Tech Note 1007

Wonderware Application Server Scripting Implementations Part 1: Inserting a DB Row During Each Scan Cycle

Topic#: 002839
Created: January 2014

Introduction

Scripting is one of the most powerful features in the Application Server world. This Tech Note will demonstrate

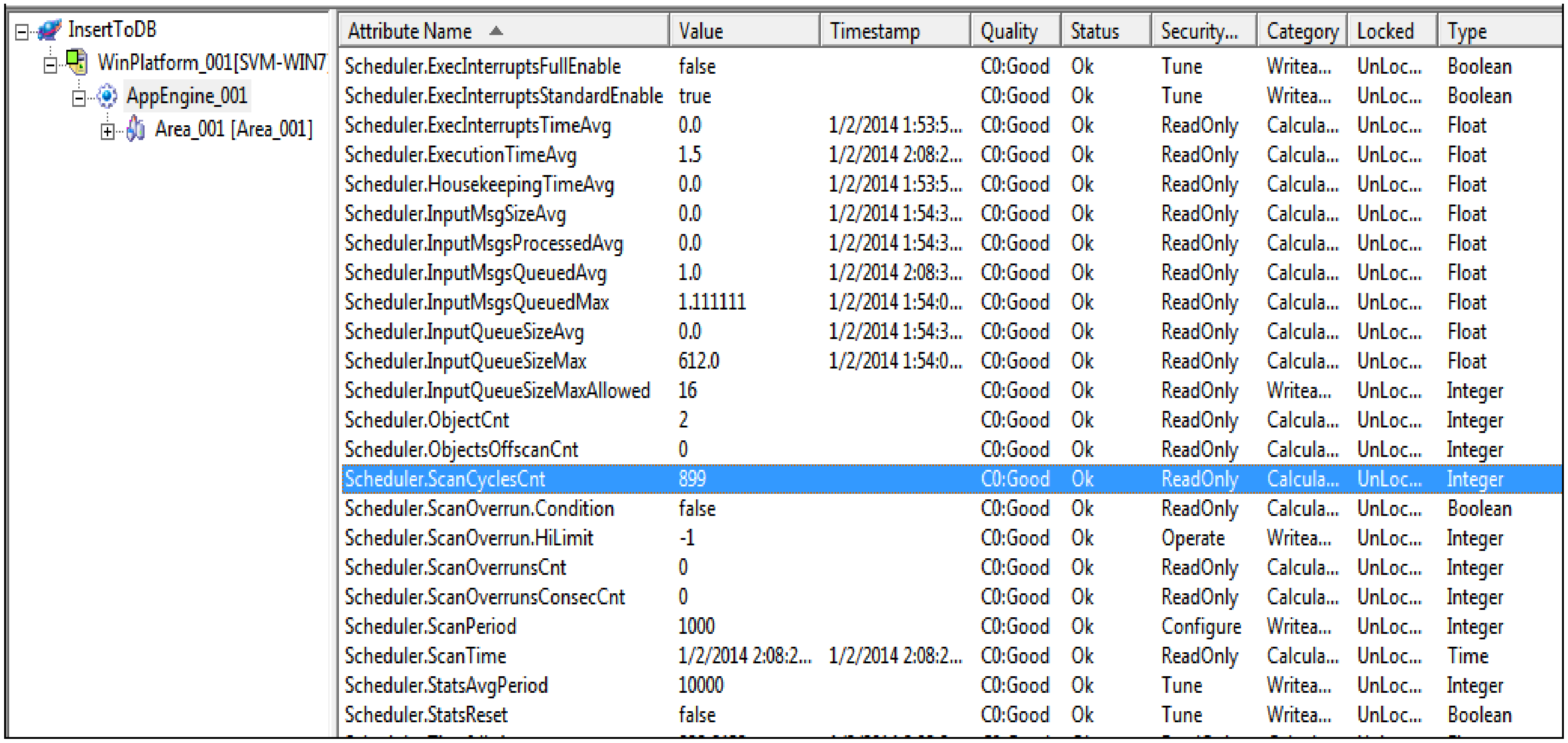
- The approach that can insert one data row into a database table during each AppEngine's scan cycle.
- The proper sequence for using scripts' Execution Types.
- Effective use of ArchestrA Objects' predefined attributes in the script's Expression for special requests.

Application Versions

- Wonderware Application Server 2012 and later

Inserting a DB Row in Each Scan Cycle

The most interesting part of this task is how to set up the proper script expression that triggers the corresponding implementation. Since the condition is **each Scan Cycle**, we know this condition is related to the AppEngine's Scheduler. From the Object Viewer, we found some predefined Scheduler's attributes:

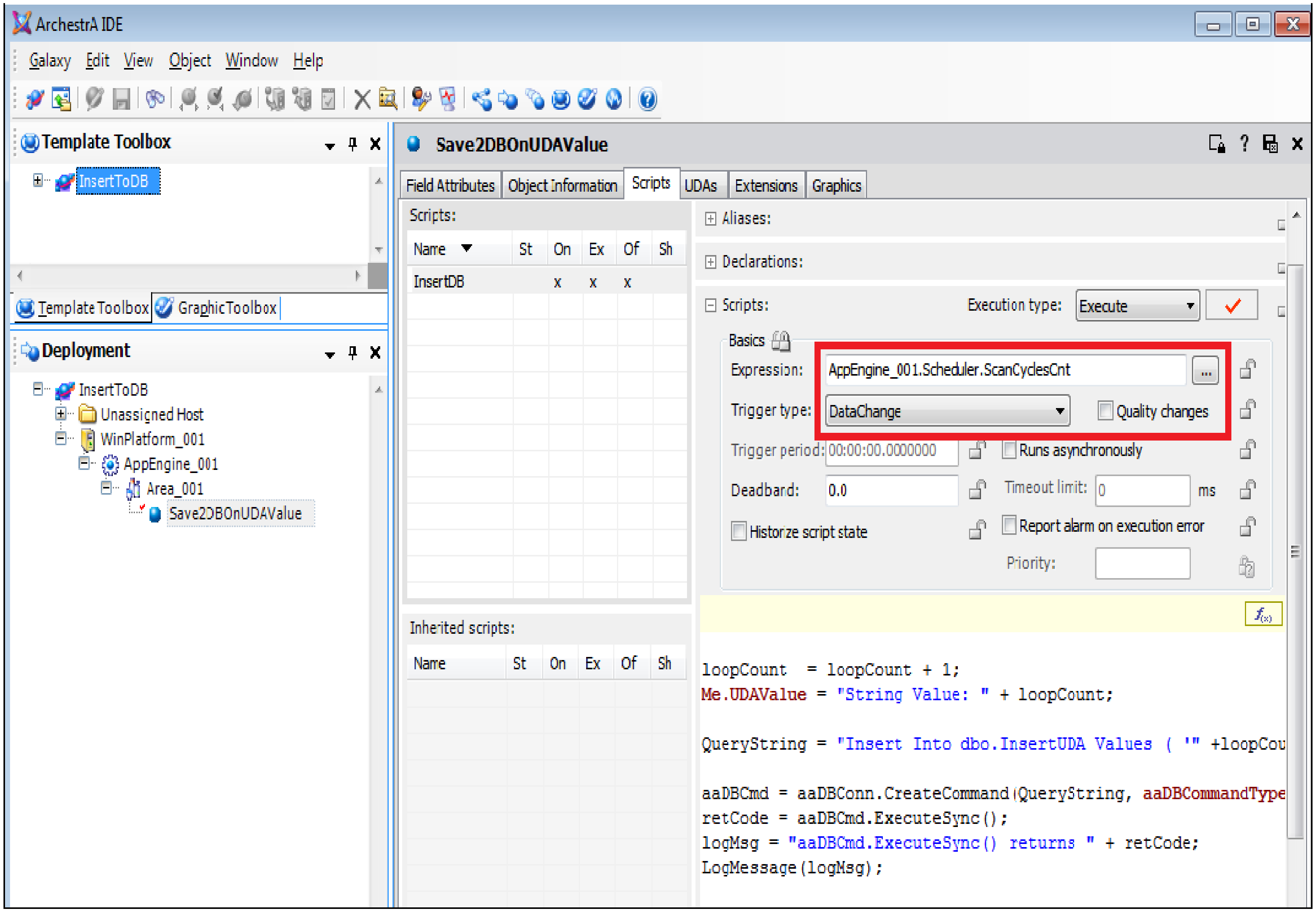


InsertToDB
WinPlatform_001[SVM-WIN7
AppEngine_001
Area_001 [Area_001]

Attribute Name	Value	Timestamp	Quality	Status	Security...	Category	Locked	Type
Scheduler.ExecInterruptsFullEnable	false		C0:Good	Ok	Tune	Writea...	UnLoc...	Boolean
Scheduler.ExecInterruptsStandardEnable	true		C0:Good	Ok	Tune	Writea...	UnLoc...	Boolean
Scheduler.ExecInterruptsTimeAvg	0.0	1/2/2014 1:53:5...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.ExecutionTimeAvg	1.5	1/2/2014 2:08:2...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.HousekeepingTimeAvg	0.0	1/2/2014 1:53:5...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputMsgSizeAvg	0.0	1/2/2014 1:54:3...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputMsgsProcessedAvg	0.0	1/2/2014 1:54:3...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputMsgsQueuedAvg	1.0	1/2/2014 2:08:3...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputMsgsQueuedMax	1.111111	1/2/2014 1:54:0...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputQueueSizeAvg	0.0	1/2/2014 1:54:3...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputQueueSizeMax	612.0	1/2/2014 1:54:0...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Float
Scheduler.InputQueueSizeMaxAllowed	16		C0:Good	Ok	ReadOnly	Writea...	UnLoc...	Integer
Scheduler.ObjectCnt	2		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Integer
Scheduler.ObjectsOffscanCnt	0		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Integer
Scheduler.ScanCyclesCnt	899		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Integer
Scheduler.ScanOverrun.Condition	false		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Boolean
Scheduler.ScanOverrun.HiLimit	-1		C0:Good	Ok	Operate	Writea...	UnLoc...	Integer
Scheduler.ScanOverrunsCnt	0		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Integer
Scheduler.ScanOverrunsConsecCnt	0		C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Integer
Scheduler.ScanPeriod	1000		C0:Good	Ok	Configure	Writea...	UnLoc...	Integer
Scheduler.ScanTime	1/2/2014 2:08:2...	1/2/2014 2:08:2...	C0:Good	Ok	ReadOnly	Calcula...	UnLoc...	Time
Scheduler.StatsAvgPeriod	10000		C0:Good	Ok	Tune	Writea...	UnLoc...	Integer
Scheduler.StatsReset	false		C0:Good	Ok	Tune	Writea...	UnLoc...	Boolean

Figure 1: AppEngine's Scheduler's Attributes

From examining each Scheduler's attribute, **Scheduler.ScanCyclesCnt** is the attribute we can use if we set the trigger at every CHANGING point. This analysis brings the following screenshot which shows the script Trigger Conditions.



Name	St	On	Ex	Of	Sh
InsertDB		x	x	x	

Scripts: Execution type: Execute

Expression: AppEngine_001.Scheduler.ScanCyclesCnt

Trigger type: DataChange

Quality changes: ☐

Trigger period: 00:00:00.0000000

Runs asynchronously: ☐

Deadband: 0.0

Timeout limit: 0 ms

Historize script state: ☐

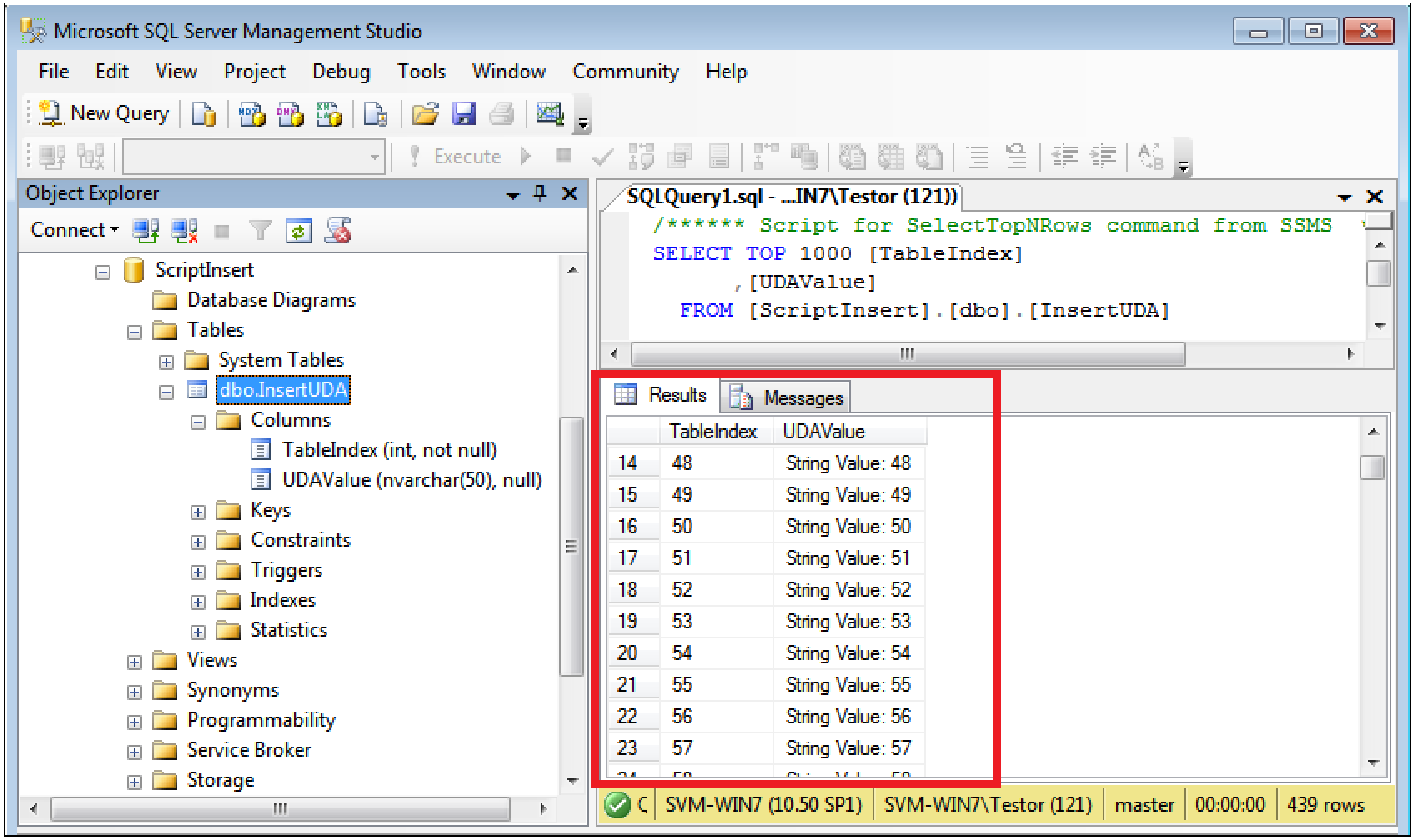
Report alarm on execution error: ☐

Priority:

```
loopCount   = loopCount + 1;
Me.UDAValue = "String Value: " + loopCount;

QueryString = "Insert Into dbo.InsertUDA Values ( '" +loopCou

aaDBCmd = aaDBConn.CreateCommand(QueryString, aaDBCommandType
retCode = aaDBCmd.ExecuteSync();
logMsg = "aaDBCmd.ExecuteSync() returns " + retCode;
LogMessage(logMsg);
```

Figure 2: Expression and Trigger Type Setup

As the result, we see the DB row is indeed inserted during each Scan Cycle.


Microsoft SQL Server Management Studio
File Edit View Project Debug Tools Window Community Help
New Query
Execute
Object Explorer
Connect
SQLQuery1.sql - ...IN7\Testor (121))
Results
Messages

```
ScriptInsert
    Database Diagrams
    Tables
        System Tables
        dbo.InsertUDA
            Columns
                TableIndex (int, not null)
                UDAValue (nvarchar(50), null)
            Keys
            Constraints
            Triggers
            Indexes
            Statistics
    Views
    Synonyms
    Programmability
    Service Broker
    Storage
```

```
/****** Script for SelectTopNRows command from SSMS
SELECT TOP 1000 [TableIndex]
      ,[UDAValue]
  FROM [ScriptInsert].[dbo].[InsertUDA]
```

	TableIndex	UDAValue
14	48	String Value: 48
15	49	String Value: 49
16	50	String Value: 50
17	51	String Value: 51
18	52	String Value: 52
19	53	String Value: 53
20	54	String Value: 54
21	55	String Value: 55
22	56	String Value: 56
23	57	String Value: 57

```
SVM-WIN7 (10.50 SP1) | SVM-WIN7\Testor (121) | master | 00:00:00 | 439 rows
```

FIGURE 3: DB ROWS ARE INSERTED

Implementation Details

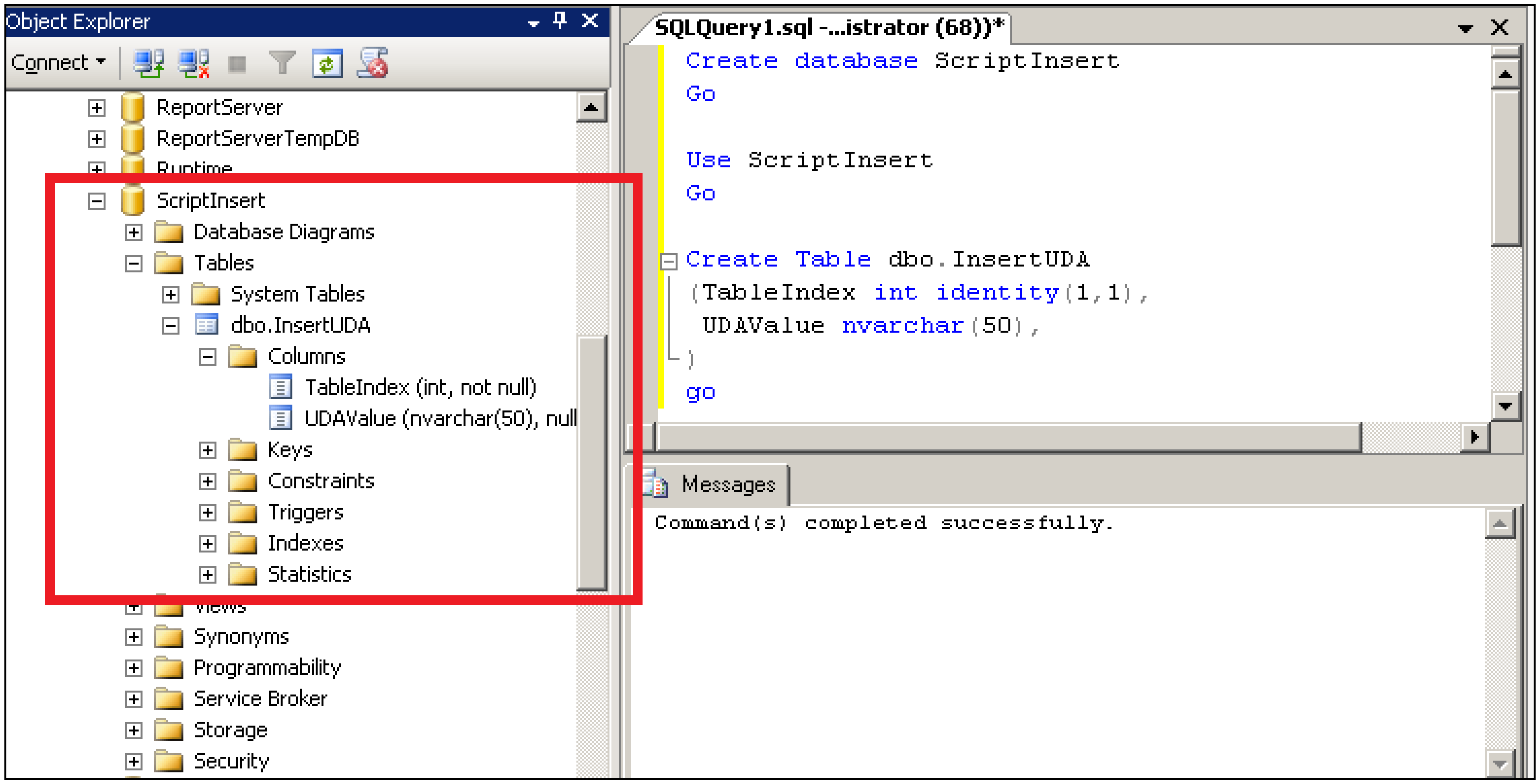
The following steps show the detail implementation of inserting a DB row during each Scan Cycle.

Create a Database and Table in SQL Server

Before inserting data, we must create a single table database. The SQL script below creates a database, ScriptInsert and a table, InsertUDA. The table has two columns, TableIndex and UDAValue.

```
Create database ScriptInsert
Go

Use ScriptInsert
Go
```

```
Create Table dbo.InsertUDA
(TableIndex int identity(1,1) ,
UDAValue nvarchar(50),
)
Go
```



```
Create database ScriptInsert
Go

Use ScriptInsert
Go

Create Table dbo.InsertUDA
(TableIndex int identity(1,1),
 UDAValue nvarchar(50),
)
go
```

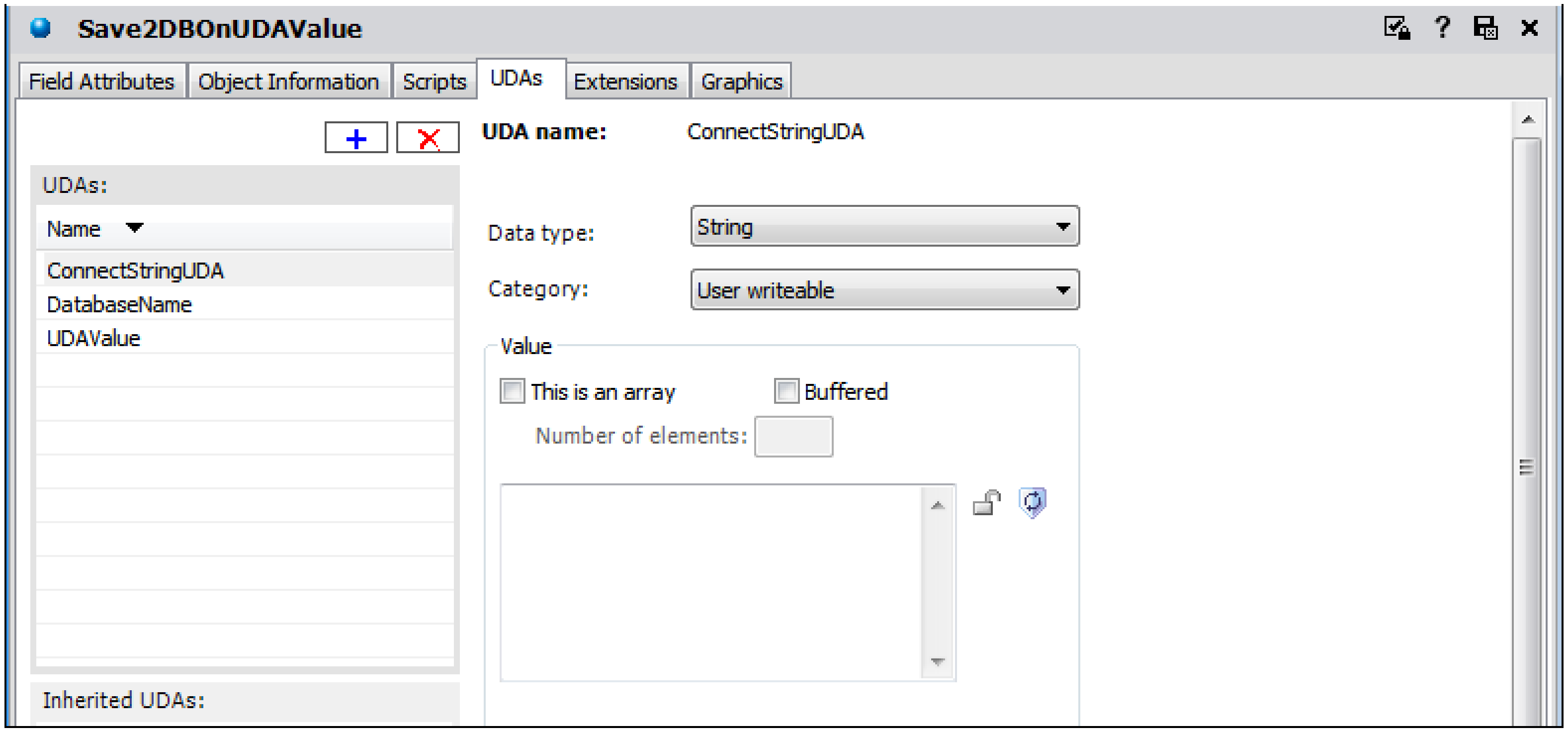
```
Command(s) completed successfully.
```

Figure 4: Create InsertUDA Table in ScriptInsert DB

Application Server Scripts

The following steps are showing the details that can insert data into SQL database during each AppEngine's scan cycle.

1. Define attributes in the UDA. For this example, we define three UDAs with string type:
   - ConnectStringUDA
   - DatabaseName with default value **ScriptInsert**
   - UDAValue



UDA name: ConnectStringUDA

UDAs: ConnectStringUDA, DatabaseName, UDAValue

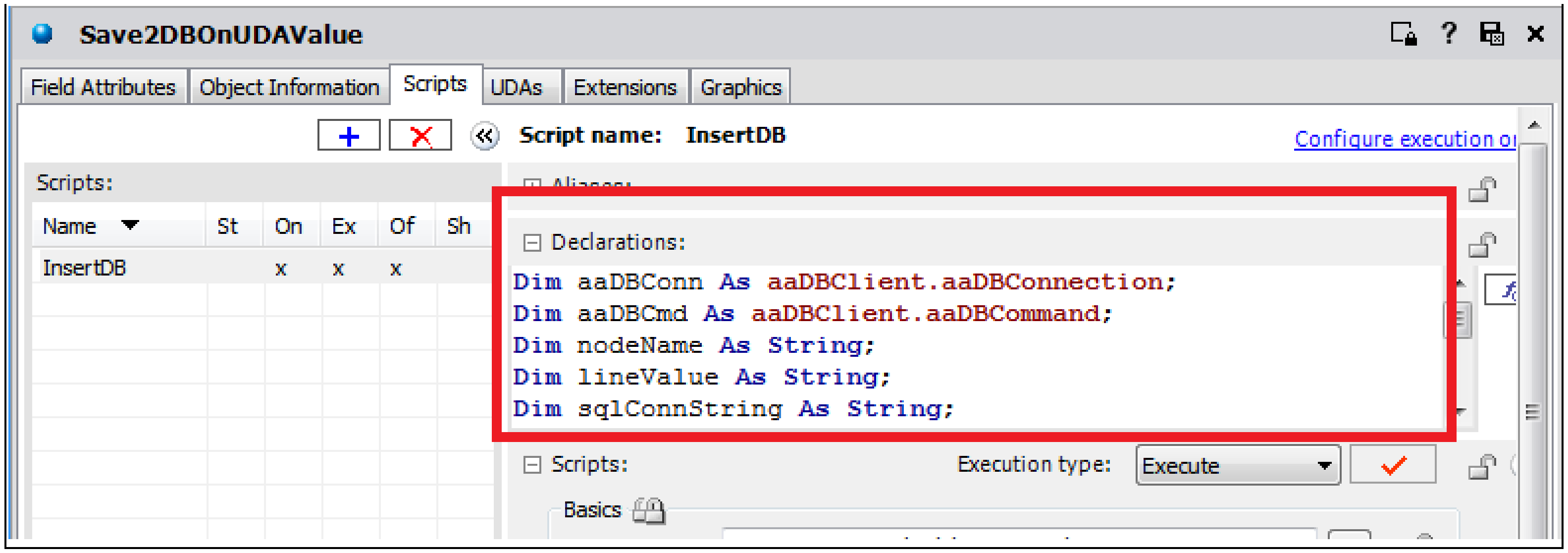
Data type: String

Category: User writeable

FIGURE 5: CREATE UDAS

2. Create a script called **InsertDB**, and define the following variables in the script's **Declarations** pane.

```
Dim aaDBConn As aaDBClient.aaDBConnection;
Dim aaDBCmd As aaDBClient.aaDBCommand;
Dim nodeName As String;
Dim lineValue As String;
Dim sqlConnString As String;
Dim QueryString As String;
Dim logMsg as String;
Dim retCode As Integer;
Dim loopCount As Integer;
Dim currDataSet As System.Data.DataSet;
Dim sqlDataTable As System.Data.DataTable;
```

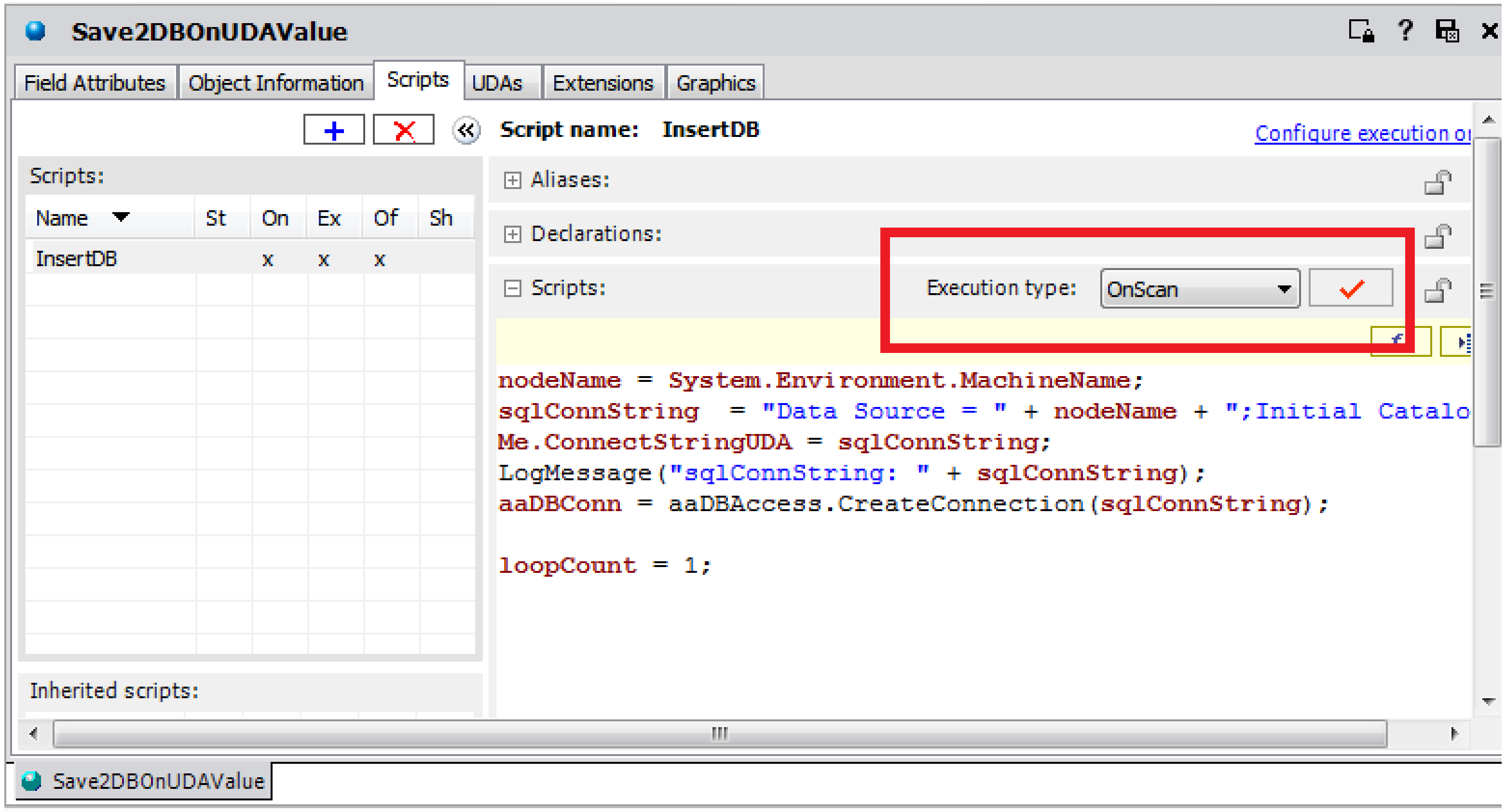


```
Dim aaDBConn As aaDBClient.aaDBConnection;
Dim aaDBCmd As aaDBClient.aaDBCommand;
Dim nodeName As String;
Dim lineValue As String;
Dim sqlConnString As String;
```

Figure 6: Declare Variables

3. In the Script's **OnScan** pane, use the following code snippet for creating a database connection.

```
nodeName = System.Environment.MachineName;
sqlConnString = "Data Source = " + nodeName + ";Initial Catalog=" + me.DatabaseName + ";Integrated Security=true";
Me.ConnectStringUDA = sqlConnString;
LogMessage("sqlConnString: " + sqlConnString);
aaDBConn = aaDBAccess.CreateConnection(sqlConnString);
loopCount = 1;
```



```
nodeName = System.Environment.MachineName;
sqlConnString  = "Data Source = " + nodeName + ";Initial Catalo
Me.ConnectStringUDA = sqlConnString;
LogMessage("sqlConnString: " + sqlConnString);
aaDBConn = aaDBAccess.CreateConnection(sqlConnString);

loopCount = 1;
```

Figure 7: Connect to Database

Important Note

As one of the best practices in AppServer's Scripting, we strongly suggest that you put the above statement, **aaDBAccess.CreateConnection**, into the **OnScan** or **Startup** Execution type no matter in which situation. In other words, you only need to create the database connection once and use it in the **Execute** Execution type.

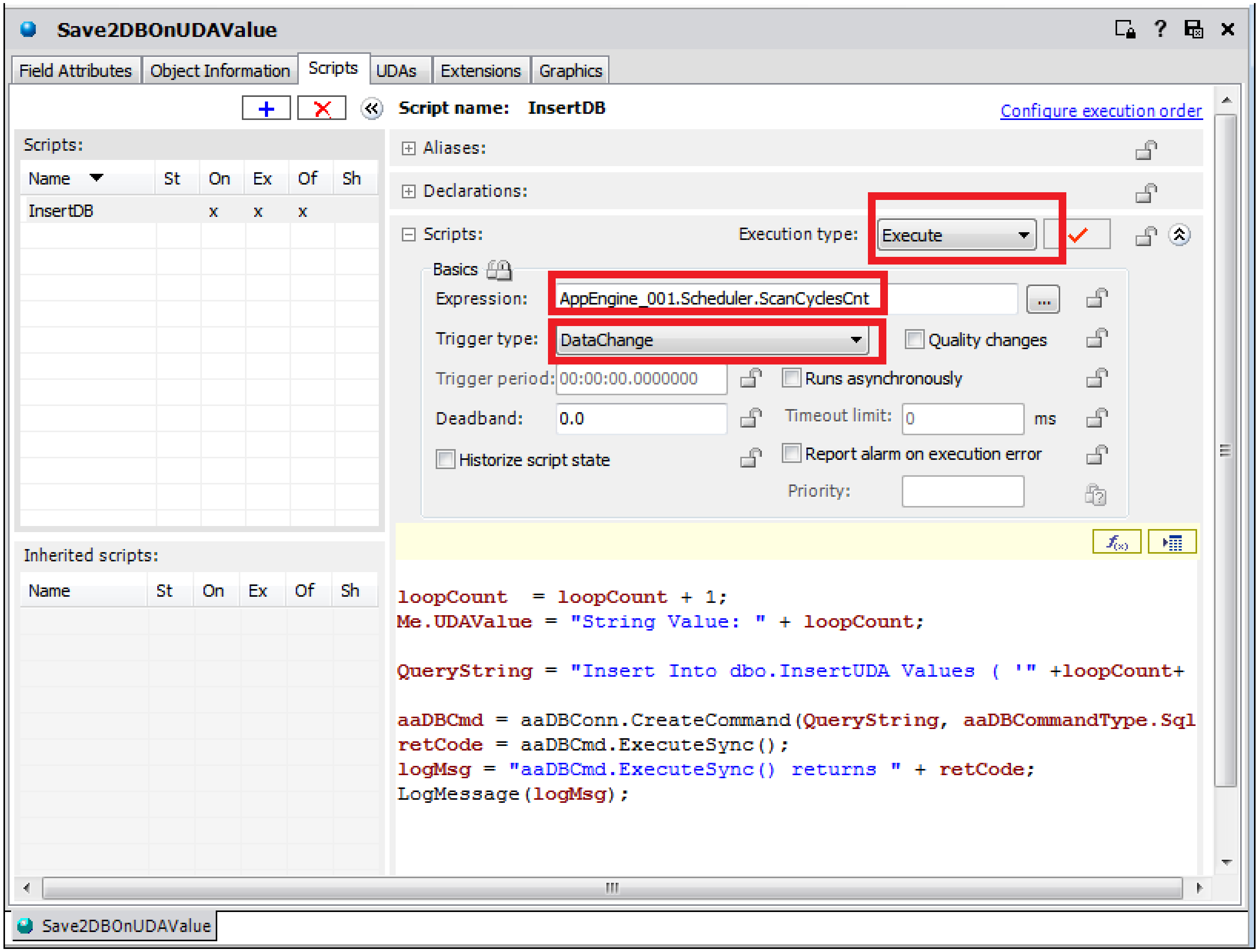
Please keep in mind that creating a database connection is always expensive. This is because

- Creating a DB connection uses a substantial amount of memory on both the database server and database client machines.
- Establishing a DB connection takes multiple network round trips to and from the database server.
- Opening numerous connections can contribute to out-of-memory conditions, which might cause paging of memory to disk and, thus, overall performance degradation.

4. In the Script's **Execute** pane, use the following code snippet to insert data into database in each AppEngine's Scan Cycle.

• Trigger Expression: **Scheduler.ScanCyclesCnt**
• Trigger Type: **DataChange**

```
loopCount = loopCount + 1;
Me.UDAValu = "String Value: " + loopCount;

QueryString = "Insert Into dbo.InsertUDA(UDAvalue) Values('" + Me.UDAValue + "')";

aaDBCmd = aaDBConn.CreateCommand(QueryString, aaDBCommandType.SqlStatement, true);
retCode = aaDBCmd.ExecuteSync();
logMsg = "aaDBCmd.ExecuteSync() returns " + retCode;
LogMessage(logMsg);
```



Script name: InsertDB

Name	St	On	Ex	Of	Sh
InsertDB		x	x	x	

Execution type: Execute

Expression: AppEngine_001.Scheduler.ScanCyclesCnt

Trigger type: DataChange

Quality changes

Trigger period: 00:00:00.0000000

Runs asynchronously

Deadband: 0.0

Timeout limit: 0 ms

Historize script state

Report alarm on execution error

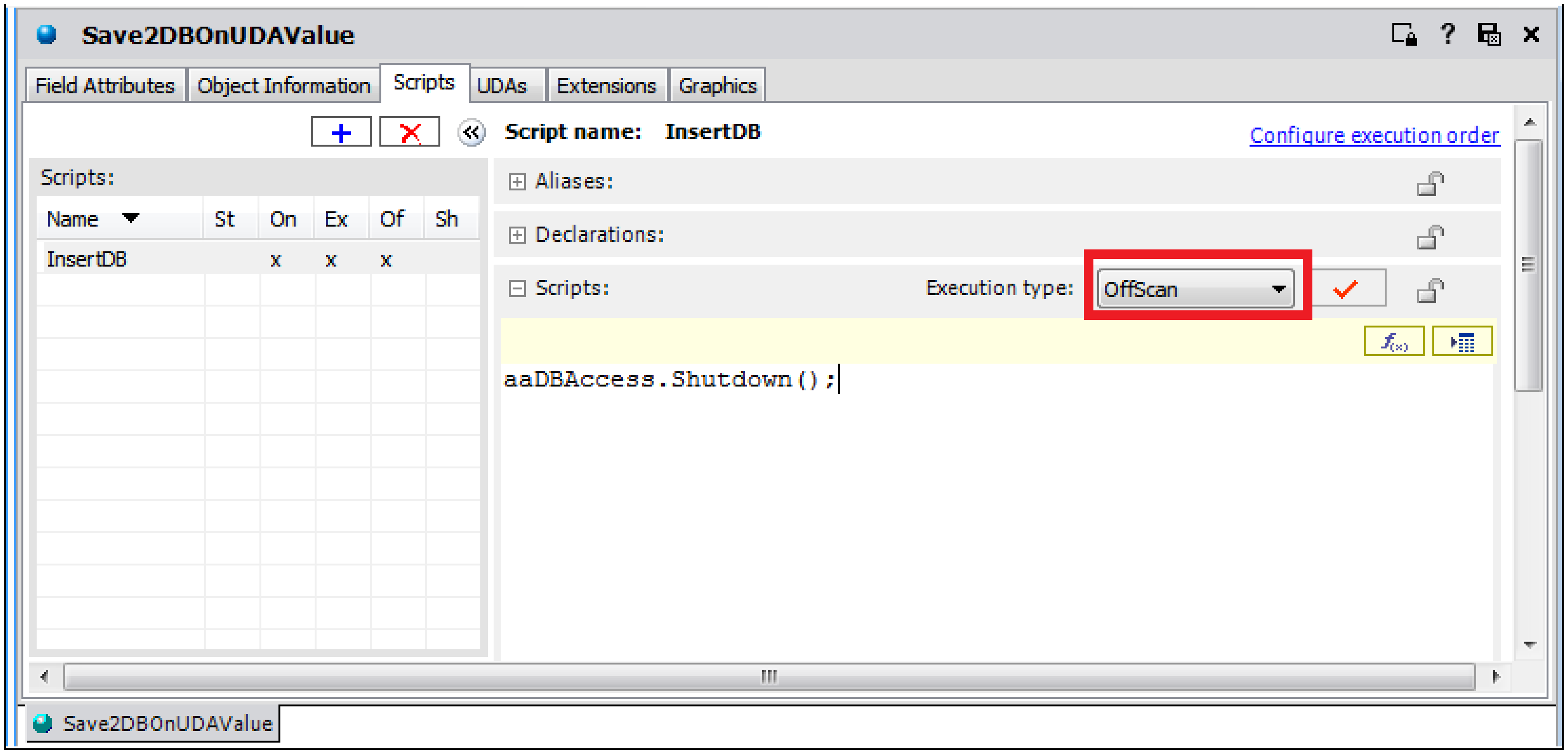
Priority:

```
loopCount  = loopCount + 1;
Me.UDAValue = "String Value: " + loopCount;

QueryString = "Insert Into dbo.InsertUDA Values ( '" +loopCount+

aaDBCmd = aaDBConn.CreateCommand(QueryString, aaDBCommandType.Sql
retCode = aaDBCmd.ExecuteSync();
logMsg = "aaDBCmd.ExecuteSync() returns " + retCode;
LogMessage(logMsg);
```

Figure 8: Insert a Row Into DB in Each AppEngine's Scan Cycle

5. In the Script's **OffScan** pane, use the following code snippet to shut down the database connection.

```
aaDBAccess.Shutdown();
```



```
aaDBAccess.Shutdown();
```

Figure 9: Shutdown the Inserting Execution

Scripts' Execution Types

There are five Script Execution types in Application Server: **Startup**, **OnScan**, **Execute**, **offScan** and **Shutdown**.

Each type has different executing condition.

- **Startup**: Scripts are called when the object containing the scripts is loaded into memory, such as during deployment, platform or engine start.
- **OnScan**: Scripts are called the first time an AppEngine calls this object to execute after the object's scan state changes to OnScan. The OnScan method initiates local object attribute values or provides more flexibility in the creation of .NET or COM objects.
- **Execute**: Scripts are called each time the AppEngine performs a scan and the object is OnScan.
- **OffScan**: Scripts are called when the object is taken OffScan. This script type is primarily used to clean up the object and account for any needs to address as a result of the object no longer executing.
- **Shutdown**: Scripts are primarily used to destroy COM objects and .NET objects, and to free memory.

References

- Wonderware Application Server 2012 R2 – IDE.PDF

B. Lin and E. Xu

Tech Notes are published occasionally by Wonderware Technical Support. Publisher: Invensys Systems, Inc., 26561 Rancho Parkway South, Lake Forest, CA 92630. There is also technical information on our software products at **Wonderware Technical Support.**

For technical support questions, send an e-mail to **wwsupport@invensys.com**.

Back to top