X-Arrays

When an ordinary array field is defined, you have to specify the index bounds exactly, hence the number of occurrences for each dimension. At runtime, the complete array field is existent by default; each of its defined occurrences can be accessed without performing additional allocation operations. The size layout cannot be changed anymore; you may neither add nor remove field occurrences.

However, if the number of occurrences needed is unknown at development time, but you want to flexibly increase or decrease the number of the array fields at runtime, you should use what is called an X-array (eXtensible array).

An X-array can be resized at runtime and can help you manage memory more efficiently. For example, you can use a large number of array occurrences for a short time and then reduce memory when the application is no longer using the array.

This chapter covers the following topics:

- Definition
- Storage Management of X-Arrays
- Storage Management of X-Group Arrays
- Referencing an X-Array
- Parameter Transfer with X-Arrays
- Parameter Transfer with X-Group Arrays
- X-Array of Dynamic Variables
- Lower and Upper Bound of an Array

Definition

An X-array is an array of which the number of occurrences is undefined at compile time. It is defined in a `DEFINE DATA` statement by specifying an asterisk (*) for at least one index bound of at least one array dimension. An asterisk (*) character in the index definition represents a variable index bound which can be assigned to a definite value during program execution. Only one bound - either upper or lower - may be defined as variable, but not both.

An X-array can be defined whenever a (fixed) array can be defined, i.e. at any level or even as an indexed group. It cannot be used to access MU-/PE-fields of a database view. A multidimensional array may have a mixture of constant and variable bounds.

Example:

```
DEFINE DATA LOCAL
1 #X-ARR1 (A5/1:*)          /* lower bound is fixed at 1, upper bound is variable
1 #X-ARR2 (A5/*)            /* shortcut for (A5/1:*)
1 #X-ARR3 (A5/*:100)        /* lower bound is variable, upper bound is fixed at 100
1 #X-ARR4 (A5/1:10,1:*)     /* 1st dimension has a fixed index range with (1:10)
END-DEFINE                  /* 2nd dimension has fixed lower bound 1 and variable upper bound
```

Storage Management of X-Arrays

Occurrences of an X-array must be allocated explicitly before they can be accessed. To increase or decrease the number of occurrences of a dimension, the statements EXPAND, RESIZE and REDUCE may be used.

However, the number of dimensions of the X-array (1, 2 or 3 dimensions) cannot be changed.

Example:

```
DEFINE DATA LOCAL
1 #X-ARR(I4/10:*)
END-DEFINE
EXPAND ARRAY #X-ARR TO (10:10000)
/* #X-ARR(10) to #X-ARR(10000) are accessible
WRITE *LBOUND(#X-ARR)                  /* is 10
   *UBOUND(#X-ARR)                     /* is 10000
   *OCCURRENCE(#X-ARR)                 /* is 9991
#X-ARR(*) := 4711                      /* same as #X-ARR(10:10000) := 4711
/* resize array from current lower bound=10 to upper bound =1000
RESIZE ARRAY #X-ARR TO (*:1000)
/* #X-ARR(10) to #X-ARR(1000) are accessible
/* #X-ARR(1001) to #X-ARR(10000) are released
WRITE *LBOUND(#X-ARR)                  /* is 10
   *UBOUND(#X-ARR)                     /* is 1000
   *OCCURRENCE(#X-ARR)                 /* is 991
/* release all occurrences
REDUCE ARRAY #X-ARR TO  0
WRITE *OCCURRENCE(#X-ARR)              /* is 0
```

Storage Management of X-Group Arrays

If you want to increase or decrease occurrences of X-group arrays, you must distinguish between independent and dependent dimensions.

A dimension which is specified directly (not inherited) for an X-(group) array is *independent*.

A dimension which is *not* specified directly, but inherited for an array is *dependent*.

Only independent dimensions of an X-array can be changed in the statements EXPAND, RESIZE and REDUCE; dependent dimensions must be changed using the name of the corresponding X-group array which owns this dimension as independent dimension.

Example - Independent/Dependent Dimensions:

```
DEFINE  DATA LOCAL
1 #X-GROUP-ARR1(1:*)                  /* (1:*)
  2 #X-ARR1    (I4)                   /* (1:*)
  2 #X-ARR2    (I4/2:*)               /* (1:*,2:*)
  2 #X-GROUP-ARR2                     /* (1:*)
    3 #X-ARR3  (I4)                   /* (1:*)
    3 #X-ARR4  (I4/3:*)               /* (1:*,3:*)
    3 #X-ARR5  (I4/4:*, 5:*)          /* (1:*,4:*,5:*)
END-DEFINE
```

The following table shows whether the dimensions in the above program are independent or dependent.

Name	Dependent Dimension	Independent Dimension
#X-GROUP-ARR1		(1:*)
#X-ARR1	(1:*)	
#X-ARR2	(1:*)	(2:*)
#X-GROUP-ARR2	(1:*)	
#X-ARR3	(1:*)	
#X-ARR4	(1:*)	(3:*)
#X-ARR5	(1:*)	(4:*,5:*)

The only index notation permitted for a dependent dimension is either a single asterisk (*), a range defined with asterisks (*:*) or the index bounds defined.

This is to indicate that the bounds of the dependent dimension must be kept as they are and cannot be changed.

The occurrences of the dependent dimensions can only be changed by manipulating the corresponding array groups.

```
EXPAND ARRAY #X-GROUP-ARR1 TO (1:11)       /* #X-ARR1(1:11) are allocated
                                           /* #X-ARR3(1:11) are allocated
EXPAND ARRAY #X-ARR2 TO (*:*, 2:12)        /* #X-ARR2(1:11, 2:12) are allocated
EXPAND ARRAY #X-ARR2 TO (1:*, 2:12)        /* same as before
EXPAND ARRAY #X-ARR2 TO (*  , 2:12)        /* same as before
EXPAND ARRAY #X-ARR4 TO (*:*, 3:13)        /* #X-ARR4(1:11, 3:13) are allocated
EXPAND ARRAY #X-ARR5 TO (*:*, 4:14, 5:15)  /* #X-ARR5(1:11, 4:14, 5:15) are allocated
```

The EXPAND statements may be coded in an arbitrary order.

The following use of the EXPAND statement is not allowed, since the arrays only have dependent dimensions.

```
EXPAND ARRAY #X-ARR1 TO ...
EXPAND ARRAY #X-GROUP-ARR2 TO ...
EXPAND ARRAY #X-ARR3 TO ...
```

Referencing an X-Array

The occurrences of an X-array must be allocated by an EXPAND or RESIZE statement before they can be accessed. The statements READ, FIND and GET allocate occurrences implicitly if values are obtained from Tamino.

As a general rule, an attempt to address a non existent X-array occurrence leads to a runtime error. In some statements, however, the access to a non materialized X-array field does not cause an error situation if all occurrences of an X-array are referenced using the complete range notation, for example: #X-ARR(*). This applies to

- parameters used in a CALL statement,
- parameters used in the statements CALLNAT, PERFORM, SEND EVENT or OPEN DIALOG, if defined as optional parameters,
- source fields used in a COMPRESS statement,
- output fields supplied in a PRINT statement,
- fields referenced in a RESET statement.

If individual occurrences of a non materialized X-array are referenced in one of these statements, a corresponding error message is issued.

Example:

```
DEFINE DATA LOCAL
1 #X-ARR (A10/1:*) /* X-array only defined, but not allocated
END-DEFINE
RESET #X-ARR(*)    /* no error, because complete field referenced with (*)
RESET #X-ARR(1:3)  /* runtime error, because individual occurrences (1:3) are referenced
END
```

The asterisk (*) notation in an array reference stands for the complete range of a dimension. If the array is an X-array, the asterisk is the index range of the currently allocated lower and upper bound values, which are determined by the system variables *LBOUND and *UBOUND.

Parameter Transfer with X-Arrays

X-arrays that are used as parameters are treated in the same way as constant arrays with regard to the verification of the following:

- format,
- length,
- dimension or
- number of occurrences.

In addition, X-array parameters can also change the number of occurrences using the statement RESIZE, REDUCE or EXPAND. The question if a resize of an X-array parameter is permitted depends on three factors:

- the type of parameter transfer used, that is by reference or by value,
- the definition of the caller or parameter X-array, and
- the type of X-array range being passed on (complete range or subrange).

The following tables demonstrate when an EXPAND, RESIZE or REDUCE statement can be applied to an X-array parameter.

Example with Call By Value

Caller	Parameter		
	Static	Variable (1:V)	X-Array
Static	no	no	yes
X-array subrange, for example: `CALLNAT...#XA(1:5)`	no	no	yes
X-array complete range, for example: `CALLNAT...#XA(*)`	no	no	yes

Call By Reference/Call By Value Result

Caller	Parameter			
	Static	Variable (1:V)	X-Array with a fixed lower bound, e.g. `DEFINE DATA PARAMETER 1 #PX (A10/1:*)`	X-Array with a fixed upper bound, e.g. `DEFINE DATA PARAMETER 1 #PX (A10/*:1)`
Static	no	no	no	no
X-array subrange, for example: `CALLNAT...#XA(1:5)`	no	no	no	no

X-Array with a fixed lower bound, complete range, for example:
 	
`DEFINE DATA LOCAL`
`1 #XA(A10/1:*)`
`...`
`CALLNAT...#XA(*)`
	no	no	yes	no

X-Array with a fixed upper bound, complete range, for example:
 	
`DEFINE DATA LOCAL`
`1 #XA(A10/*:1)`
`...`
`CALLNAT...#XA(*)`
	no	no	no	yes

Parameter Transfer with X-Group Arrays

The declaration of an X-group array implies that each element of the group will have the same values for upper boundary and lower boundary. Therefore, the number of occurrences of dependent dimensions of fields of an X-group array can only be changed when the group name of the X-group array is given with a `RESIZE`, `REDUCE` or `EXPAND` statement (see *Storage Management of X-Group Arrays* above).

Members of X-group arrays may be transferred as parameters to X-group arrays defined in a parameter data area. The group structures of the caller and the callee need not necessarily be identical. A RESIZE, REDUCE or EXPAND done by the callee is only possible as far as the X-group array of the caller stays consistent.

Example - Elements of X-Group Array Passed as Parameters:

Program:

```
DEFINE DATA LOCAL
1 #X-GROUP-ARR1(1:*)                    /* (1:*)
   2 #X-ARR1 (I4)                       /* (1:*)
   2 #X-ARR2 (I4)                       /* (1:*)
1 #X-GROUP-ARR2(1:*)                    /* (1:*)
   2 #X-ARR3 (I4)                       /* (1:*)
   2 #X-ARR4 (I4)                       /* (1:*)
END-DEFINE
...
CALLNAT ... #X-ARR1(*) #X-ARR4(*)
...
END
```

Subprogram:

```
DEFINE DATA PARAMETER
1 #X-GROUP-ARR(1:*)                     /* (1:*)
   2 #X-PAR1 (I4)                       /* (1:*)
   2 #X-PAR2 (I4)                       /* (1:*)
END-DEFINE
...
RESIZE ARRAY #X-GROUP-ARR to (1:5)
...
END
```

The RESIZE statement in the subprogram is not possible. It would result in an inconsistent number of occurrences of the fields defined in the X-group arrays of the program.

X-Array of Dynamic Variables

An X-array of dynamic variables may be allocated by first specifying the number of occurrences using the EXPAND statement and then assigning a value to the previously allocated array occurrences.

Example:

```
DEFINE DATA LOCAL
 1 #X-ARRAY(A/1:*)  DYNAMIC
END-DEFINE
EXPAND  ARRAY   #X-ARRAY TO (1:10)
  /* allocate #X-ARRAY(1) to #X-ARRAY(10) with zero length.
  /* *LENGTH(#X-ARRAY(1:10)) is zero
#X-ARRAY(*) := 'abc'
  /* #X-ARRAY(1:10) contains 'abc',
  /* *LENGTH(#X-ARRAY(1:10)) is 3
EXPAND  ARRAY   #X-ARRAY TO (1:20)
  /* allocate #X-ARRAY(11) to #X-ARRAY(20) with zero length
```

```
  /* *LENGTH(#X-ARRAY(11:20)) is zero
#X-ARRAY(11:20) := 'def'
  /* #X-ARRAY(11:20) contains 'def'
  /* *LENGTH(#X-ARRAY(11:20)) is 3
```

Lower and Upper Bound of an Array

The system variables *LBOUND and *UBOUND contain the current lower and upper bound of an array for the specified dimension(s): (1,2 or 3).

If no occurrences of an X-array have been allocated, the access to *LBOUND or *UBOUND is undefined for the variable index bounds, that is, for the boundaries that are represented by an asterisk (*) character in the index definition, and leads to a runtime error. In order to avoid a runtime error, the system variable *OCCURRENCE may be used to check against zero occurrences before *LBOUND or *UBOUND is evaluated:

Example:

```
IF *OCCURRENCE (#A) NE 0  AND  *UBOUND(#A) < 100 THEN ...
```