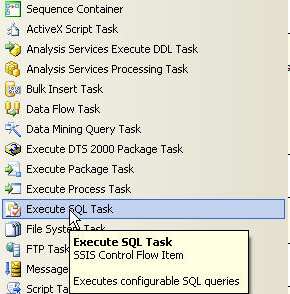
**[The Execute SQL Task](http://www.sqlis.com/sqlis/post/The-Execute-SQL-Task.aspx)**

After looking at these introductory topics we will then get into some examples. The examples will show different types of usage for the task:

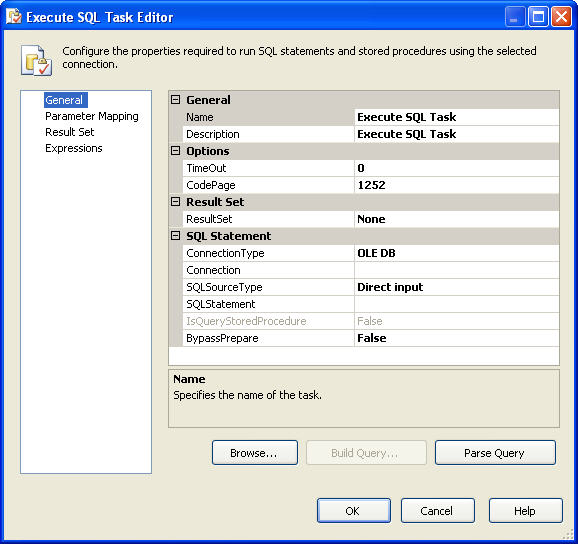
* **Returning a single value from a SQL query with two input parameters.**
* **Returning a rowset from a SQL query.**
* **Executing a stored procedure and retrieveing a rowset, a return value, an output parameter value and passing in an input parameter.**
* **Passing in the SQL Statement from a vari**able.
* **Passing in the SQL Statement from a file.**

**Tour Of The Task**



Now drag the task onto the designer

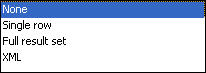
Double click on the task itself to take a look at the custom user interface provided to us for this task. The task will open on the general tab as shown below. Take a bit of time to have a look around here as throughout this article we will be revisting this page many times.



Whilst on the general tab, drop down the combobox next to the *ConnectionType* property. In here you will see the types of connection manager which this task will accept.



As with SQL Server 2000 DTS, SSIS allows you to output values from this task in a number of formats. Have a look at the combobox next to the *Resultset* property. The major difference here is the ability to output into XML.

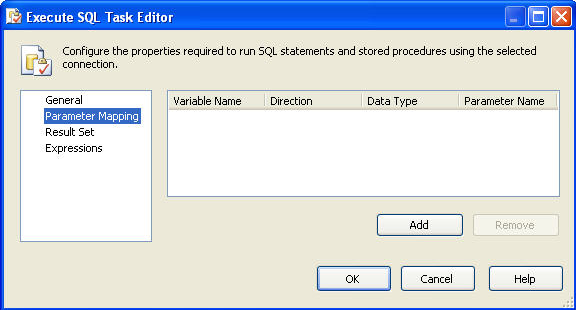


If you drop down the combobox next to the *SQLSourceType* property you will see the ways in which you can pass a SQL Statement into the task itself. We will have examples of each of these later on but certainly when we saw these for the first time we were very excited.

SourceType

Next to the *SQLStatement* property if you click in the empty box next to it you will see ellipses appear. Click on them and you will see the very basic query editor that becomes available to you.

Once you've finished looking around the general tab, move on to the next tab which is the parameter mapping tab. We shall, again, be visiting this tab throughout the article but to give you an initial heads up this is where you define the input, output and return values from your task. Note this is not where you specify the resultset.

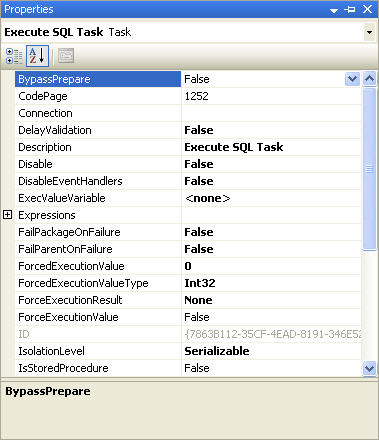


If however you now move on to the ResultSet tab this *is* where you define what variable will receive the output from your SQL Statement in whatever form that is.

For a more detailed discussion of what should be the parameter markers in the SQL Statements on the General tab and how to map them to variables on the Parameter Mapping tab see [Working with Parameters and Return Codes in the Execute SQL Task](http://msdn.microsoft.com/en-us/library/cc280502.aspx).

**Task Properties**

There are two places where you can specify the properties for your task. One is in the task UI itself and the other is in the property pane which will appear if you right click on your task and select *Properties* from the context menu. We will be doing plenty of property setting in the UI later so let's take a moment to have a look at the property pane. Below is a graphic showing our properties pane.



Now we shall take you through all the properties and tell you exactly what they mean. A lot of these properties you will see across all tasks as well as the package because of everything's base structure **The Container**.

**BypassPrepare**

Should the statement be prepared before sending to the connection manager destination (True/False)

**Connection**

This is simply the name of the connection manager that the task will use. We can get this from the connection manager tray at the bottom of the package.

**DelayValidation**

Really interesting property and it tells the task to not validate until it actually executes. A usage for this may be that you are operating on table yet to be created but at runtime you know the table will be there.

**Disable**

Should the task be enabled or not? You can also set this through a context menu by right clicking on the task itself.

**DisableEventHandlers**

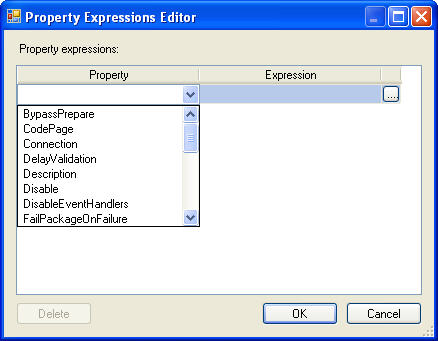
As a result of events that happen in the task, should the event handlers for the container fire?

**ExecValueVariable**

The variable assigned here will get or set the execution value of the task.

**Expressions**

Expressions as we mentioned earlier are a really powerful tool in SSIS and this graphic below shows us a small peek of what you can do. We select a property on the left and assign an expression to the value of that property on the right causing the value to be dynamically changed at runtime.



One of the most obvious uses of this is that the property value can be built dynamically from within the package allowing you a great deal of flexibility

**FailPackageOnFailure**

If this task fails does the package?

**FailParentOnFailure**

If this task fails does the parent container? A task can he hosted inside another container i.e. the For Each Loop Container and this would then be the parent.

**ForcedExecutionValue**

This property allows you to hard code an execution value for the task.

**ForcedExecutionValueType**

What is the datatype of the ForcedExecutionValue?

**ForceExecutionResult**

Force the task to return a certain execution result. This could then be used by the workflow constraints. Possible values are None, Success, Failure and Completion.

**ForceExecutionValue**

Should we force the execution result?

**IsolationLevel**

This is the transaction isolation level of the task.

**IsStoredProcedure**

Certain optimisations are made by the task if it knows that the query is a Stored Procedure invocation. The docs say this will always be false unless the connection is an ADO connection.

**LocaleID**

Gets or sets the LocaleID of the container.

**LoggingMode**

Should we log for this container and what settings should we use? The value choices are UseParentSetting, Enabled and Disabled.

**MaximumErrorCount**

How many times can the task fail before we call it a day?

**Name**

Very simply the name of the task.

**ResultSetType**

How do you want the results of your query returned? The choices are ResultSetType\_None, ResultSetType\_SingleRow, ResultSetType\_Rowset and ResultSetType\_XML.

**SqlStatementSource**

Your Query/SQL Statement.

**SqlStatementSourceType**

The method of specifying the query. Your choices here are DirectInput, FileConnection and Variables

**TimeOut**

How long should the task wait to receive results?

**TransactionOption**

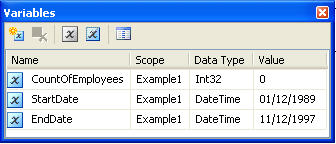
How should the task handle being asked to join a transaction?

**Usage Examples**

As we move through the examples we will only cover in them what we think you must know and what we think you should see. This means that some of the more elementary steps like setting up variables will be covered in the early examples but skipped and simply referred to in later ones. All these examples used the AventureWorks database that comes with SQL Server 2005.

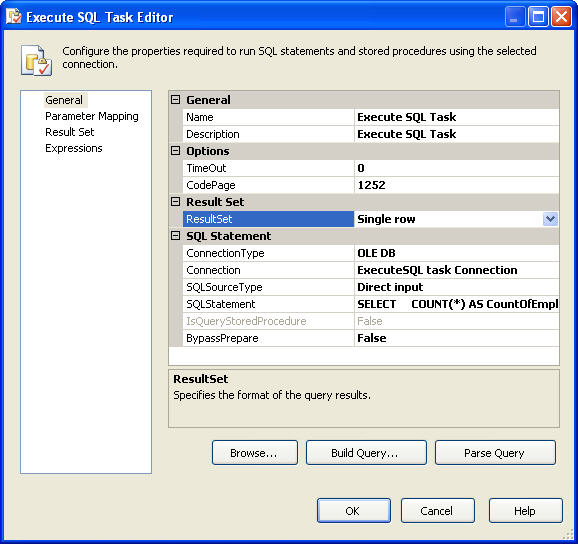
**Returning a Single Value, Passing in Two Input Parameters**

So the first thing we are going to do is add some variables to our package. The graphic below shows us those variables having been defined. Here the *CountOfEmployees* variable will be used as the output from the query and *EndDate* and *StartDate* will be used as input parameters. As you can see all these variables have been scoped to the package. Scoping allows us to have domains for variables. Each container has a scope and remember a package is a container as well. Variable values of the parent container can be seen in child containers but cannot be passed back up to the parent from a child.



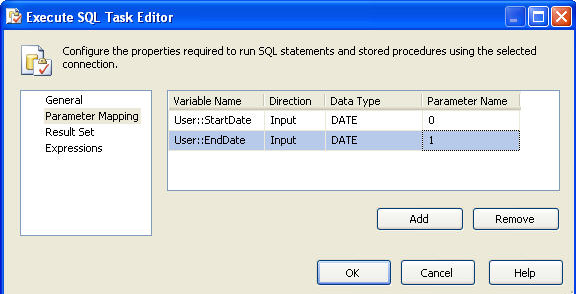
Our following graphic has had a number of changes made. The first of those changes is that we have created and assigned an OLEDB connection manager to this Task *ExecuteSQL Task Connection*. The next thing is we have made sure that the *SQLSourceType* property is set to Direct Input as we will be writing in our statement ourselves. We have also specified that only a single row will be returned from this query. The expressions we typed in was:

SELECT COUNT(\*) AS CountOfEmployees FROM HumanResources.Employee WHERE (HireDate BETWEEN ? AND ?)

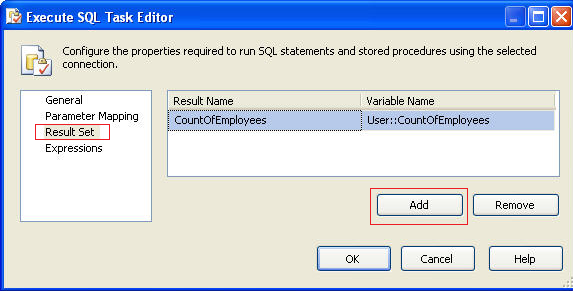


* ResultSet: SingleRow
* SQLStatement: Select count(\*) **as CountOfEmployee** from Employee

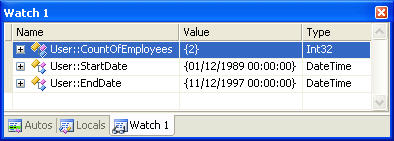
Moving on now to the *Parameter Mapping* tab this is where we are going to tell the task about our input paramaters. We **Add** them to the window specifying their direction and datatype. A quick word here about the structure of the variable name. As you can see SSIS has preceeded the variable with the word **user**. This is a default namespace for variables but you can create your own. When defining your variables if you look at the variables window title bar you will see some icons. If you hover over the last one on the right you will see it says "Choose Variable Columns". If you click the button you will see a list of checkbox options and one of them is namespace. after checking this you will see now where you can define your own namespace.



The next tab, **result set,** is where we need to get back the value(s) returned from our statement and assign to a variable which in our case is **CountOfEmployees** so we can use it later perhaps. Because we are only returning a single value then if you remember from earlier we are allowed to assign a name to the resultset but it must be the name of the column (or alias) from the query.



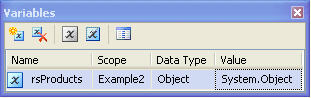
In our package we set a Breakpoint so we can break the package and have a look in a watch window at the variable values as they appear to our task and what the variable value of our resultset is after the task has done the assignment. Here's that window now.



As you can see the count of employess that matched the data range was 2.

**Returning a Rowset**

In this example we are going to return a resultset back to a variable after the task has executed not just a single row single value. One variable of type object.



Here is the statement that will form the soure for our Resultset.

select

p.ProductNumber, p.name, pc.Name as ProductCategoryName

FROM Production.ProductCategory pc

JOIN Production.ProductSubCategory psc

ON pc.ProductCategoryID = psc.ProductCategoryID

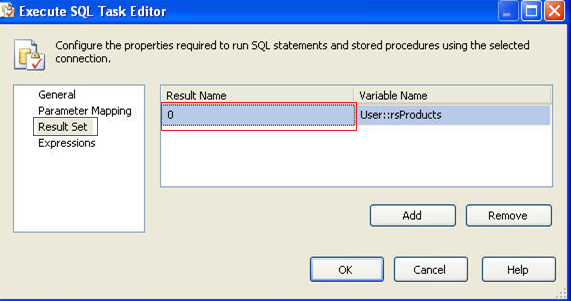
JOIN Production.Product p

ON psc.ProductSubCategoryID = p.ProductSubCategoryID

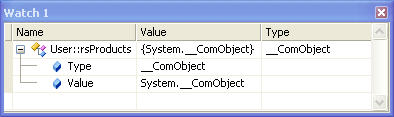
We need to make sure that we have selected **Full result set** as the *ResultSet* as shown below on the task's General tab.

Example2ResultsetOnGeneralTab

Here we need to **Add** our variable defined earlier and map it to the result name of 0 (remember we covered this earlier)

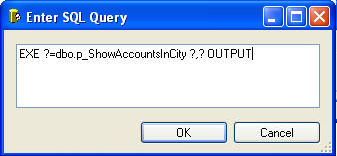


Once we run the task we can again set a breakpoint and have a look at the values coming back from the task. In the following graphic you can see the result set returned to us as a COM object. We can do some pretty interesting things with this COM object and in later articles that is exactly what we shall be doing.

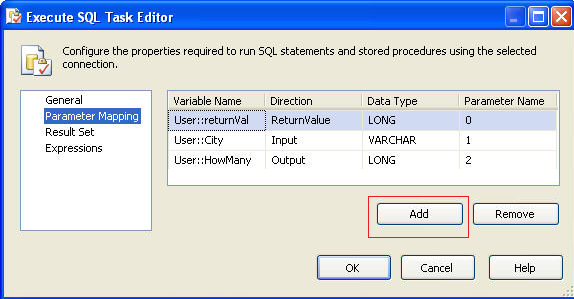


**Return Values, Input/Output Parameters and Returning a Rowset from a Stored Procedure**

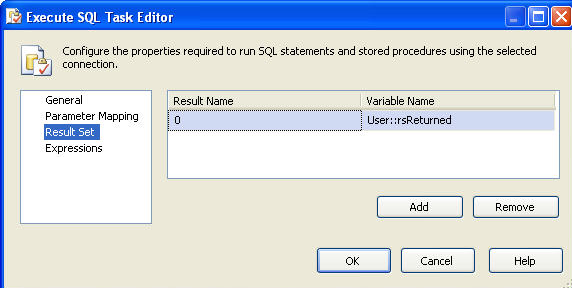
We have already covered in the previous example how to specify the ResultSet to be a Full result set so we will not cover it again here. For this example we are going to need 4 variables. One for the return value, one for the input parameter, one for the output parameter and one for the result set. Here is the statement we want to execute. Note how much cleaner it is than if you wanted to do it using the current version of DTS.



In the *Parameter Mapping* tab we are going to **Add** our variables and specify their direction and datatypes.



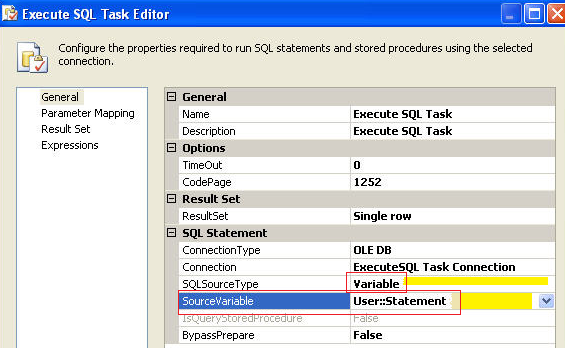
In the *Result Set* tab we can now map our final variable to the rowset returned from the stored procedure.



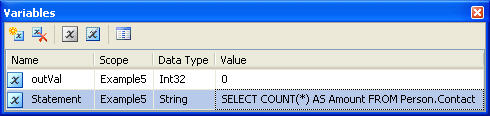
It really is as simple as that and we were amazed at how much easier it is than in DTS 2000.

**Passing in the SQL Statement from a Variable**

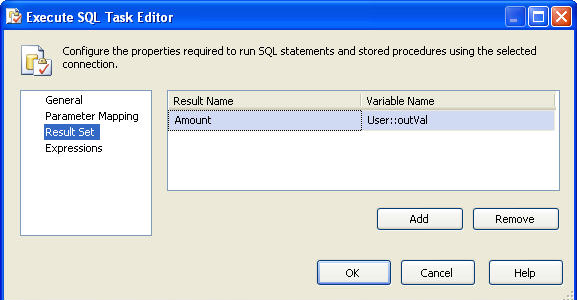
It will allow us to pass in a string variable as the SQL Statement. This variable value could have been set earlier on from inside the package or it could have been populated from outside using a configuration. The ResultSet property is set to single row and we'll show you why in a second when we look at the variables. Note also the SQLSourceType property. Here's the General Tab again.



Looking at the variable we have in this package you can see we have only two. One for the return value from the statement and one which is obviously for the statement itself.



Again we need to map the Result name to our variable and this can be a named Result Name (The column name or alias returned by the query) and not 0.

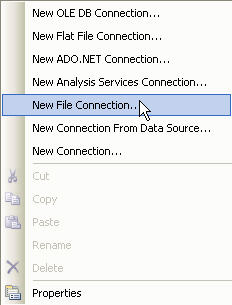


The expected result into our variable should be the amount of rows in the Person.Contact table and if we look in the watch window we see that it is.

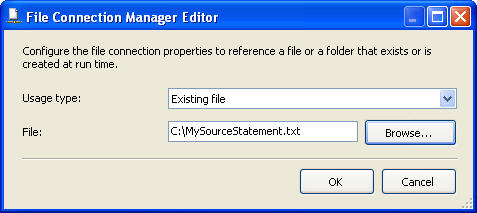


**Passing in the SQL Statement from a File**

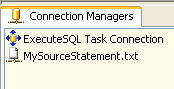
The final example we are going to show is a really interesting one. We are going to pass in the SQL statement to the task by using a file connection manager. The file itself contains the statement to run. The first thing we are going to need to do is create our file connection mananger to point to our file. Click in the connections tray at the bottom of the designer, right click and choose "New File Connection"



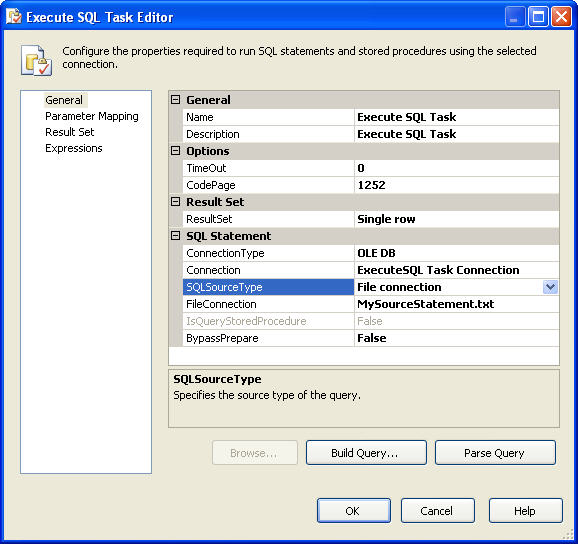
As you can see in the graphic below we have chosen to use an existing file and have passed in the name as well. Have a look around at the other "Usage Type" values available whilst you are here.



Having set that up we can now see in the connection manager tray our file connection manager sitting alongside our OLE-DB connection we have been using for the rest of these examples.



Now we can go back to the familiar *General Tab* to set up how the task will accept our file connection as the source.



All the other properties in this task are set up exactly as we have been doing for other examples depending on the options chosen so we will not cover them again here.