Tech Note 808
Extending User Defined Attribute Arrays to DASABCIP Arrays

Topic#: 002595
Created: November 2011

Introduction

This *Tech Note* outlines how to populate a User Defined Object's (UDO's) User Defined Attribute (UDA) Array with array values from a ControlLogix/CompactLogix processor using DASABCIP.

Application Versions

- Application Server 3.1 SP3 p01
- DASABCIP 4.1 SP2

Note: This *Tech Note* only addresses reading of array values, not poking of new values.

Overview

A UDA array is not directly extendable to IO sources; however, customers would often like to be able to directly map arrays from their Logix processors to a UDA array. DASABCIP has block read capabilities that can be utilized to accomplish this requirement. When this block read is executed via an OPCClient DIObject, the array values are returned with a comma delimiter and then can be parsed with a simple script to move the values into corresponding UDA Array elements.

Important Limitations and Information

- The OPCClient DIObject is required to accomplish this functionality since the values are returned with a comma delimiter. If you try to use the DDESuitelinkClient DIObject, the values are returned in Hex, making the process more difficult since a hex to decimal conversion will be required.
- Logix arrays are base-0, but UDA Array are base-1, so you will need to remember that there is an offset that will occur.
- DASABCIP Block Read syntax is required:
  - **<Tag_Name>[<first_element_X>],L<number_of_items_#>**
- DASABCIP only supports Block Reads and Writes of one-dimensional arrays from the supported ControlLogix, FlexLogix, and CompactLogix controllers.
  - The following features are ***NOT*** supported by the DAServer:
    - Block Reads/Writes of strings
    - Block Reads of structures (either predefined or user-defined)
    - Block Reads of greater than 486 bytes
      - There are five different data types that are supported, each of which requires a different allowance on the qualifier due to the block size limitation (Table 1 below).

- There are three optimization modes supported, each with a different maximum qualifier allowance as shown in the following table: Optimize for Reads, Optimize for Startup, and No Optimization.
- Note: The number in the "**Ln**" qualifier should not need an offset, because it is the total number counting from 1 (one).

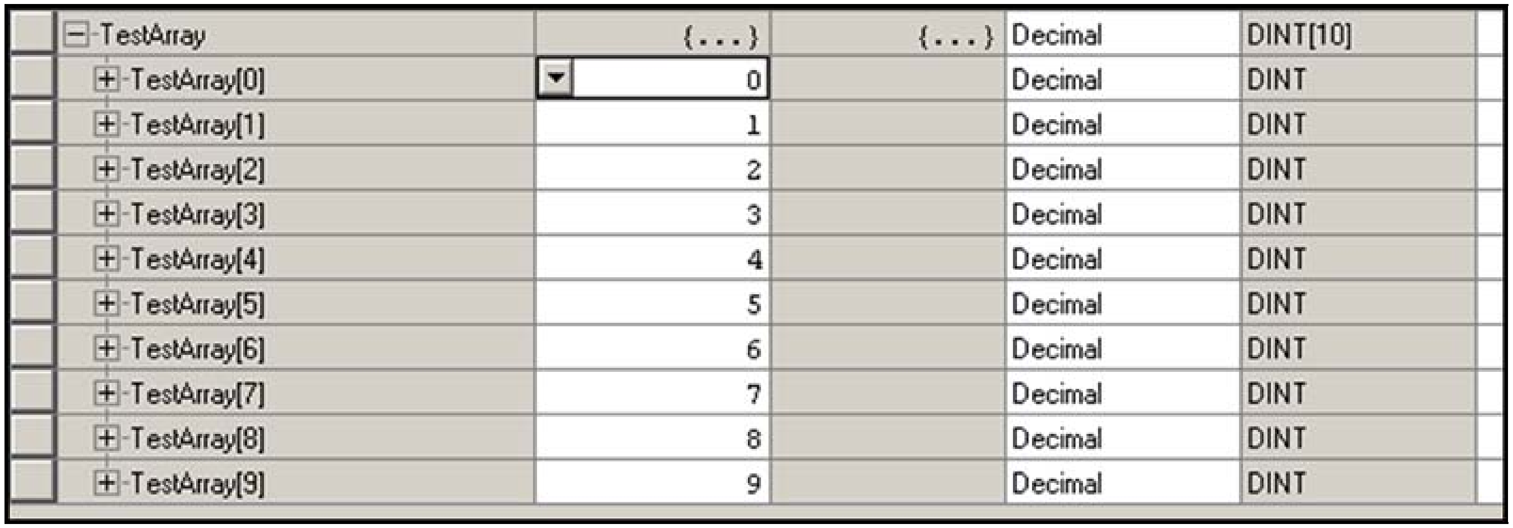
Data Type	**Qualifier Allowance (number of items)**	
	Optimize for Read	**Optimize for Startup**
		No Optimization
Boolean	3840	3831
SINT	486	478
INT	243	239
DINT	121	119
REAL	121	119
LINT	60	59

Table 1: Block Read Qualifier Limits

Sample Configuration

Controller Tag Configuration

For this example, we've created a 10 element DINT array called **TestArray** in a SoftLogix processor (Figure 1 below).

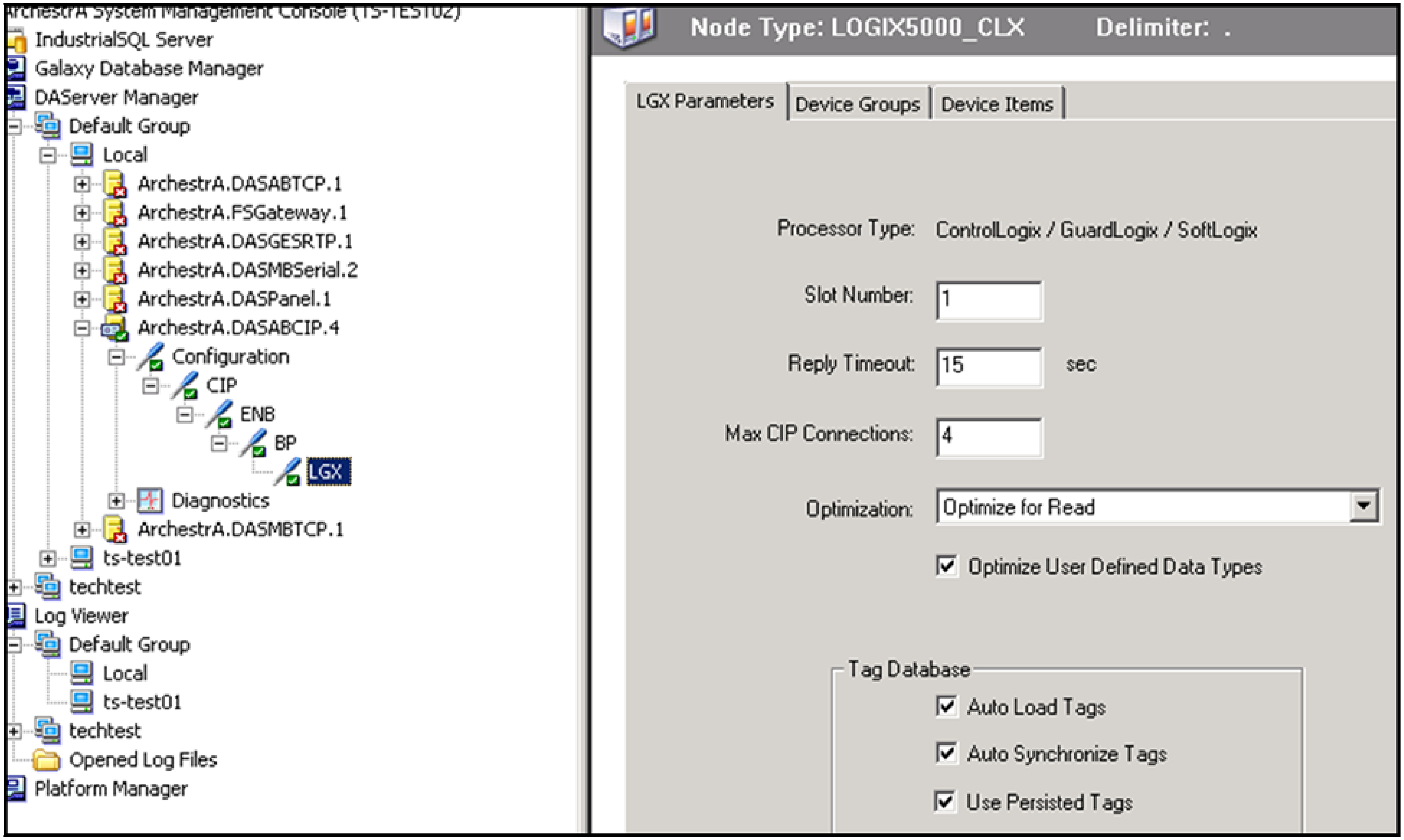


⊟-TestArray	{...}	{...}	Decimal	DINT[10]
⊞-TestArray[0]	0		Decimal	DINT
⊞-TestArray[1]	1		Decimal	DINT
⊞-TestArray[2]	2		Decimal	DINT
⊞-TestArray[3]	3		Decimal	DINT
⊞-TestArray[4]	4		Decimal	DINT
⊞-TestArray[5]	5		Decimal	DINT
⊞-TestArray[6]	6		Decimal	DINT
⊞-TestArray[7]	7		Decimal	DINT
⊞-TestArray[8]	8		Decimal	DINT
⊞-TestArray[9]	9		Decimal	DINT

Figure 1: Controller Array Example

DASABCIP Configuration

No special configuration is required here. Just set up a typical connection to a Logix processor as described in the DASABCIP help (Figure 2 below).


ArchestrA System Management Console (TS-TEST02)
- IndustrialSQL Server
- Galaxy Database Manager
- DAServer Manager
  - Default Group
    - Local
      - ArchestrA.DASABTCP.1
      - ArchestrA.FSGateway.1
      - ArchestrA.DASGESRTP.1
      - ArchestrA.DASMBSerial.2
      - ArchestrA.DASPanel.1
      - ArchestrA.DASABCIP.4
        - Configuration
          - CIP
            - ENB
              - BP
                - LGX
        - Diagnostics
      - ArchestrA.DASMBTCP.1
    - ts-test01
  - techtest
- Log Viewer
  - Default Group
    - Local
    - ts-test01
  - techtest
  - Opened Log Files
- Platform Manager

Node Type: LOGIX5000_CLX Delimiter: .

LGX Parameters | Device Groups | Device Items

Processor Type: ControlLogix / GuardLogix / SoftLogix

Slot Number: 1

Reply Timeout: 15 sec

Max CIP Connections: 4

Optimization: Optimize for Read

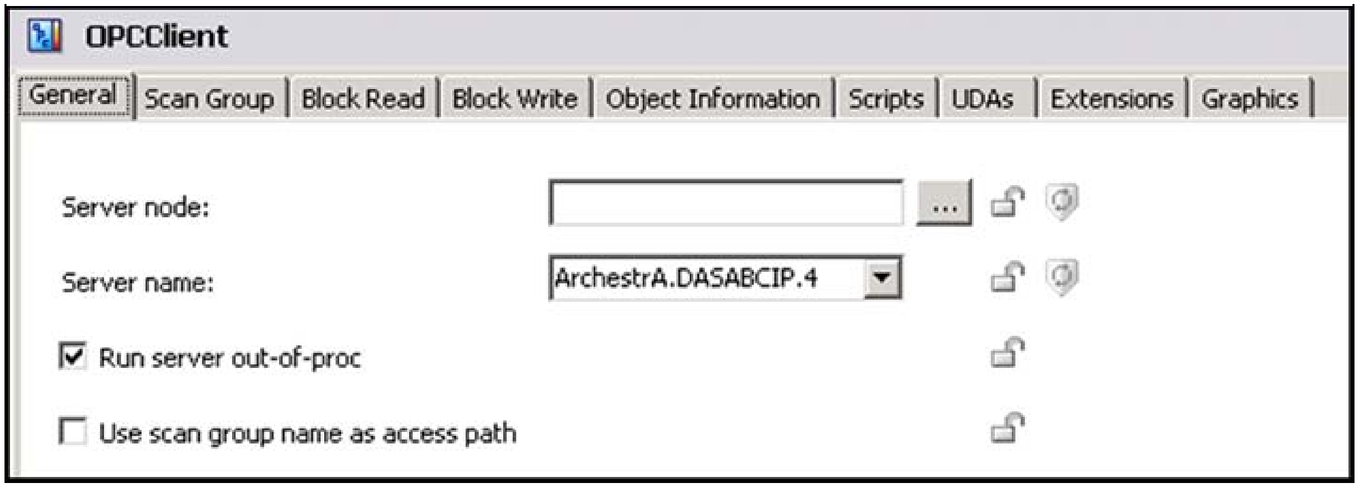
☑ Optimize User Defined Data Types

Tag Database
- ☑ Auto Load Tags
- ☑ Auto Synchronize Tags
- ☑ Use Persisted Tags

Figure 2: Sample DASABCIP Configuration

OPCClient DIObject Configuration

Configure the OPCClient DIObject to communicate via OPC to DASABCIP (Figure 3 below).



Server node:

Server name: ArchestrA.DASABCIP.4

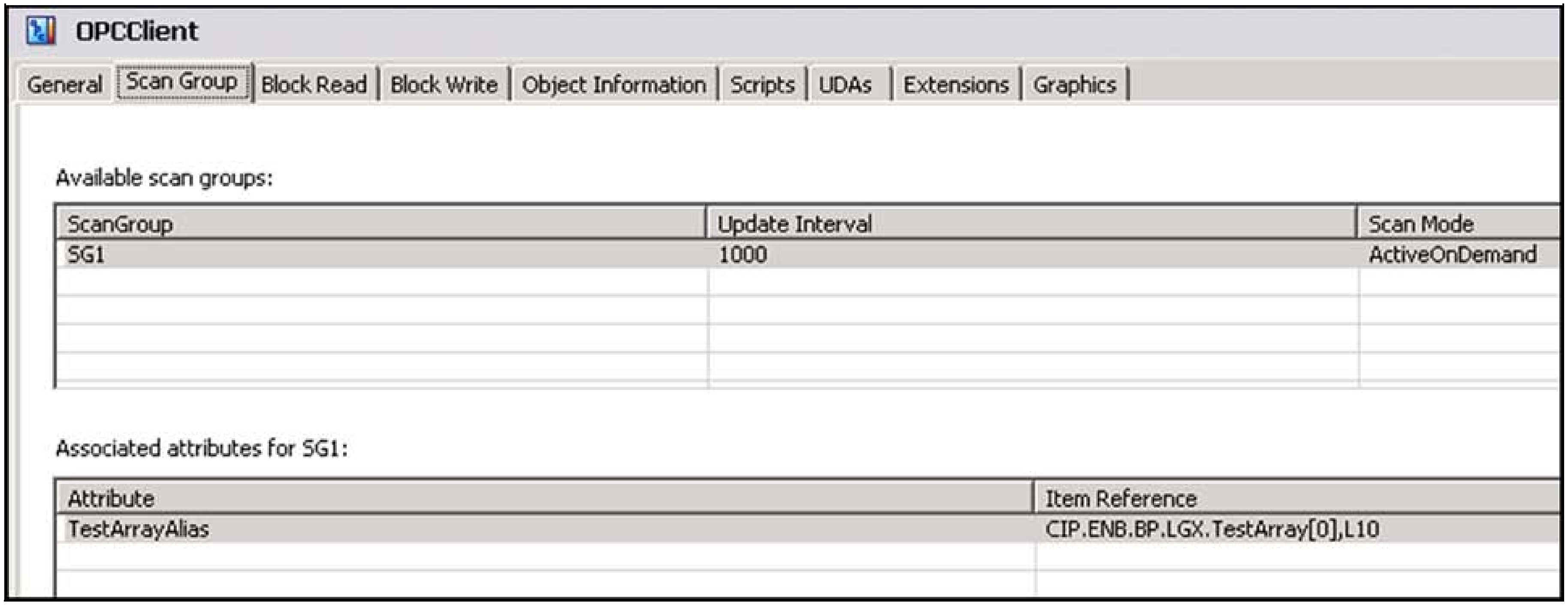
☑ Run server out-of-proc

☐ Use scan group name as access path

FIGURE 3: OPCCLIENT GENERAL CONFIGURATION

A scan group must be created with an attribute that utilizes the proper syntax to block read the Logix array (Figure 4 below).

- If you use an alias in the Attribute Column, it will make the scripting simpler
- The item syntax is a concatenation of the object names in the DASABCIP configuration hierarchy, separated by periods, then <Tag_Name>[<first_element_X>],L<number_of_items_#>
  • If you are having trouble with the item reference syntax, you can press the blue + symbol on the attribute list, then the ellipsis [...] in the item reference field to browse the DASABCIP hierarchy.
  • In the figure below you can see that we're reading an array named "TestArray", beginning at element 0, and reading 10 elements total. (TestArray[0],L10), using an alias name of "TestArrayAlias"



Available scan groups:

ScanGroup	Update Interval	Scan Mode
SG1	1000	ActiveOnDemand

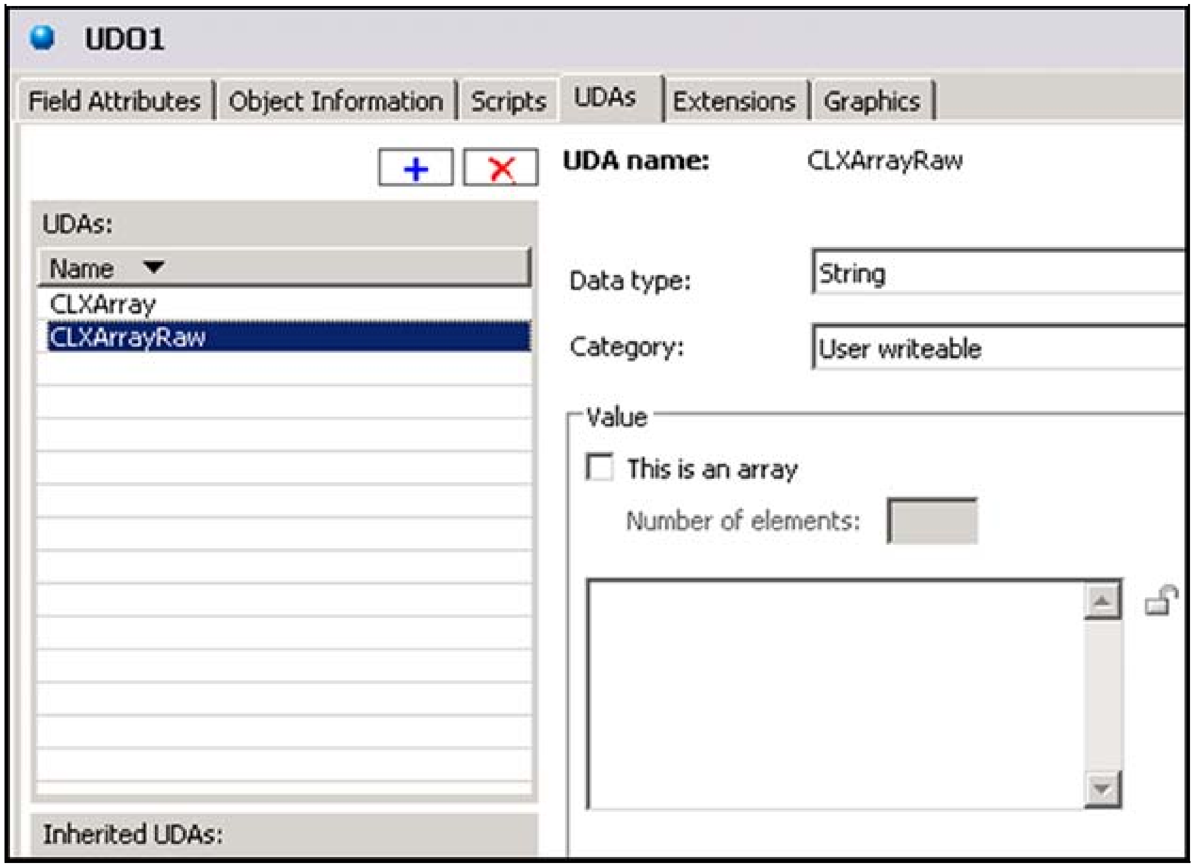
Associated attributes for SG1:

Attribute	Item Reference
TestArrayAlias	CIP.ENB.BP.LGX.TestArray[0],L10

Figure 4: OPCClient Scan Group Configuration

User Defined Object Configuration

You need to define two UDAs, one to hold the raw IO data from the Logix array (string), and another that is the UDA array to be populated, which is an integer array in this example (Figures 5 & 6 below).



CLXArray
CLXArrayRaw

UDA name: CLXArrayRaw

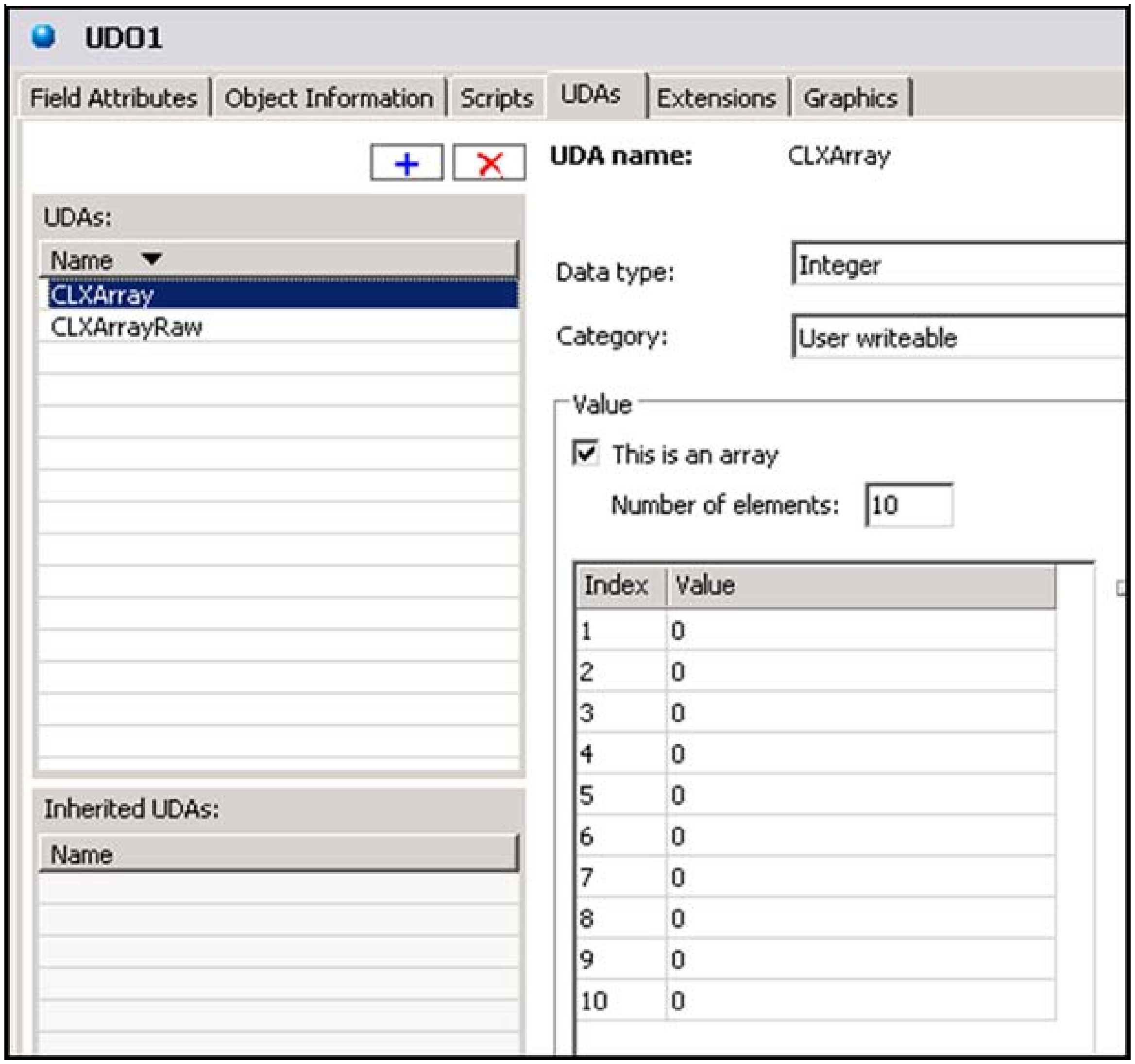
Data type: String

Category: User writeable

☐ This is an array

Number of elements:

FIGURE 5: UDA STRING TO HOLD RAW ARRAY IO DATA



UDA name: CLXArray

Data type: Integer

Category: User writeable

Value

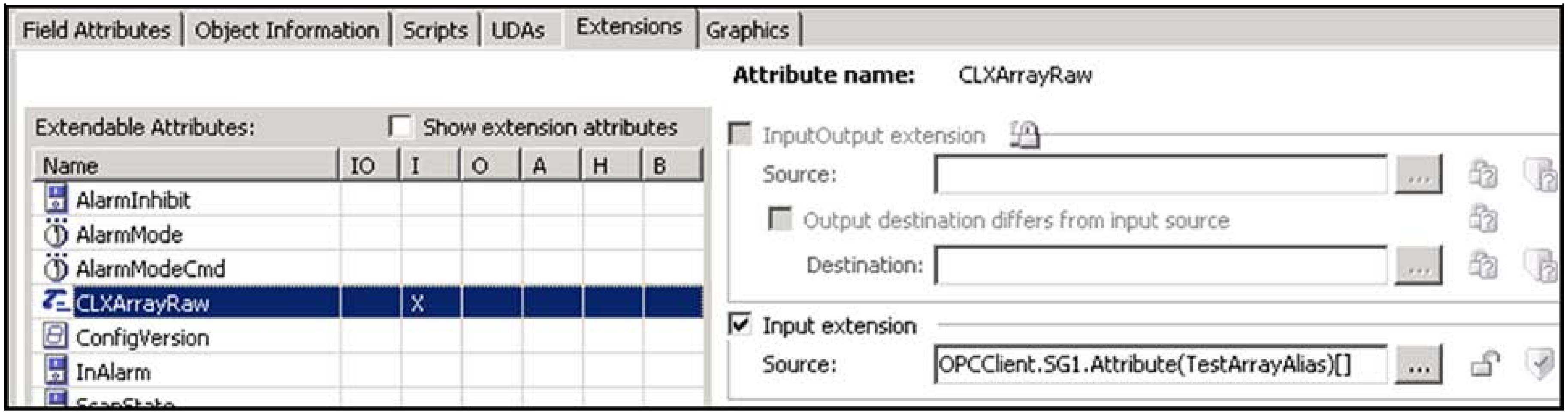
☑ This is an array

Number of elements: 10

Index	Value
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0

FIGURE 6: UDA ARRAY

The string UDA that holds the raw IO data needs to be extended to the input with a special syntax as below in order to capture the elements in the proper comma delimited format (Figure 7 below).



Attribute name: CLXArrayRaw

Extendable Attributes: ☐ Show extension attributes

Name	IO	I	O	A	H	B
AlarmInhibit						
AlarmMode						
AlarmModeCmd						
CLXArrayRaw		X				
ConfigVersion						
InAlarm						

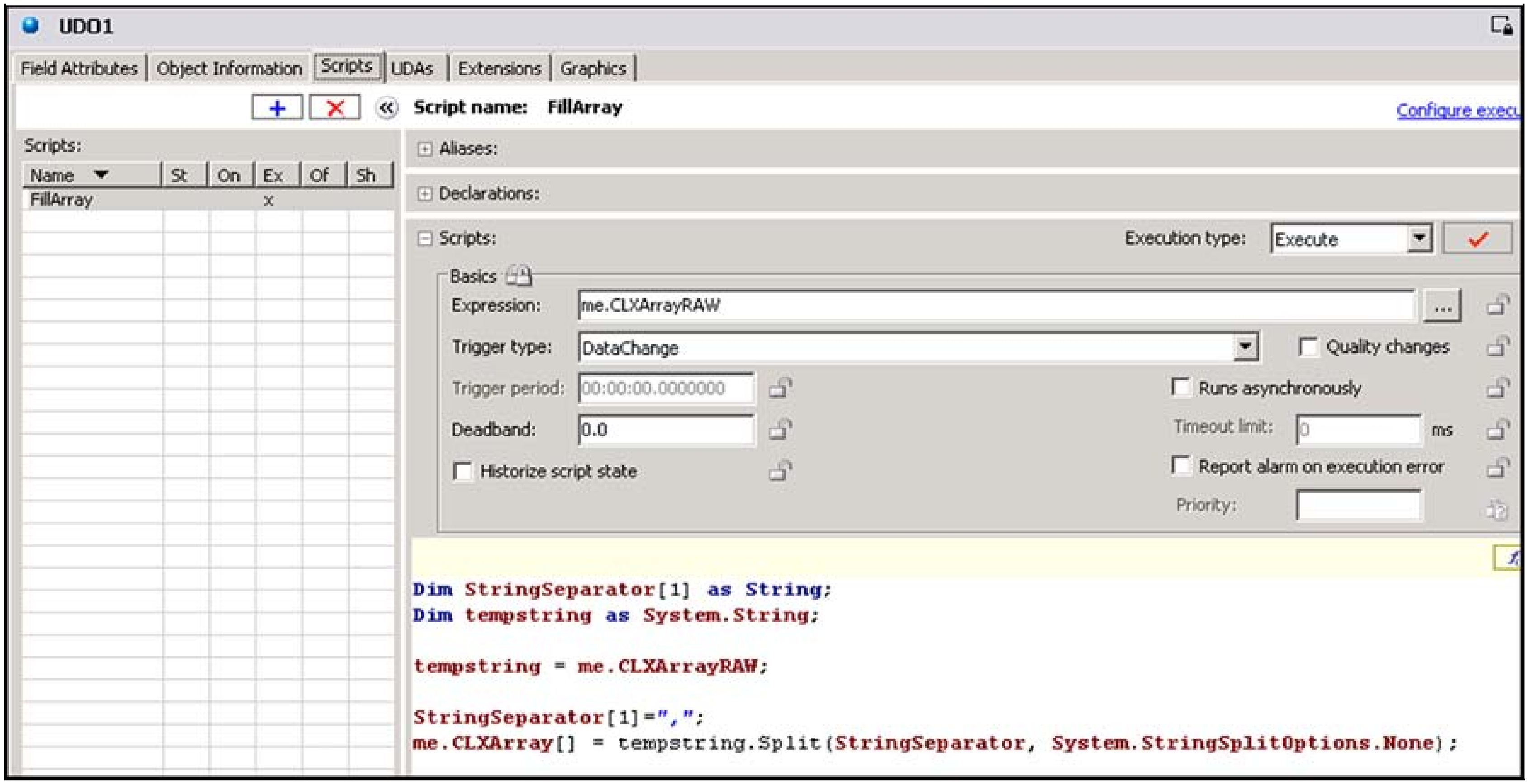
☐ InputOutput extension
Source:
☐ Output destination differs from input source
Destination:

☑ Input extension
Source: OPCClient.SG1.Attribute(TestArrayAlias)[]

Figure 7: UDA String Extension

Sample Script

An OnDataChange Script can monitor the raw data for changes, then parse the raw data and populate the UDA array using the Microsoft .NET System String Split method as needed (Figure 8 below).



Script name: FillArray

Name	St	On	Ex	Of	Sh
FillArray			x		

Execution type: Execute

Expression: me.CLXArrayRAW

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:

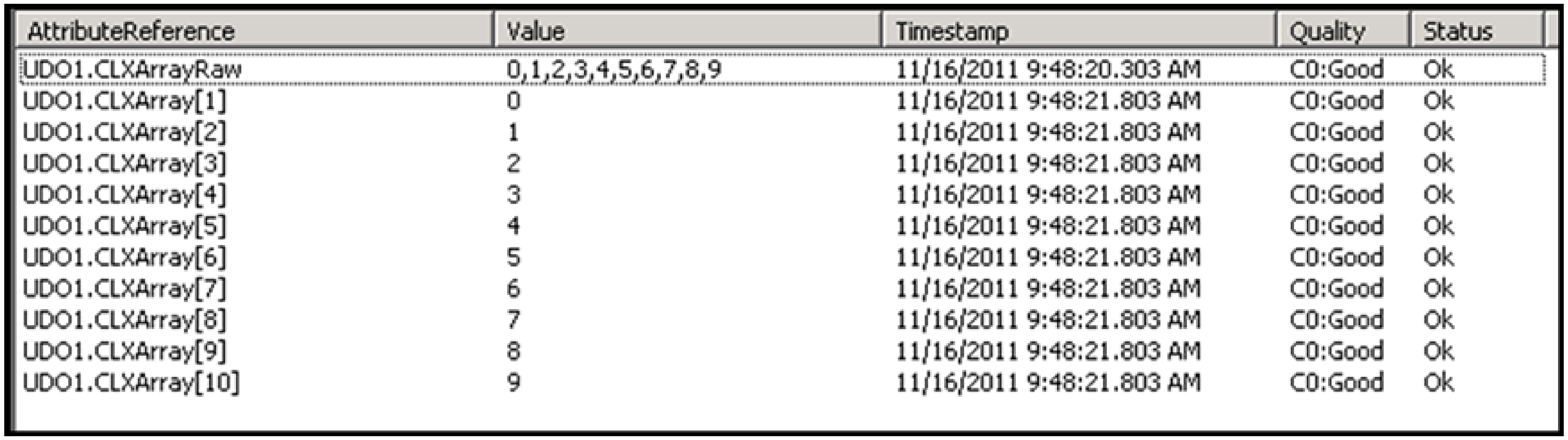
```
Dim StringSeparator[1] as String;
Dim tempstring as System.String;

tempstring = me.CLXArrayRAW;

StringSeparator[1]=",";
me.CLXArray[] = tempstring.Split(StringSeparator, System.StringSplitOptions.None);
```

Figure 8: Using .NET System String Split to Parse the Raw Data and Populate the UDA Array

Here are the example attributes as seen from Object Viewer (Figure 9 below).



AttributeReference	Value	Timestamp	Quality	Status
UDO1.CLXArrayRaw	0,1,2,3,4,5,6,7,8,9	11/16/2011 9:48:20.303 AM	C0:Good	Ok
UDO1.CLXArray[1]	0	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[2]	1	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[3]	2	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[4]	3	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[5]	4	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[6]	5	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[7]	6	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[8]	7	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[9]	8	11/16/2011 9:48:21.803 AM	C0:Good	Ok
UDO1.CLXArray[10]	9	11/16/2011 9:48:21.803 AM	C0:Good	Ok

Figure 9: Object Viewer with Example Attributes

Back to top