdventureWorks2017** database.
- 4) Next, open the **Variables window** by right-clicking on the design surface and selecting **Variables**. Click the **Add Variable** button to add a new variable named `intCounter` with an `Int32` data type, as shown in Fig 1.

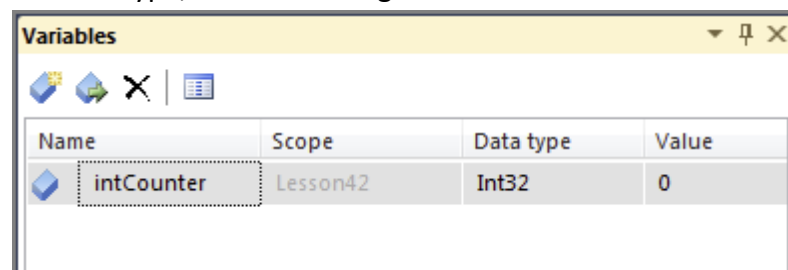


Fig. 1

- 5) Drag an **Execute SQL Task** into the **Control Flow** and open its editor by double-clicking the task.
- 6) On the **General tab**, select **AdventureWorks2017** as your connection, change the **ResultSet** to **Single row**, and type the following query into the **SQLStatement** property:

```
declare @RecordsInserted int
if exists(select Name
from ForLoop)
set @RecordsInserted = 1
else
set @RecordsInserted = 0
select @RecordsInserted as RecordsInserted
```

This query checks to see if the **ForLoop** table has any records, and if it does, it **returns** the number **1**. If no rows are in the table, it **returns** the number **0**. Fig. 2 shows the editor after these changes have been made.

- 7) Select the **Result Set tab** and click **Add**. In the **Result Name** column, change the default **NewResultName** to **RecordsInserted** and keep the **Variable Name** column the default of **User::intCounter**. After you make these changes, the results of

the **Execute SQL Task** will be loaded into the **variable**. After setting up this page, you are done with this editor, so click OK

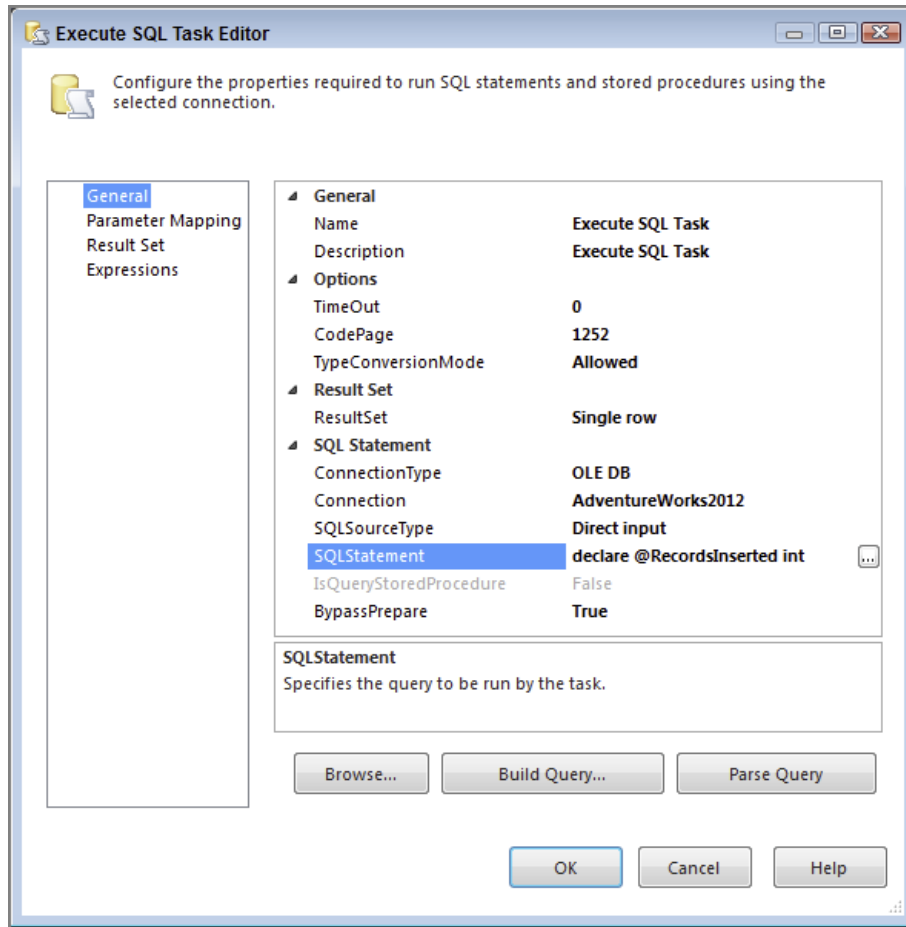


Fig. 2

8) Now drag a **For Loop Container** in the **Control Flow** and place the **Execute SQL Task** inside the new container.

9) Open the **For Loop Editor** and make the following changes:

- **InitExpression**—@intCounter = 0
- **EvalExpression**—@intCounter == 0
- **AssignExpression**—Leave blank

Once these properties have been filled, **click OK**. Your screen should look like Fig. 3.

10) The package is now complete and ready to run. After you execute this package, you will see that because there is currently no data in the **ForLoop** table, the package will continue to run until new data is inserted into the table

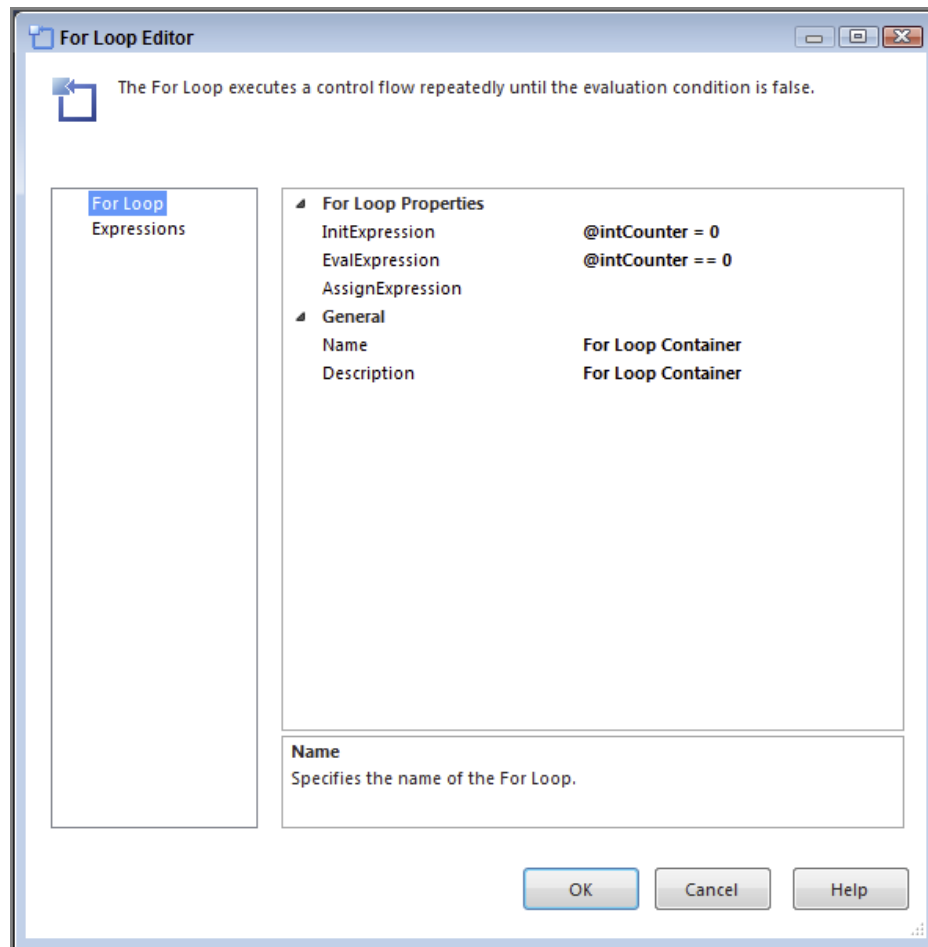


Fig. 3

- 11) While the **package is running**, open **SQL Server Management Studio** and run the following statement to **load some records** in the ***ForLoop*** table:

```
INSERT INTO ForLoop
(Name)
Select Distinct
LastName
From Person.Person
```

This should cause the package to complete because the loop you created was waiting for new records to be inserted before it could complete. Figure 4 shows what the completed package should look like after these rows have been inserted.

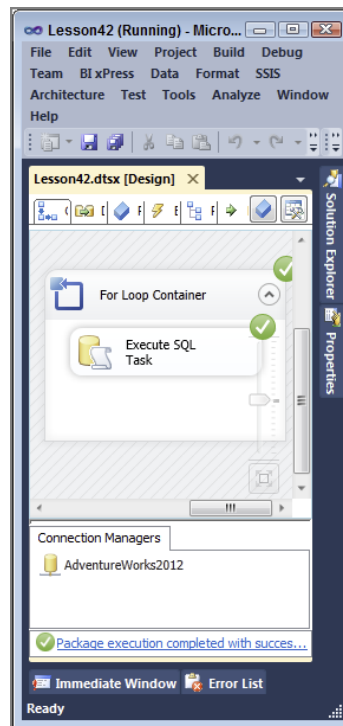


Fig. 4

Using the Foreach Loop Container to Loop Through a Collection of Objects

The **Foreach Loop Container** is a very powerful and very useful tool for **repeating Control Flow items**. It is often used when you have a **collection of files** to which you want to **apply the same changes**. If you provide the directory for a set of files, the **Foreach Loop Container** can apply the same **Control Flow tasks** to each file. You might ask yourself, how is this different from the **For Loop Container**? The easy answer is that the **For Loop** iterates through the content of the container a number of times you define or you define with an expression, whereas the **Foreach Loop Container** iterates through **its content** as many times as it takes to effect the full collection.

The configuration of the **Foreach Loop Container** can differ depending on which *enumerator* you decide to use. An *enumerator* specifies the collection of objects that the container will **loop through**. **All tasks inside the container will be repeated for each member** of a specified *enumerator*. The **Foreach Loop Editor** can significantly change depending on what you set for this option:

- **Foreach File Enumerator**—Performs an action for each file in a directory with a given file extension
- **Foreach Item Enumerator**—Loops through a list of items that are set manually in the container
- **Foreach ADO Enumerator**—Loops through a list of tables or rows in a table from an ADO recordset
- **Foreach ADO.NET Schema Rowset Enumerator**—Loops through an ADO.NET schema
- **Foreach From Variable Enumerator**—Loops through a SQL Server Integration Services (SSIS) variable
- **Foreach Nodelist Enumerator**—Loops through a node list in an XML document
- **Foreach SMO Enumerator**—Enumerates a list of SQL Management Objects (SMO)

To configure the **Foreach Loop Container**, drag it on the design surface and **double-click** the top portion of the container **to open the editor**. Click the **Collection tab** to choose which type of *enumerator* you want to use, as shown in Fig. 1. For example, say you want to loop through a **directory to load a group of flat files** to a table, so you choose the **Foreach File Enumerator**. Then you must specify what folder directory the files are in and what kind of file extension they have. Assume the flat files are `.txt` files, so in the Files box `*.txt` is used to bring back only the text files in the directory. Last, because the flat files each have a different name, you can use the **Variable Mappings tab** to **dynamically change a variable value** for each **iteration of the loop**. That **variable** then can pass the **correct file name** to the **Flat File Connection** with an expression.

Another commonly used enumerator is the **Foreach ADO Enumerator**. This *enumerator* is handy for looping through a **set of records** and executing every task inside the container for

each record in that set. For example, you want to **run each task** in your package for **every database on a server**. With the **Foreach ADO Enumerator**, you could **loop through a table** that **lists all the database names** on your server and **dynamically change a connection manager's database name for each iteration** of the loop.

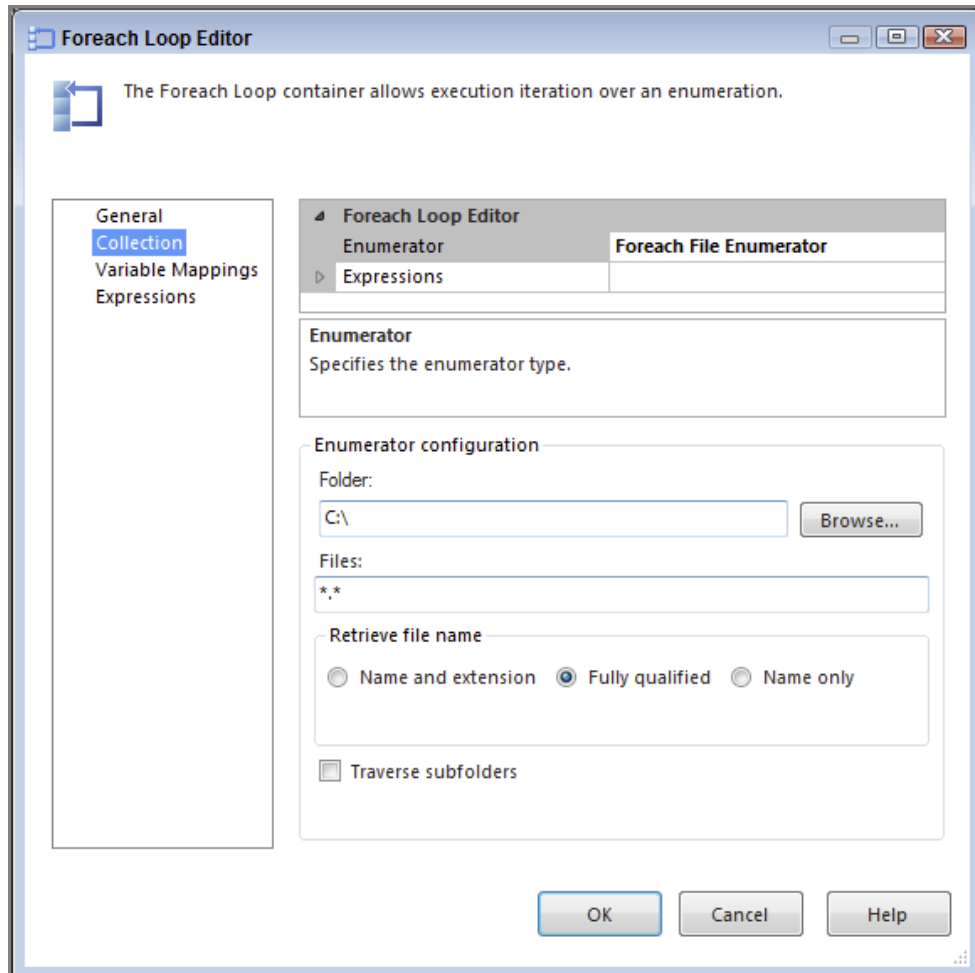


Fig. 1

3. Cel ćwiczenia

You create a package that uses the most common type of *enumerator*, the **Foreach File Enumerator**, to loop through a **collection of flat files** and **load them to a table**. After this lesson, you will understand how to use the **Foreach Loop Container** to loop through a collection of files and load each to a table.

4. Przebieg ćwiczenia

- 1) Create a new package
- 2) Drag a **Data Flow Task** onto your designer and name it **DFT - Load Flat Files**.

- 3) Create a new **Flat File Connection Manager**, name it **File Extract**, and point it to `File 1.txt`. Also, check the **Column names** in the **first data row** option and go to the **Columns** page to ensure all the columns are defined properly; then **click OK**
- 4) In the **Data Flow**, bring a new **Flat File Source** over and name it **File Extract**. Open the **Flat File Source Editor** by double-clicking the **Flat File Source** and make the connection manager the newly created **File Extract**. Then click **OK**.
- 5) Next, create another connection manager, this time an **OLE DB Connection Manager**, using the **AdventureWorks2017** database.
- 6) Bring an **OLE DB Destination** in the **Data Flow** and connect the **Data Flow** path from the source to it. Open the editor and set the **OLE DB Connection Manager** to **AdventureWorks2017**. Create a **new table** with the following SQL statement by clicking **New** next to the table selection drop-down box:

```
CREATE TABLE [ForEachLoop] (  
    [Name] varchar(50) ,  
    [State] varchar(50)  
)
```

Ensure the columns are mapped correctly; then click OK. Your Data Flow should now look like Fig. 2.

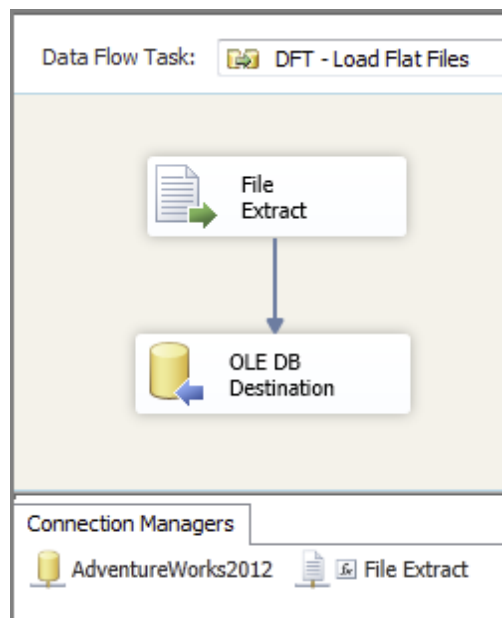


Fig. 2

- 7) Your package is now set up to run **just one file**, but because **you have four**, you now go back to the **Control Flow** and drag over a **Foreach Loop Container**.

- 8) Place the **Data Flow Task** inside the **Foreach Loop Container**; then open the **Foreach Loop Editor** by double clicking the top banner portion of the container. On the **Collections** tab, select **Foreach File Enumerator** from the **Enumerator property drop-down box**. The **Foreach File Enumerator** is the default when you open the editor.
- 9) Now, set the **Folder property** to your chosen directory and the **Files property** to `*.txt` because you want to **bring back all the text files** in the **directory**. Everything else you can leave as the **default**. After you make these changes, your editor should look like Fig. 3.
- 10) On the **Variable Mappings** tab create a **new variable** called `strFlatFileLocation` by selecting **<New Variable...>** from the **Variable drop-down box**. Fig. 4 shows the **Add Variable** dialog box.

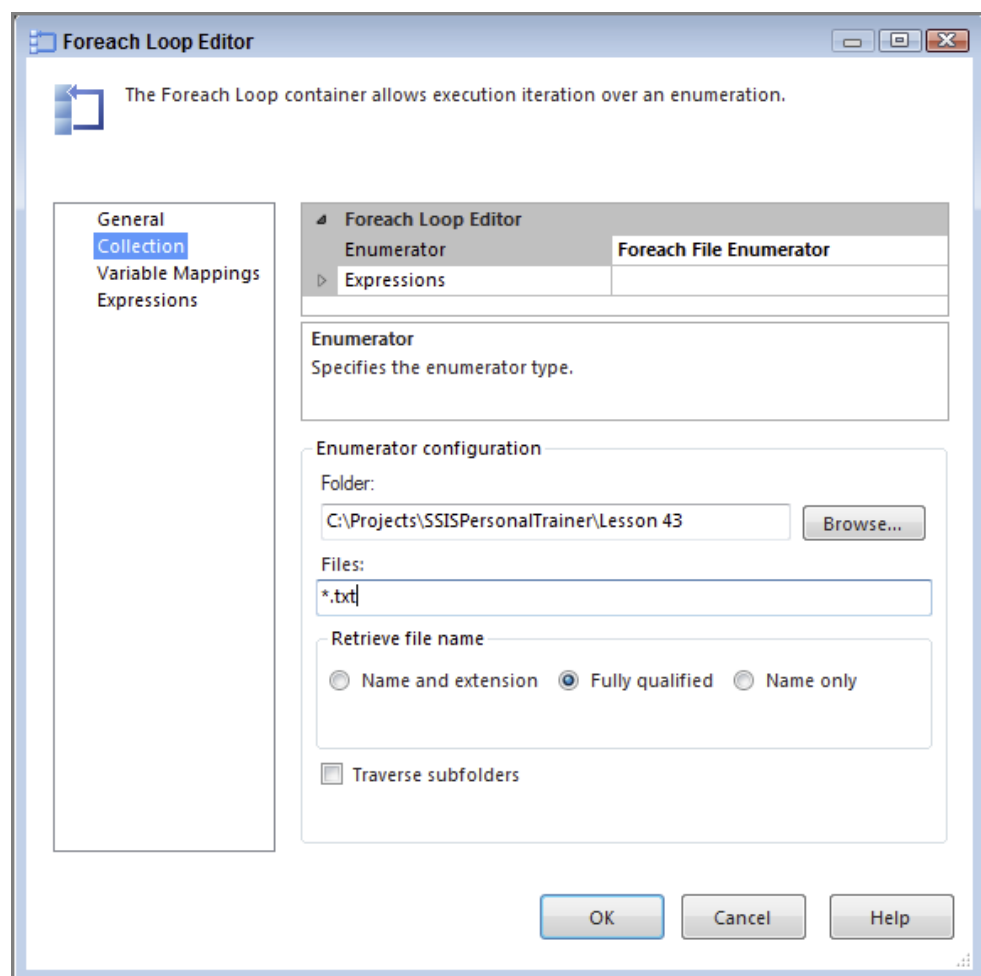
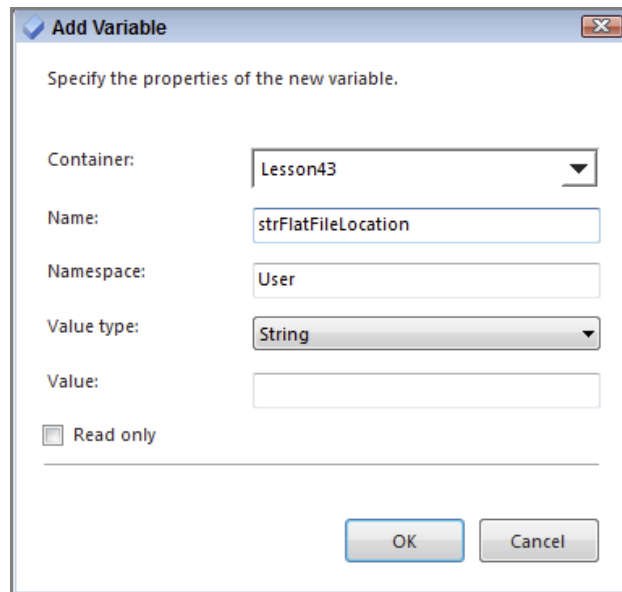


Fig. 3



Add Variable

Specify the properties of the new variable.

Container: Lesson43

Name: strFlatFileLocation

Namespace: User

Value type: String

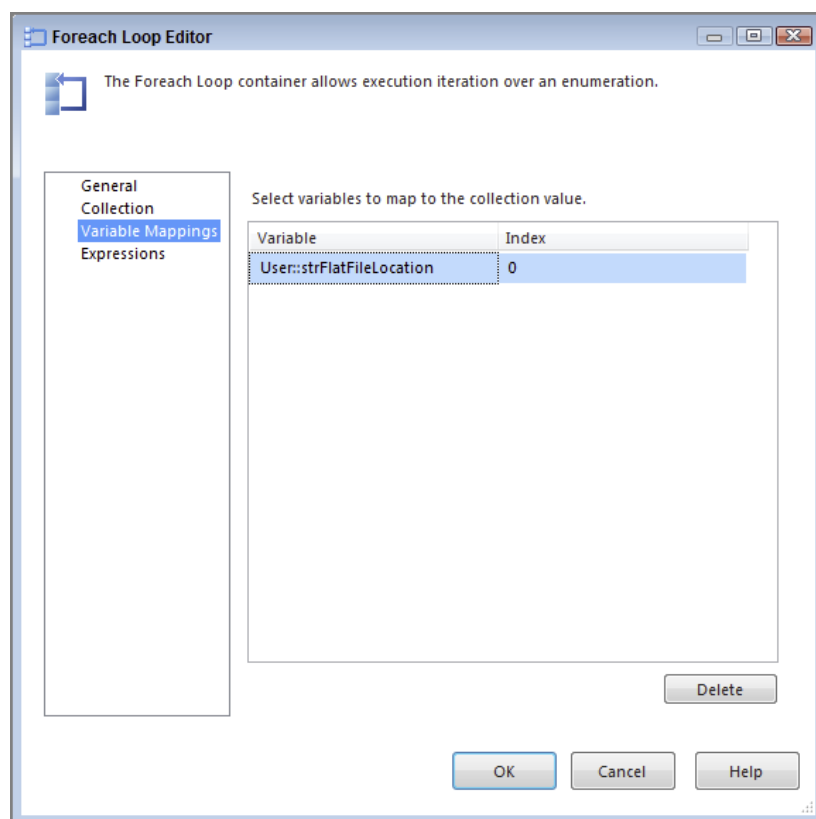
Value:

☐ Read only

OK Cancel

Fig. 4

11) This **variable's value** will change to **the current file** it is loading **each time** the container runs. In this specific case, **after File 1.txt is completed**, the container will **automatically change the variable's value to the next filename**. After the variable is created, **click OK**. The **Variable Mappings** tab should look like Fig. 5. **Click OK again to return to the Data Flow**.



Foreach Loop Editor

The Foreach Loop container allows execution iteration over an enumeration.

General
Collection
Variable Mappings
Expressions

Select variables to map to the collection value.

Variable	Index
User::strFlatFileLocation	0

Delete

OK Cancel Help

Fig. 5

- 12) The last step is to put an **expression** on the **File Extract Connection Manager** that uses the **variable** you just created inside the **Foreach Loop Container**. Select the **File Extract Connection Manager** called **File Extract** from your list of connection managers and **press F4** to bring up the **Properties window**. Click the **ellipsis (...)** next to the **Expressions property** to open the **Property Expressions Editor**. Select **ConnectionString** from the **Property** drop-down and then click the **ellipsis** in the **Expression box**
- 13) In the top left of the **Expression Builder**, expand the **Variables and Parameters** folder and drag **@[User::strFlatFileLocation]** down into the **Expression box**. If you try to click **Evaluate Expression** now, there will be **no result**. Remember that this expression will be populated at run time, so you will see nothing here yet. Your screen should look like Fig. 6. Click **OK twice** to return to the **Data Flow**.
- 14) The **package** is now complete. A successful run will loop through and load all the files in your directory to the **ForEachLoop** table in the **AdventureWorks2017** database. Fig. 7 shows what a successful run should look like.

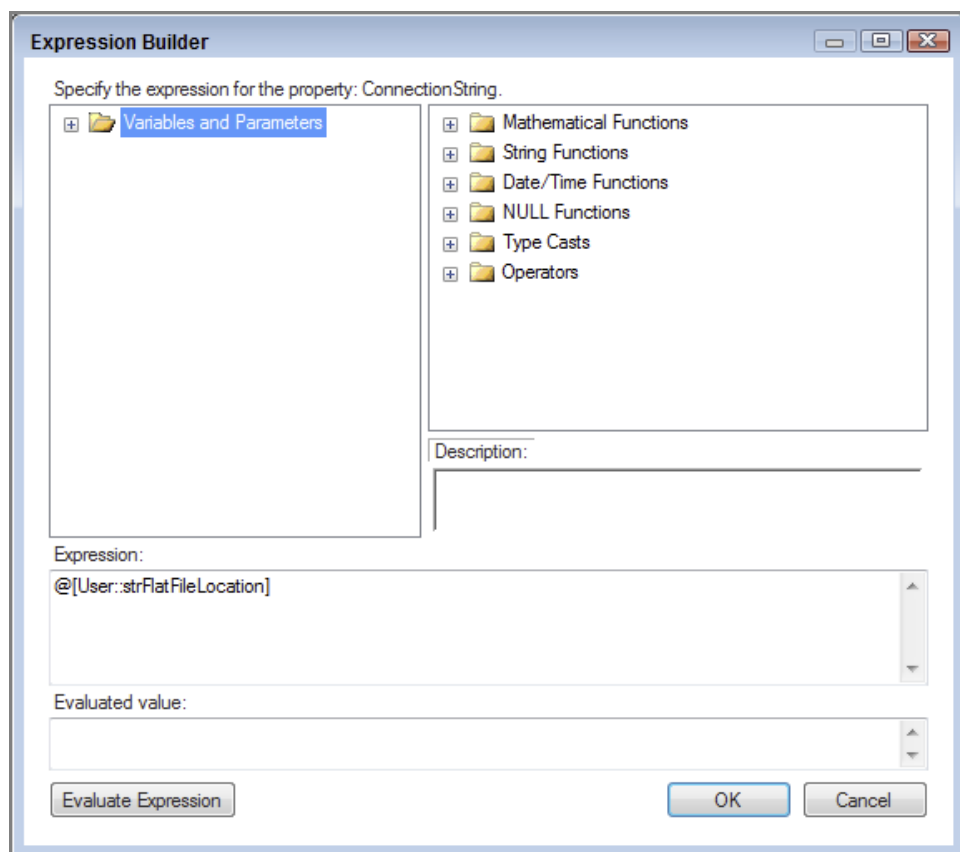


Fig. 6

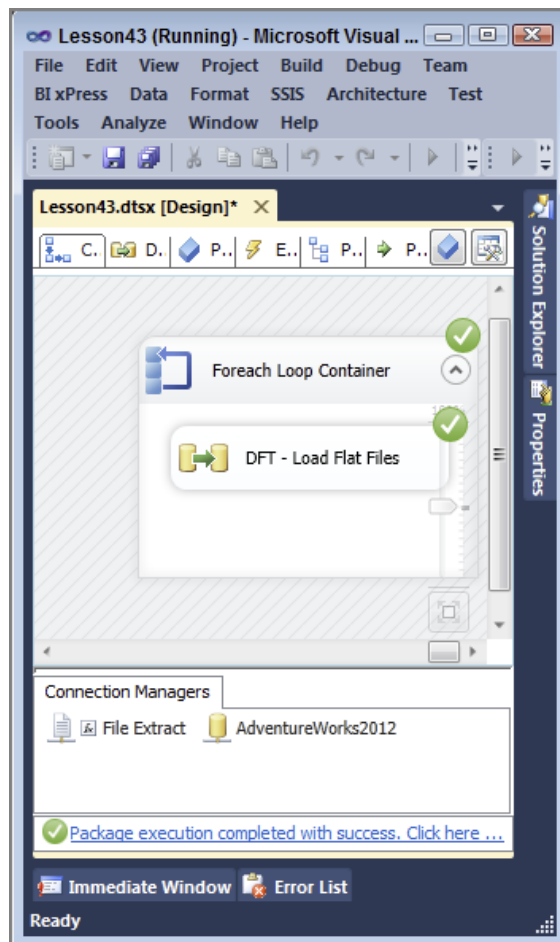


Fig. 7

Bibliografia

- 1) Knight B., Knight D., Davis M, Snyder W. (2013): Knight's Microsoft® SQL Server® 2012 Integration Services 24-Hour Trainer, John Wiley & Sons.
- 2) Knight B., Veerman E., Moss J.M., Davis M., Rock C. (2012): PROFESSIONAL Microsoft® SQL Server® 2012 Integration Services, John Wiley & Sons.
- 3) <http://www.wrox.com/WileyCDA/Section/id-814197.html>
- 4) [https://msdn.microsoft.com/library/ms169917\(SQL.120\).aspx](https://msdn.microsoft.com/library/ms169917(SQL.120).aspx)
- 5) Tok W-H., Parida R. Masson M. Ding X. Sivashanmugam (2012): Microsoft SQL Server 2012 Integration Services, Promise (tłumaczenie j. polski).
- 6) Kimball R. (2004): The Data Warehouse ETL Toolkit. John Wiley & Sons