Symptoms

If you try to initialize an array of a user-defined data type in the DEFINE_CONSTANT or DEFINE_VARIABLE sections of your NetLinx code, you may get the compile error "Too many elements in initializer" or other errors. Even if you do not get compile errors when you run the code you find the structure array contains no data.

Cause

Initializing an array of a user-defined data type in DEFINE_CONSTANT or DEFINE_VARIABLE is not supported at this time.
Only arrays of the pre-defined data types such as DEV, DEVCHAN, etc. can be initialized in these sections.

Resolution

Instead initialize the structure array in DEFINE_START, for example:

```
DEFINE_TYPE
STRUCT innerStruct
{
INTEGER most
INTEGER secret
}
STRUCT someStruct
{
CHAR str[2][32]
INTEGER num
DEV dv
innerStruct inner
}
DEFINE_VARIABLE
someStruct struct1 = {{'Eleanor','Rigby'}, 1, 33001:1:0,{11,12}}
someStruct struct2 = {{'Colonel','Mustard'},2,33002:1:0,{21,22}}
someStruct struct3 = {{'MC','Hammer'},3,33003:1:0,{31,32}}
someStruct structArray[3]
DEFINE_START
structArray[1] = struct1
structArray[2] = struct2
structArray[3] = struct3
```

In this first example you end up declaring two structures instead of one for each array element to be initialized, but the syntax is more concise than if you set each structure member individually in DEFINE_START (as below) so the memory usage is not increased as much as you might think.
However, if your structure contains arrays of another structure, even of the pre-defined data types, you must type it all out longhand in DEFINE_START:

```
DEFINE_TYPE
STRUCT innerStruct
{
INTEGER most
INTEGER secret
}
STRUCT someStruct
{
CHAR str[2][32]
INTEGER num
DEV dv[2]
innerStruct inner
}
DEFINE_VARIABLE
someStruct structArray[3]
DEFINE_START
structArray[1].str[1] = 'Eleanor'
structArray[1].str[2] = 'Rigby'
structArray[1].num = 1
structArray[1].dv[1] = 33001:1:0
structArray[1].dv [2] = 33001:2:0
structArray[1].inner.most = 11
structArray[1].inner.secret = 12
// etc…
```