SUN/77.8

Starlink Project
Starlink User Note 77.8

D.J. Allan

11 September 1995

NBS
The Noticeboard System
Version 2.5.5
Programmer's Manual

Abstract

This document describes Version 2.5.5 of the Noticeboard System. It is aimed at programmers interested in or intending to use the system and contains all information necessary to use it.

Contents

1 Introduction

The noticeboard system routines provide a fast means for processes to share data in global memory. A given process may own as many noticeboards as it wishes and may access noticeboards owned by other processes. Normally the only process that writes to a noticeboard is its owner but other processes that know what they are doing can subvert this rule either by calling a special routine or else by accessing noticeboard data by using a pointer.

The original interfaces were defined at the AAO workshop in October 1985. V1.0 was implemented in C by William Lupton at RGO in January 1986. The changes for V2.3 onwards were made by David Allan at the University of Birmingham. Refer to Section 7.1 for details.

Noticeboards are identified by name and each can contain a hierarchy of items. Each item has a name, a type, a structure / primitive attribute, and, if primitive, a maximum number of dimensions, a maximum number of bytes, a current shape and a current value. The type and shape are not used by the routines but their values can be put and got and they can be used when implementing higher-level routines on top of the noticeboard routines. Noticeboards are self-defining — a process can find and access data from a noticeboard without knowing anything about what it contains.

2 General Description

2.1 Names and Types

All names and types are converted to upper-case and all white space and non-printing characters in them are ignored. Thus " `William Lupton` " is regarded as "`WILLIAMLUPTON`". An upper limit of 16 characters is imposed on both names and types (the limit applies to the length after the removal of white space and non-printing characters).

2.2 Data Consistency

Provided there is only one writer to a noticeboard and the standard get and put routines are used, the routines will guarantee that consistent data is read from the noticeboard. A writer will never have to wait but a reader will retry potentially until a timeout occurs.

2.3 Static Definition

A (deliberate) restriction of the noticeboard system is that a noticeboard is static in structure. Its structure must be defined before any values are put into it and once the definition is complete no more items can be created. There are a set of routines called `NBS_DEFINE_*` and `NBS_*_DEFINITION` which allow definition of noticeboard contents, saving definitions to and restoring them from file (the initial state of a noticeboard is that all items have zero length values). Only once the definition phase is complete can the `NBS_PUT_*` and `NBS_GET_*` routines be used.

The initial reason for this was efficiency. However it is not in fact particularly difficult to allow new items to created on the fly without compromising efficiency and this restriction should be seen more as a way of preventing the noticeboard routines from being used for purposes for which they were not designed and for which better tools (such as HDS) exist. Note that this "static" restriction refers only to the creation of items in the noticeboard. It is always possible to change item shapes and values.

2.4 Saving to Disc

When saving a noticeboard to disc, the programmer has a choice as to whether to save only the definition or else the definition plus the data. If only the definition is saved then each time that the definition is restored all items will revert to having zero lengths. However if the data is saved as well the `NBS_RESTORE_NOTICEBOARD` routine can be used to restore both the definition and the data, and a subsequent call to `NBS_SAVE_NOTICEBOARD` will update the disc file so that the next time that the noticeboard is restored it will be in the same state as when it was saved.

2.5 C-callable Version

The `NBS` routines have been implemented to make them easy to use from Fortran, and for this reason all character strings are passed as standard Fortran character arguments (address of `VMS` descriptor on `VMS`, character pointer with trailing hidden length argument on most other architectures) and all other parameters are passed by reference.

The routines are actually written in C and it would be unreasonable and wasteful to force C programmers to build descriptors just so that they could be decoded back to the same C strings that they started off as. For this reason, every `NBS` routine (except a few which handle character strings) has an associated `NBC` routine which passes all character parameters as C zero-terminated strings, all input scalar parameters by value and all other parameters by reference. See the source of the demonstration programs described in Appendix C for examples of the use of this C-callable version.

2.6 Portability to UNIX

`NBS` V2.4 was the first version of `NBS` to be fully portable to UNIX – V2.2 onwards was written portably but had several deficiencies and bugs. The shared memory services of UNIX System V are sufficiently close to the `VMS` global section system services to require only a small amount of conditionally compiled code in the low-level "create section" and "map section" routines. There are however two limitations which may affect users planning to use the software on `UNIX` platforms.

- The first and least serious is that `UNIX` does not permit a process to have a shared memory segment mapped more than once the process's address space. `NBS` has been written to take account of this and will always return the same address in a request to map a noticeboard once it is mapped the first time. It does however keep a reference count, and will not unmap a noticeboard until this drops to zero.
- The system identifier (or key) for shared memory sections (and hence noticeboards) on `UNIX` is a four byte integer. NBS therefore hashes the user supplied name into this integer.

The algorithm for doing this is much improved in V2.5.5 but it is still not proof against different names mapping on to the same key value.

2.7 Status Conventions

All the NBS routines use ADAM modified status conventions in that they will do nothing if status is not NBS__OK (0) on entry. They can all optionally be called as functions, in which case the function value is the same as the returned status value. In all cases where an NBS routine is the source of the error, an error report will be made using the error and message reporting service.

2.8 Terminology

These routines use the terminology "item" to refer to either a structured noticeboard object (ie, one with no values but possibly having lower-level objects) or to a primitive noticeboard object (ie, one with a shape and with values). The terms "item identifier", "identifier" or "ID" refer to "handles" (cf HDS locators) which allow access to all information pertaining to an item. As far as callers are concerned these are just integers. As far as the noticeboard system is concerned they are pointers to data structures called "item_descriptor"s. In the routine specifications they are regarded as integers. A zero ID is always an invalid ID and all routines will detect an attempt to use a zero ID. Similarly all routines that return ID's will return zero ID's on failure. Note that there is a danger of an access violation if true garbage ID's are passed. This is unlikely to happen, because of the use of the modified status convention.

3 Examples of Using the NBS Routines

This section takes the form of a tutorial. Starting with a listing of a noticeboard contents, we present the calls necessary to define that noticeboard, to put the required values into it and for another process to access those values.

Suppose that we wish to create and access a simple noticeboard. We will use as an example a subset of the noticeboard that is maintained by the AUTOFIB control software. The notation used is the same as that used by the NBTRACE (Trace Noticeboard) program (see Section C.1) — each structure entry is of the form:

```
type name (children)
```

and each primitive entry is of the form:

```
type name[(maxd) dim1,dim2...] (actb/maxb/mod) val1,val2,...
```

with appropriately indented entries. The shape information is shown only if the maximum number of dimensions (maxd) is greater than zero. The three slash-separated numbers are actual number of bytes, maximum number of bytes and modified count.

The result of running NBTRACE on this noticeboard might be:

```
Software version   = 5 (5)
Size of section    = 13284 (33e4)
Size of definition = 8952 (22f8)
Noticeboard owner  = 405 (195)
Modified count     = 2 (2)

NOTICEBOARD AUTOFIB (3)
   _CHAR            CURRENT_CONFIG         (18/132/2) "current.fib"
   CURRENT_STATUS   CURRENT_STATUS[(1) 72] (0/4608/0)
   FIBRE_PARAMETERS FIBRE_PARAMETERS (1)
       _INTEGER     TRANS_MATRIX[(2) 2,2]  (16/16/2)  1,0,0,1
```

- Type and shape have no significance to the NBS routines — the type is just a character string and the shape is just a list of numbers. In the above, HDS primitive type conventions have been used and this allows a sensible representation of values. Remember though that it is the NBTRACE program that is making use of the type information, not the NBS routines.
- The initial state of an item is that the actual number of bytes and the modified count are both zero. CURRENT_STATUS above is in its initial state.
- It is possible to give an item a shape during the definition phase so an item can initially have a non-scalar shape.
- The noticeboard modified count is incremented just after updating any item in the noticeboard. An "update" may be an update of an item's value, its shape or its size.
- An item's modified count is incremented just before updating an item and just afterwards. Thus if it is odd then an item is in the process of being updated, and the number of completed updates is MODIFIED / 2. An "update" may be an update of either an item's value or its shape.
- You can use the NBS_GET_UPDATED routine to determine whether a noticeboard or a specific item has been updated since the last call to NBS_GET_UPDATED.

3.1 Defining the Noticeboard Contents

The following Fortran-like calls will define the contents of the above noticeboard and save the definition in a disc file.

```
INCLUDE   'NBS_ERR'       ! Error code definitions
INCLUDE   'SAE_PAR'       ! Error code definitions

INTEGER   STATUS          ! Modified STATUS variable
INTEGER   TOPSID          ! Top-level static ID
INTEGER   SID             ! General purpose static ID
INTEGER   FIBSID          ! FIBRE_PARAMETERS static ID
INTEGER   DIMS(2)         ! TRANS_MATRIX dimensions

STATUS = SAI__OK          ! Initially set status to be OK
```

Begin the noticeboard definition.

```
      NBS_BEGIN_DEFINITION (TOPSID,STATUS)      ! Top-level static ID
```

Define the primitive item `CURRENT_CONFIG`. This is a character string of maximum length 132 bytes. We choose to regard it as a scalar.

```
      NBS_DEFINE_PRIMITIVE (TOPSID,             ! Parent static ID
                            'CURRENT_CONFIG',   ! Name of item
                            '_CHAR',            ! Type of item
                            0,132,              ! Max # dims and bytes
                            SID,STATUS)         ! Returned static ID
```

Define the primitive item `CURRENT_STATUS`. This is a 1D array of 72 64 byte records.

```
      NBS_DEFINE_PRIMITIVE (TOPSID,             ! Parent static ID
                            'CURRENT_STATUS',   ! Name of item
                            'CURRENT_STATUS',   ! Type of item
                            1,72*64,            ! Max # dims and bytes
                            SID,STATUS)         ! Returned static ID

      NBS_DEFINE_SHAPE     (SID,                ! Static ID
                            1,72,STATUS)        ! Actual # dims and dims
```

Define the structured item `FIBRE_PARAMETERS`. Structured items cannot have shapes or values.

```
      NBS_DEFINE_STRUCTURE (TOPSID,             ! Parent static ID
                            'FIBRE_PARAMETERS', ! Name of item
                            'FIBRE_PARAMETERS', ! Type of item
                            FIBSID,STATUS)      ! Returned static ID
```

Define the lower-level primitive item `TRANS_MATRIX`. This is a 2D array of 2 x 2 integers.

```
      NBS_DEFINE_PRIMITIVE (FIBSID,             ! Parent static ID
                            'TRANS_MATRIX',     ! Name of item
                            '_INTEGER',         ! Type of item
                            2,4*4,              ! Max # dims and bytes
                            SID,STATUS)         ! Returned static ID
      DIMS(1) = 2
      DIMS(2) = 2
      NBS_DEFINE_SHAPE     (SID,                ! Static ID
                            2,DIMS,STATUS)      ! Actual # dims and dims
```

End the definition, writing to file `AUTOFIB.NBD`.

```
      NBS_END_DEFINITION   ('AUTOFIB',          ! Name of noticeboard file
                            'DEFINITION_SAVE',  ! Write definition to disc
                            STATUS)
```

- All of the routines used during the definition phase are called `NBS_*_DEFINITION` or `NBS_DEFINE_*`.

3.2 Creating the Noticeboard

Having defined the noticeboard contents, the actual noticeboard must be created. There are three ways of doing this, depending on whether the definition is being read from a file, the definition plus data is being read from a file (this is illustrated in Section 3.7), or the noticeboard is being created in memory.

Restore the noticeboard definition from a file and create the noticeboard . . .

```
      NBS_RESTORE_DEFINITION (‘AUTOFIB’,            ! Noticeboard name
                              ‘AUTOFIB’,            ! Name of noticeboard file
                              STATUS)
```

. . . or, instead of issuing the above NBS_END_DEFINITION call, issue this one.

```
      NBS_END_DEFINITION     (‘AUTOFIB’,            ! Noticeboard name
                              ‘CREATE_NOTICEBOARD’, ! Don’t write a file
                              STATUS)
```

- Regardless of which method is used, the situation is the same after either of these calls. The former would normally be used if the noticeboard definition had been compiled earlier by another program and the latter would be used if the noticeboard had just been defined and if no noticeboard definition file was required.
- The noticeboard now exists and is owned by this process. All items in it have an actual size of zero and a modified count of zero.

3.3 Finding the Noticeboard

Any process that wants to use a noticeboard must map it. This is done in precisely the same way by both the noticeboard owner and any other processes wishing to use it.

```
      INTEGER   TOPID                           ! Top-level noticeboard ID
```

Map (find) the noticeboard.

```
      NBS_FIND_NOTICEBOARD (‘AUTOFIB’,          ! Noticeboard name
                            TOPID,STATUS)       ! Top-level ID
```

- This is the first NBS routine that a non-owner need call.
- It does no harm to make multiple calls to this routine, but each call will map the noticeboard into a new part of virtual memory (on VMS only — the UNIX version will return the same virtual memory address) and will return a different ID.
- This TOPID is *not* the same as the TOPSID returned by the NBS_BEGIN_DEFINITION routine. Static identifiers are used only during the definition phase and we are no longer in this phase.

3.4 Putting Values into the Noticeboard

We can now put values into primitive items.

```
INTEGER   CONID                         ! CURRENT_CONFIG ID
INTEGER   STAID                         ! CURRENT_STATUS ID
INTEGER   FIBID                         ! FIBRE_PARAMETERS ID
INTEGER   MATID                         ! COORD_MATRIX ID

CHARACTER STRING*(*)                    ! String to write to CURRENT_CONFIG
PARAMETER (STRING='CURRENT.FIB')
INTEGER   MATRIX(2,2)                   ! Matrix to write to COORD_MATRIX
DATA      MATRIX /1,0,0,1/
```

Get the IDs for all the items we want to write to.

```
NBS_FIND_ITEM (TOPID,'CURRENT_CONFIG  ',CONID,STATUS)
NBS_FIND_ITEM (TOPID,'CURRENT_STATUS  ',STAID,STATUS)
NBS_FIND_ITEM (TOPID,'FIBRE_PARAMETERS',FIBID,STATUS)
NBS_FIND_ITEM (FIBID,'COORD_MATRIX    ',MATID,STATUS)
```

Put values to some of them.

```
NBS_PUT_CVALUE (CONID,0,STRING,STATUS)
NBS_PUT_VALUE  (MATID,0,16,MATRIX,STATUS)
NBS_PUT_VALUE  (CONID,0,LEN(STRING),%REF(STRING),STATUS)
```

- Normally one would get IDs for the items of interest just after finding the noticeboard. Finding an item involves a search and is relatively slow, whereas putting and getting a value is very fast.
- The second parameter to NBS_PUT_VALUE and NBS_PUT_CVALUE is a byte offset. You can use non-zero offsets to put a slice of the data associated with an item. The current size of an item's data is the high-water mark of all PUTs. You can use NBS_PUT_SIZE explicitly to set the size of an item's data.
- Because the NBS routines don't use the type information, it is necessary to pass all values by reference. In Fortran this is the default passing mechanism for all variables except for character strings, hence the routine NBS_PUT_CVALUE to handle this case.
- However, on VMS *only* the %REF function enables the user to pass the actual character pointer to NBS_PUT_VALUE, and supply the length of the character string (or indeed, any other number) as the maximum number of bytes to be read.
- The noticeboard is now in the same state as in the listing at the start of this section. If another process ran NBTRACE at this stage, it would get precisely the same output.

3.5 Getting values from the Noticeboard

Values can be accessed either by copying to and from the user's buffer or else directly to and from the noticeboard. Both methods are illustrated here.

```
INTEGER   ACTBYTES               ! Actual number of bytes stored
INTEGER   POINTER                ! Pointer to MATRIX data
CHARACTER*20  CONF               ! Configuration to be read
```

Get current value of `MATRIX`. We expect the actual number of bytes to be at least 16.

```
NBS_GET_VALUE (MATID,0,16,MATRIX,ACTBYTES,STATUS)
IF (ACTBYTES .LT. 16) THEN
  panic
ENDIF
```

Again, reading character strings is slightly different,

```
NBS_GET_CVALUE (MATID,0,CONF,     ! Use the portable routine
              ACTBYTES,STATUS)
NBS_GET_VALUE (MATID,0,LEN(CONF), ! or the VMS specific call
    %REF(CONFIG),ACTBYTES,STATUS) ! using %REF
```

Alternatively get a pointer to the actual data in the noticeboard.

```
NBS_GET_POINTER (MATID,POINTER,STATUS)
MATRIX_OP       (%VAL(POINTER),STATUS)
```

- The second parameter to `NBS_GET_VALUE` and `NBS_GET_CVALUE` is a byte offset. You can use non-zero offsets to get a slice of the data associated with an item.
- The length of the string passed into `NBS_GET_CVALUE` serves the same purpose as the third argument to `NBS_GET_VALUE` — ie. it limits the amount of data that can be read.
- It is not an `NBS` error if an item contains fewer bytes than are being asked for.
- When accessing data directly via pointer, no checks are possible and the modified count is not (and could not be) accessed.

3.6 Finding out about Items

There is a complete set of enquiry routines that allow programs that know nothing about a noticeboard to navigate through it. `NBTRACE` is just such a program. A few of these routines are illustrated here.

```
INCLUDE   'NBS_PAR'               ! Parameter definitions

CHARACTER NAME*(NBS_K_MAXNAME)    ! Item name
CHARACTER TYPE*(NBS_K_MAXTYPE)    ! Item type
INTEGER   MAXBYTES                ! Maximum number of bytes
INTEGER   ACTBYTES                ! Actual number of bytes
INTEGER   MAXDIMS                 ! Maximum number of dimensions
INTEGER   DIMS(7)                 ! Actual dimensions
INTEGER   ACTDIMS                 ! Actual number of dimensions

NBS_GET_NAME  (MATID,NAME,STATUS)
NBS_GET_TYPE  (MATID,TYPE,STATUS)
NBS_GET_SIZE  (MATID,MAXBYTES,ACTBYTES,STATUS)
MAXDIMS = 7                       ! MAXDIMS is a MODIFIED parameter
NBS_GET_SHAPE (MATID,MAXDIMS,DIMS,ACTDIMS,STATUS)
```

- Higher-level routines may wish to check consistency between shape, size and type. For example, in the above we expect ACTBYTES to be 4 (the number of bytes per pixel) times the product of the dimensions.

3.7 Saving and Restoring of Values

It is possible to save and restore the noticeboard data as well as just the definition by using a different option when ending the noticeboard definition.

End the definition, writing it and the noticeboard data to file AUTOFIB.NBD.

```
NBS_END_DEFINITION   ('AUTOFIB',          ! Name of noticeboard file
                      'NOTICEBOARD_SAVE', ! Write noticeboard to disc
                      STATUS)
```

Restore the noticeboard definition plus data from a file and create the noticeboard.

```
NBS_RESTORE_NOTICEBOARD ('AUTOFIB',       ! Noticeboard name
                         'AUTOFIB',       ! Name of noticeboard file
                         STATUS)
```

Map (find) the noticeboard.

```
NBS_FIND_NOTICEBOARD ('AUTOFIB',          ! Noticeboard name
                      TOPID,STATUS)       ! Top-level ID
```

Get the ID for an item we want to write to and put its value.

```
NBS_FIND_ITEM (TOPID,'CURRENT_CONFIG',CONID,STATUS)
NBS_PUT_CVALUE (CONID,0,STRING,STATUS)
```

Save the noticeboard to disc.

```
        NBS_SAVE_NOTICEBOARD (TOPID,             ! ID of any item on noticeboard
                              STATUS)
```

- NBS_RESTORE_NOTICEBOARD can be used to restore from a file that contains only a definition as well as from a file that contains a definition plus data. However if a file does not contain data the NBS__DATANOTSAVED warning status is returned — this indicates that the noticeboard has been created but that it is in its initial state.
- NBS_RESTORE_DEFINITION cannot be used to restore from a file that contains data in addition to a definition. In cases where both sorts of file are to be accessed, use NBS_RESTORE_NOTICEBOARD.
- The implementation of NBS_SAVE_NOTICEBOARD is inefficient in that the entire definition plus data is copied to disc. Under operating systems that support the mapping of files into virtual memory it could be implemented using an "update section" call.
- NBS_SAVE_NOTICEBOARD does not create a new version of the noticeboard definition file; it always overwrites the data in the file from which it was restored.

3.8 Losing Noticeboards

Noticeboards can be explicitly lost (unmapped). Often you will not need to worry about this and will be happy to allow them to be unmapped on image exit. Sometimes, however, you may require explicit control. Note though that a noticeboard is not actually deleted until the last process unmaps it.

Find a noticeboard and an item within it.

```
        NBS_FIND_NOTICEBOARD (‘AUTOFIB’,         ! Noticeboard name
                              TOPID,STATUS)      ! Top-level ID
        NBS_FIND_ITEM (TOPID,‘CURRENT_CONFIG’,CONID,STATUS)
```

Lose the item and the noticeboard.

```
        NBS_LOSE_ITEM (CONID,‘CHECK’,STATUS)
        NBS_LOSE_NOTICEBOARD (TOPID,‘CHECK’,STATUS)
```

Alternatively, if you like living dangerously, try this.

```
        NBS_LOSE_NOTICEBOARD (TOPID,‘FORCE’,STATUS)
```

- FIND and LOSE calls are normally paired. If the CHECK flag is given, you can only lose items and noticeboards for which all derived IDs have already been lost.
- If you know that there is no chance that any of a noticeboard's or item's derived IDs will be used, you can specify the FORCE flag. This allows the item or noticeboard to be lost regardless of whether any derived IDs are still in use.

4 Compiling and Linking Applications

4.1 Compiling

On both VMS and UNIX systems, FORTRAN source code may include NBS error codes and NBS public constants using the symbolic names NBS_ERR and NBS_PAR respectively. On VMS these are logical names and UNIX the appropriate symbolic links can be made in your current directory by typing,

```
% nbs_dev
```

These links can be deleted using `nbs_dev remove` if required.

The error codes are standard VMS ones and use facility number 1802. This number is one of the ones allocated to the AAO. See Appendix C for a list of the symbolic error codes.

4.2 Linking on VMS

NBS can be linked either in a stand-alone or ADAM fashion. The approriate link commands are,

```
LINK program,NBS_LINK/OPT        ! Stand-alone

ALINK program,NBS_LINK/OPT       ! ADAM version
```

The following LINK options file is used stand-alone,

```
NBS_IMAGE/SHARE
```

and this one for linking with ADAM,

```
NBS_IMAGE_ADAM/SHARE
```

If you are linking in lots of other sub-systems as well, you may need to incorporate these lines into your own options file (unfortunately, LINK options files cannot be nested).

4.3 Linking on UNIX

The linking process on UNIX is even simpler than that of VMS. To link against the stand-alone version simply append a `‘nbs_link‘` to the compiler or loader command line.

```
% f77 program.f ‘nbs_link‘
```

Similarly for the ADAM version,

```
% alink program.f ‘nbs_link_adam‘
```

5 NBS Routines Listed by Category

In this section, routines are listed by category. For each routine there is a one-line description of what it does. There is a full alphabetical description of the routines in Appendix A.

5.1 NBS_TUNE — Routines used to alter global noticeboard system parameters

Two types of parameters can be altered. The first type is global to a process and not to a noticeboard (they are essentially Fortran COMMON block or C static variables). The second type is global to a noticeboard and are thus shared by all processes accessing that noticeboard. The original NBS_TUNE routine is used to alter parameters of the first type and the newer NBS_TUNE_NOTICEBOARD is used to tune parameters of the second type. When a noticeboard is created, its initial parameter values are copied from the current values of the first type.

When a parameter is altered, its previous value is returned and this permits a routine to alter a parameter, use the new value and then restore the parameter to its previous value.

The NBS_FIND, NBS_GET and NBS_PUT routines make rather complicated use of these values. For those parameters which are logical flags, they use the OR of the value of the first type and the value of the second type. For those parameters which are numeric values, they use the value of the first type.

NBS_TUNE: Alter the value of a global parameter

NBS_TUNE_NOTICEBOARD: Alter the value of a noticeboard-specific global parameter

5.2 NBS_DEFINITION — Routines used to define the noticeboard structure

A private memory area is obtained and item definitions are created in it. Storage is not allocated for data but nevertheless data pointers are defined as though the data began at address zero. These pointers are relocated at the time that the definition is either written to disc or else copied to a noticeboard.

NBS_BEGIN_DEFINITION: Begin definition of the contents of a noticeboard and return a static identifier to the top level of the noticeboard

NBS_DEFINE_STRUCTURE: Define a new entry for a structured item within another structured item and return a static identifier to the new item

NBS_DEFINE_PRIMITIVE: Define a new entry for a primitive item within another structured item and return a static identifier to the new item

NBS_DEFINE_SHAPE: Define an initial shape for a primitive item

NBS_END_DEFINITION: End the definition of a noticeboard and then create the noticeboard, save the definition in a file, or save the definition plus data in a file

NBS_RESTORE_DEFINITION: Restore a noticeboard definition from file and create the noticeboard

NBS_RESTORE_NOTICEBOARD: Restore a noticeboard definition and data from file and create the noticeboard

NBS_SAVE_NOTICEBOARD: Save a noticeboard to its noticeboard definition file

5.3 NBS_FIND — Routines that find noticeboards and items within noticeboards

Items can be found either by name or by position. When a noticeboard is found a private copy of its item descriptors is made in which all non NIL pointers are relocated by the virtual address of the start of the relevant noticeboard. Thus, once a noticeboard has been found, all information pertaining to it can in fact be accessed using standard C structures.

NBS_FIND_NOTICEBOARD: Find a named noticeboard and return an identifier to it

NBS_FIND_ITEM: Find an item with a specified name contained in a structure associated with a specified identifier and return the located item's ID

NBS_FIND_NTH_ITEM: Find the Nth item contained in a structure associated with a specified identifier and return the located item's ID

5.4 NBS_LOSE — Routines that lose noticeboards and items within noticeboards

The noticeboard system maintains counts of how many items have been found using NBS_FIND_ITEM and NBS_FIND_NTH_ITEM. The routines in this section can be used to indicate that noticeboards and items are no longer required. When a noticeboard is no longer required, it is marked for deletion. When an item is no longer required, its parent's item count is decremented. Normally, noticeboards and items can only be lost if the count of items found from them is zero (this is to minimise the chance of using an item ID to access a non-existent noticeboard), but this can be overridden.

NBS_LOSE_NOTICEBOARD: Unmap a specified noticeboard

NBS_LOSE_ITEM: Declare an intention never again to use a specified item

5.5 NBS_PUT — Routines that write information to a noticeboard

This can normally only be done by the owner of the noticeboard. The only things which can be written are shape information and data. The item's modified count is incremented both before and after each write using the ADD_INTERLOCKED instruction. The noticeboard's modified count is incremented after each write. These modified counts allow other processes to monitor the noticeboard for changes.

When writing data to an item, it is possible to specify the offset at which the data is to start. The actual size of the item's data is the high-water mark of all the data that has been written to the item. For example, if 10 bytes are written at offset 0, the size is 10 bytes, but if then 100 bytes are written at offset 4000, the size is 4100 bytes, even though bytes 11 to 4000 have not been written. It is guaranteed that all such unwritten bytes are zero. It is occasionally necessary to adjust the actual size of an item's data and there is a routine explicitly to do this. For example, the size of the above item could be adjusted back to 10 bytes. Note that if 1 byte were now written at an offset of 10000, the item size would now be 10001 bytes and the 100 bytes at offset 4000 would retain the values that they were given earlier (they would not be reset to zero).

There are two global parameters which can be altered using the NBS_TUNE_NOTICEBOARD routine and which affect the behaviour of these routines:

1. `WORLD_WRITE` is `FALSE` by default and this prevents processes other than the noticeboard's owner from writing to it. If set `TRUE`, this process will be allowed to write to the noticeboard even if it does not own it. If set `TRUE` then it is possible for a reader to read data that is currently varying, so only do this if you know what you are doing. All processes can always access all noticeboard's directly if they do so via pointer — such access is not affected by the `WORLD_WRITE` flag.

2. `INCREMENT_MODIFY` is `TRUE` by default and this causes the item's modified count to be incremented before and after the update and the noticeboard's modified count to be incremented after the update. If set `FALSE`, neither count is altered. Clearly, if this is done, other processes monitoring the modified counts will not realise that noticeboard data is changing.

There is also a lower-level routine that must only be used by people who really know what they are doing. It increments either the noticeboard modified count or a primitive item's modified count. This *must* be called according to the same conventions as are applied in `NBS_PUT_VALUE`. If this is not done, the whole system is subverted.

Finally, there is a routine which specifies a user-supplied routine to be called whenever an item's shape, data, size or modified count is altered. This routine is called in the context of the process which alters the noticeboard — there are no facilities for notifying directly any other processes (although the user-supplied routine may of course do this itself).

NBS_PUT_VALUE: Put a byte array into a slice of a primitive item associated with a specified identifier

NBS_PUT_CVALUE: Put a character string into a slice of a primitive item associated with a specified identifier

NBS_PUT_SHAPE: Put a new shape to a primitive item associated with a specified identifier

NBS_PUT_SIZE: Put a new size to a primitive item associated with a specified identifier

NBS_INC_MODIFIED: Increment the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

NBS_PUT_TRIGGER: Specify a routine to be called whenever a primitive item is updated

5.6 `NBS_GET` — Routines that read information from a noticeboard

Normally this is a very straightforward operation, but when reading shape information and data it is necessary to check that the owner of the information has not altered it whilst it is being read.

There are three global parameters which can be altered using the `NBS_TUNE_NOTICEBOARD` routine and which affect the behaviour of the shape and data reading routines:

1. `CHECK_MODIFY` is `TRUE` by default and this means that the item's modified count is checked both before and after reading the data. The data is read repeatedly until the value of this count is even and unchanging or until a timeout occurs. If set `FALSE`, no such checks are made and no timeout can occur.

2. `TIMEOUT_COUNT` is 100 by default and is the maximum number of times that the data reading will be tried.

3. `TIMEOUT_INTERVAL` is 100 by default and is the delay in milliseconds between tries.

NBS_GET_VALUE: Get a byte array from a slice of a primitive item associated with the specified identifier

NBS_GET_CVALUE: Get a byte array from a slice of a primitive item associated with the specified identifier and store in a character string

NBS_GET_SHAPE: Get the shape of a primitive item associated with the specified identifier

NBS_GET_MODIFIED: Get the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

NBS_GET_MODIFIED_POINTER: Get a pointer to the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

NBS_GET_UPDATED: Determine whether a primitive item or the noticeboard has been updated since the noticeboard was found or this routine was last called.

NBS_GET_POINTER: Return a pointer to the first byte of the data of a primitive item associated with the specified identifier

NBS_GET_NAME: Get the name of an item associated with the specified identifier

NBS_GET_TYPE: Get the type of an item associated with the specified identifier

NBS_GET_SIZE: Get the maximum and actual sizes of a primitive item associated with the specified identifier

NBS_GET_PRIMITIVE: Determine whether or not an item is primitive

NBS_GET_PARENT: Get the identifier of an item's parent structure

NBS_GET_CHILDREN: Get the number of children of a structured item

NBS_GET_INFO: Get general non-character information on a given noticeboard

NBS_GET_CINFO: Get general character information on a given noticeboard

6 Routine Timings

Several operations are tested below on all architectures on which NBS has been extensively tested. The figures given are for a noticeboard with 100 items all at the same level and are the *lowest* measured values of `CPU` microseconds:

Test #	Test name	μVAX 3500	DEC Alpha AXP 3000/400	SPARCstation 10
0	Define, save and find noticeboard	820000	16666	16666
1	Scalar assignment	1	0.1	-
2	Put scalar	36	2.0	1.6
3	Get scalar	34	1.9	1.0
4	Move 10 longwords directly	17	0.3	0.6
5	Put 10 longword array	44	2.2	3.3
6	Get 10 longword array	43	2.2	3.2
7	Move 1024 longwords directly	591	16	87
8	Put 1024 longword array	670	33	89
9	Get 1024 longword array	646	32	89
10	Find 100th item by position	277	35	23
11	Find 100th item by name	544	58	31
12	Enquire name	42	1.2	2.5
13	Enquire size	15	0.7	1.0

These are fairly consistent with the times reported for the MON system. They are also pretty close to those for V2.2 of the noticeboard system and are better in some cases, presumably partly because of the greater use of inline code in V2.3.1 onwards. The speed of the UNIX machine is particularly noticeable on file related operations where the speed up is an order of magnitude greater those operations simply involved with moving data around.

7 Omissions and Future Plans

The following are possible improvements and developments:

- There are no provided exit handlers and if a process crashed in the middle of an update then a modified count could get left as an odd number which would cause NBS_GET_VALUE and NBS_GET_SHAPE to time out.
- There are inefficiencies associated with the definition and creation (not the use) of noticeboards with large numbers of sibling items. Definition involves linear searches with character comparison through linked lists and creation involves very deep recursion.
- It has been proposed that tables (rather like HDS structure arrays) should be supported.
- It has been proposed that it should be possible to associate "extra" information with noticeboard items. This could be used for holding things like units.
- The restriction on the names of UNIX noticeboards is a nuisance. Perhaps a global list of names could be maintained, but this system may be succeptible to access conflicts on a system with intensive use of NBS.

7.1 New Features

The only major new feature in V2.5 is that the noticeboard listing program previously named `ln` has been renamed to `nbtrace` and is now installed during the NBS installation process.

7.2 Bug Fixes

Two problems are fixed in V2.5.5. NBS has hitherto allocated the data for primitive noticeboard items using exactly the data size supplied in the item definition. In V2.5.5 this size is rounded up to a multiple of the size of a double precision number. This ensures that items composed of types likely to mapped using `NBS_GET_POINTER` or `nbc_get_pointer` will be correctly aligned.

The second problem concerns the algorithm used to generate UNIX shared memory identifiers (4 byte integers) from memory section names. The method used in previous releases simply copied the values of the first three characters into an integer. The new algorithm adds the value of each character code in the name shifted leftwards by a number of bits equal to its position in the string. This means that all the characters in the name are significant (only 16 are allowed, which means all the bits in the last character are significant if used), and that anagrams of names map on to different key values.

A Description of Individual NBS Routines

These descriptions are taken directly from the code. All of the routines are written in C and some C-specific terms are used.

NBS_TUNE
Alter the value of a global parameter

Description:

Check that the parameter name is legal.
Copy the previous value of the parameter to the supplied variable.
Alter the specified global parameter.

There are currently six global parameters which can be altered in this way:

[MAX_DEFN_SIZE:] an integer that indicates how much memory should be allocated for building the noticeboard definition during the definition phase. The default is 32768 bytes.

[TIMEOUT_COUNT:] an integer that indicates how many times to loop before timing out when finding noticeboards or when getting values or shapes. The default is 100.

[TIMEOUT_INTERVAL:] an integer that indicates how many milliseconds to wait between attempts at finding noticeboards or when getting values or shapes. The default is 100.

[WORLD_WRITE:] a logical (only the lsb is used) that indicates whether the world (ie non-owners) can write to noticeboards. The default is FALSE.

[INCREMENT_MODIFY:] a logical (only the lsb is used) that indicates whether the modified count should be incremented when putting the values of items. The default is TRUE.

[CHECK_MODIFY:] a logical (only the lsb is used) that indicates whether the modified count should be checked when getting the values of items. The default is TRUE.

Note that the parameters which can be altered are global to a process and not to a noticeboard (they are essentially Fortran COMMON block or C static variables). When a parameter is altered its previous value is returned and this permits a routine to alter a parameter, use the new value and then restore the parameter to its previous value.

The NBS_FIND, NBS_GET and NBS_PUT routines make rather complicated use of these values. For those parameters which are logical flags, they use the OR of the default value (or the value set using NBS_TUNE) and the value set using NBS_TUNE_NOTICEBOARD. For those parameters which are numeric values, they use the value set using NBS_TUNE.

When a noticeboard is created, it inherits the default values or the values set using NBS_TUNE and these values may subsequently be altered using the
NBS_TUNE_NOTICEBOARD routine.

Invocation:

```
(Int) = NBS_TUNE (NAME,VALUE,OLDVALUE,STATUS)
```

Arguments:

NAME = CHARACTER$*$($*$) (Given)
The name of the parameter to alter. See the above list. Can be abbreviated so long as it remains unambiguous but this is not recommended because new parameters may be supported in the future. Case is not significant.

VALUE = INTEGER (Given)
The value that the parameter is to take.

OLDVALUE = INTEGER (Returned)
The old value of the parameter.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__BADOPTION` Illegal parameter name

Prior Requirements :

None.

NBS_TUNE_NOTICEBOARD
Alter the value of a noticeboard-specific global parameter

Description:

Check that the parameter name is legal.
Copy the previous value of the parameter to the supplied variable.
Alter the specified value in the noticeboard.

There are currently three global parameters which can be altered in this way:

[WORLD_WRITE:] a logical (only the lsb is used) that indicates whether the world (ie non-owners) can write to noticeboards. The default is FALSE.

[INCREMENT_MODIFY:] a logical (only the lsb is used) that indicates whether the modified count should be incremented when putting the values of items. The default is TRUE.

[CHECK_MODIFY:] a logical (only the lsb is used) that indicates whether the modified count should be checked when getting the values of items. The default is TRUE.

Note that the parameters which can be altered are global to a specific noticeboard. When a parameter is altered its previous value is returned and this permits a routine to alter a parameter, use the new value and then restore the parameter to its previous value.

The NBS_FIND, NBS_GET and NBS_PUT routines make rather complicated use of these values. They use the OR of the default value (or the value set using NBS_TUNE) and the value set using NBS_TUNE_NOTICEBOARD.

When a noticeboard is created, it inherits the default values or the values set using NBS_TUNE and these values may subsequently be altered using the
NBS_TUNE_NOTICEBOARD routine.

Invocation:

```
(Int) = NBS_TUNE_NOTICEBOARD (ID,NAME,VALUE,OLDVALUE,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of noticeboard or of any item in it, whose parameter value is to be altered.

NAME = CHARACTER∗(∗) (Given)
The name of the parameter to alter. See the above list. Can be abbreviated so long as it remains unambiguous but this is not recommended because new parameters may be supported in the future. Case is not significant.

VALUE = INTEGER (Given)
The value that the parameter is to take.

OLDVALUE = INTEGER (Returned)
The old value of the parameter.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__BADOPTION`	Illegal parameter name

NBS_BEGIN_DEFINITION
Begin definition of the contents of a noticeboard and return a static identifier to the top level of the noticeboard

Description:

Check that we are not currently in the middle of defining a noticeboard.
Allocate the memory area in which the noticeboard definition is built.
From this area, allocate space to describe the new noticeboard and fill in fields appropriate to the top level of a noticeboard.
Remember the address of the memory area and note that we are now in the middle of defining a noticeboard.
Return the address of the item descriptor to the caller for use in subsequent calls.

Invocation:

```
(Int) = NBS_BEGIN_DEFINITION (SID,STATUS)
```

Arguments:

SID = INTEGER (Returned)

Static identifier of the top-level of the noticeboard. This should be used in subsequent calls to the NBS_DEFINE_$*$ routines.

STATUS = INTEGER (Given and returned)

The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__BADOPTION`	Illegal parameter name
`NBS__DEFINING`	Already defining a noticeboard
`NBS__INITALLOCFAILED`	Storage allocation failed
`NBS__NOMOREROOM`	Storage area is full up

NBS_DEFINE_STRUCTURE
Define a new entry for a structured item within another structured item and return a static identifier to the new item

Description:

Check that we are currently in the middle of defining a noticeboard.
Check that the environment static ID is not NIL and does not pertain to a primitive item.
Allocate space to describe the new item and fill in fields appropriate to a structured item such that items at this level are in alphabetical order. (If an item of this name already exists, create a new item but position it before the existing item).
Return the address of the item descriptor to the caller for use in subsequent calls.

Invocation:

```
(Int) = NBS_DEFINE_STRUCTURE (ENVSID,NAME,TYPE,SID,STATUS)
```

Arguments:

ENVSID = INTEGER (Given)
Static identifier of the item in the noticeboard which is the parent of the item to be created.

NAME = CHARACTER*(*) (Given)
Name of the new item.

TYPE = CHARACTER*(*) (Given)
Type of the new item.

SID = INTEGER (Returned)
Static identifier of the new structured item. This should be used in subsequent calls to the NBS_DEFINE_* routines.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NOTDEFINING`	Not currently defining a noticeboard
`NBS__NILSID`	NIL static ID
`NBS__PRIMITIVE`	Prospective parent is primitive
`NBS__NOMOREROOM`	Storage area is full up

NBS_DEFINE_PRIMITIVE
Define a new entry for a primitive item within another structured item and return a static identifier to the new item

Description:

Check that we are currently in the middle of defining a noticeboard.
Check that the environment static ID is not NIL and does not pertain to a primitive item.
Allocate space to describe the new item and fill in fields appropriate to a primitive item such that items at this level are in alphabetical order. (If an item of this name already exists, create a new item but position it before the existing item).
Return the address of the item descriptor to the caller for use in subsequent calls.

Invocation:

```
(Int) = NBS_DEFINE_PRIMITIVE (ENVSID,NAME,TYPE,MAXDIMS,MAXBYTES,SID,
STATUS)
```

Arguments:

ENVSID = INTEGER (Given)
Static identifier of the item in the noticeboard which is the parent of the item to be created.

NAME = CHARACTER*(*) (Given)
Name of the new item.

TYPE = CHARACTER*(*) (Given)
Type of the new item.

MAXDIMS = INTEGER (Given)
Maximum number of dimensions possessed by this item.

MAXBYTES = INTEGER (Given)
Maximum number of bytes in this item's value

SID = INTEGER (Returned)
Static identifier of the new structured item. This should be used in subsequent calls to the NBS_DEFINE_* routines (only NBS_DEFINE_SHAPE is permitted though).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NOTDEFINING`	Not currently defining a noticeboard
`NBS__NILSID`	NIL static ID
`NBS__PRIMITIVE`	Prospective parent is primitive
`NBS__NOMOREROOM`	Storage area is full up

NBS_DEFINE_SHAPE
Define an initial shape for a primitive item

Description:

Check that we are currently in the middle of defining a noticeboard.
Check that the environment static ID is not NIL and pertains to a primitive item.
Check that the requested number of dimensions is not too large.
Copy the shape information to the relevant parts of the item information.

Invocation:

```
(Int) = NBS_DEFINE_SHAPE (SID,NDIMS,DIMS,STATUS)
```

Arguments:

SID = INTEGER (Given)
Static identifier of the item in the noticeboard which is to be given an initial shape.

NDIM = INTEGER (Given)
Number of dimensions.

DIMS = INTEGER(∗) (Given)
Dimensions.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NOTDEFINING`	Not currently defining a noticeboard
`NBS__NILSID`	NIL static ID
`NBS__PRIMITIVE`	Prospective parent is primitive
`NBS__TOOMANYDIMS`	Too many dimensions

NBS_END_DEFINITION
End the definition of a noticeboard and then create the noticeboard, save the definition in a file, or save the definition plus data in a file

Description:

Check that we are currently in the middle of defining a noticeboard.
Calculate how large the definition and data parts of the noticeboard are.
Relocate all pointers in the definition so that they are relative to zero rather than being actual program virtual addresses (actually they are made relative to a small positive integer to avoid problems with zero pointers).
If the option parameter indicates, write the definition and optionally the data to a file (a default file extension of .NBD is applied).
Otherwise, create the noticeboard, copy the definition to it, write the calling process' id to the global section to denote ownership and mark the noticeboard as being valid.
De-allocate the memory area used to amass the noticeboard definition and note that we are no longer defining a noticeboard.

If the noticeboard already existed, NBS__SECTIONEXISTED status is returned and the calling process becomes its owner.

Invocation:

```
(int) = NBS_END_DEFINITION (NAME,OPTION,STATUS)
```

Arguments:

NAME = CHARACTER*(*) (Given)
If OPTION is DEFINITION_SAVE or NOTICEBOARD_SAVE, the name of the file to write the definition or definition plus data to (with a default file type of .NBD). Otherwise (OPTION is CREATE_NOTICEBOARD) the name of the noticeboard to create.

OPTION = CHARACTER*(*) (Given)
Option that governs whether the noticeboard definition or definition plus data is saved to a file or whether the noticeboard is simply created on the spot without being associated with a file. Can be abbreviated so long as it remains unambiguous but this is not recommended because new options may be supported in the future. Case is not significant. Possible values are:

[DEFINITION_SAVE:] Save the definition to a file that does not contain space allocated for the data.

[NOTICEBOARD_SAVE:] Save the definition to a file that does contains space allocated for the data.

[CREATE_NOTICEBOARD:] Create the noticeboard immediately without associating it with a file. This is assumed if an illegal value of OPTION is given.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NOTDEFINING`	Not currently defining a noticeboard
`NBS__CANTOPEN`	Can't create the definition file
`NBS__CANTWRITE`	Can't write the definition file
`NBS__SECTIONEXISTED`	Noticeboard of this name already existed.

SS$_*	System service codes from SYS$CRMPSC

NBS_RESTORE_DEFINITION
Restore a noticeboard definition from file and create the noticeboard

Description:

Open the file and determine the noticeboard size.
Check that the data part was not saved to the file.
Create the noticeboard.
Read the definition part of the file into the noticeboard.
Write the calling process' id to the noticeboard to denote ownership and mark the noticeboard as being valid.
Close the file.

If the noticeboard already existed, NBS__SECTIONEXISTED status is returned and the calling process becomes its owner.

Invocation:

```
(Int) = NBS_RESTORE_DEFINITION (NAME,SAVE_NAME,STATUS)
```

Arguments:

NAME = CHARACTER*(*) (Given)
The name to give the noticeboard (and thus the name of the noticeboard).

SAVE_NAME = CHARACTER*(*) (Given)
The name of the file from which to read the definition (with a default file type of .NBD)

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__CANTOPEN`	Can't create the definition file
`NBS__DATASAVED`	Noticeboard data was saved to the definition file — cannot restore only defn
`NBS__CANTREAD`	Can't read the definition file
`NBS__BADVERSION`	Wrong version in definition file
`NBS__SECTIONEXISTED`	Noticeboard of this name already existed.
`SS$_*`	System service codes from `SYS$CRMPSC`

Prior Requirements :

None.

NBS_RESTORE_NOTICEBOARD
Restore a noticeboard definition and data from file and create the noticeboard

Description:

Open the file and determine the noticeboard size.
Create the noticeboard.
Read the file into the noticeboard.
Write the calling process' id to the noticeboard to denote ownership and mark the noticeboard as being valid.
Close the file.
If the file only contained the definition and not the noticeboard, return a warning status.

If the noticeboard already existed, NBS__SECTIONEXISTED status is returned and the calling process becomes its owner.

Invocation:

```
(Int) = NBS_RESTORE_NOTICEBOARD (NAME,SAVE_NAME,STATUS)
```

Arguments:

NAME = CHARACTER∗(∗) (Given)
The name to give the noticeboard.

SAVE_NAME = CHARACTER∗(∗) (Given)
The name of the file from which to read the definition (with a default file type of .NBD)

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__CANTOPEN`	Can't create the definition file
`NBS__CANTREAD`	Can't read the definition file
`NBS__BADVERSION`	Wrong version in definition file
`NBS__SECTIONEXISTED`	Noticeboard of this name already existed.
`NBS__DATANOTSAVED`	Noticeboard data not saved to the definition file, so not restored
`SS$_*`	System service codes from `SYS$CRMPSC` or `SYS$DELTVA` (VMS only).

Prior Requirements :

None.

NBS_SAVE_NOTICEBOARD
Save a noticeboard to its noticeboard definition file

Description:

Check that the ID is not NIL.
Check that the caller owns the noticeboard.
Check that the noticeboard was restored from a file which has room for the noticeboard data.
Check that the file is open for write access.
Write the noticeboard data to it.

Invocation:

```
(Int) = NBS_SAVE_NOTICEBOARD (ID,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of noticeboard or of any item in the noticeboard whose data is to be saved.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__CANTOPEN`	Can't open the definition file
`NBS__NOTOWNER`	Caller does not own the noticeboard
`NBS__DATANOTRESTORED`	Noticeboard data was not restored from the definition file (so can't save it)

Prior Requirements :

None.

NBS_FIND_NOTICEBOARD
Find a named noticeboard and return an identifier to it

Description:

Map the noticeboard of the given name.
Allocate an item descriptor and relocate it so that the version number can be checked and the definition size determined.
Free that item descriptor and allocate a block of memory big enough for the entire definition.
Copy the definition part to this memory and relocate it so that all pointers are once again virtual memory addresses.
Get the process' id for future checking against the owner's process id.
If this is the owner process and it hasn't already been done, unmap the copy mapped earlier.

Note that it is somewhat wasteful to build a complete copy of the noticeboard definition in private memory when in fact only the pointers must have private versions. A later version of the software should take copies only of the pointers but this requires the pointer space to be allocated separately during the definition phase — in the current implementation the pointer space is not contiguous; pointers are mixed up with fixed information, shape information and board information.

Invocation:

```
(Int) = NBS_FIND_NOTICEBOARD (NAME,ID,STATUS)
```

Arguments:

NAME = CHARACTER*(*) (Given)
The name of the noticeboard which is to be found.

ID = INTEGER (Returned)
Identifier of the top-level of the noticeboard.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

NBS__SECTIONNOTFOUND	No section called NAME existed
NBS__TIMEOUT	Timeout awaiting valid noticeboard
NBS__NOMOREROOM	Failed to allocate private memory area
NBS__BADVERSION	Wrong version in noticeboard
SS$_*	System service codes from SYS$CRMPSC or SYS$DELTVA (VMS only).

NBS_FIND_ITEM
Find an item with a specified name contained in a structure associated with a specified identifier and return the located item's ID

Description:

Check that the environment ID is not NIL and does not pertain to a primitive item.
Search for an item of the required name (don't assume that they are in any particular order).

The searching is performed using a binary search. This means that it takes roughly the same time to find all the items — best case performance is degraded but worst case performance is substantially improved.

Invocation:

```
(Int) = NBS_FIND_ITEM (ENVID,NAME,ID,STATUS)
```

Arguments:

ENVID = INTEGER (Given)
Identifier of the parent of the item which is to be found.

NAME = CHARACTER*(*) (Given)
The name of the item to be found.

ID = INTEGER (Returned)
Identifier of the found item (zero if not found).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

NBS__NILID	NIL ID
NBS__PRIMITIVE	Parent is primitive
NBS__ITEMNOTFOUND	No item of this name exists

NBS_FIND_NTH_ITEM
Find the Nth item contained in a structure associated with a specified identifier and return the located item's ID

Description:

Check that the environment ID is not NIL and does not pertain to a primitive item.
Extract the Nth item.

Invocation:

```
(Int) = NBS_FIND_NTH_ITEM (ENVID,POSN,ID,STATUS)
```

Arguments:

ENVID = INTEGER (Given)
Identifier of the parent of the item which is to be found.

POSN = INTEGER (Given)
Number of item to find (first item is item #1).

ID = INTEGER (Returned)
Identifier of the found item (zero if not found).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__PRIMITIVE`	Parent is primitive
`NBS__ITEMNOTFOUND`	No item of this name exists

NBS_LOSE_NOTICEBOARD
Unmap a specified noticeboard

Description:

Check that the ID is indeed a top-level one.
Check that no items are currently derived from the top level of the noticeboard (not if the "FORCE" option is specified).
Unmap the noticeboard.
Free the local copy of the noticeboard definition.

Calls to this routine should match calls to NBS_FIND_NOTICEBOARD. After calling this routine, all identifiers associated with this noticeboard will be invalid. The noticeboard will only be deleted if no other process has it mapped.

Invocation:

```
(Int) = NBS_LOSE_NOTICEBOARD (ID,OPTION,STATUS)
```

Arguments:

ID = INTEGER (Given)

Identifier of the top-level of the noticeboard.

OPTION = CHARACTER$*$($*$) (Given)

Option that governs whether to check that there are no identifiers currently derived from this one. Can be abbreviated so long as it remains unambiguous but this is not recommended because new options may be supported in the future. Case is not significant. Possible values are:

[FORCE:] Unmap the noticeboard regardless of whether there or not there are identifiers derived from this one.

[CHECK:] Check that no identifiers are derived from this one (assumed if invalid option is given).

STATUS = INTEGER (Given and returned)

The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTTOPLEVEL`	ID is not a top-level identifier
`NBS__HASIDS`	Noticeboard has identifiers derived from it
`SS$_*`	System service codes from `SYS$DELTVA` (VMS only).

NBS_LOSE_ITEM
Declare an intention never again to use a specified item

Description:

Check that the ID is not a top-level one.
Check that at least one item is derived from this item's parent.
Check that no items are currently derived from this item (not if the "FORCE" option is specified).
Decrement the parent's count of derived items.

Calls to this routine should match calls to NBS_FIND_ITEM / NBS_FIND_NTH_ITEM. After calling it, this identifier should not be used again (even though it is in fact still valid until NBS_LOSE_NOTICEBOARD is called).

Invocation:

```
(Int) = NBS_LOSE_ITEM (ID,OPTION,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item which is to be lost.

OPTION = CHARACTER*(*) (Given)
Option that governs whether to check that there are no identifiers currently derived from this one. Can be abbreviated so long as it remains unambiguous but this is not recommended because new options may be supported in the future. Case is not significant. Possible values are:

[FORCE:] Unmap the item regardless of whether there are identifiers derived from this one.

[CHECK:] Check that no identifiers are derived from this one (assumed if invalid option is given).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTTOPLEVEL`	ID is not a top-level identifier
`NBS__NEVERFOUND`	Item was never found (or more items have been lost than were found)
`NBS__HASIDS`	Noticeboard has identifiers derived from it
`SS$_*`	System service codes from `SYS$DELTVA` (VMS only).

NBS_PUT_VALUE
Put a byte array into a slice of a primitive item associated with a specified identifier

Description:

Check that the ID is not NIL and that it pertains to a primitive item.
Check that the caller owns the noticeboard (or WORLD_WRITE is TRUE).
Check that the offset into the data is not negative.
Check that the item is large enough to accept all of the supplied values.
Increment the item's modified count.
Update the item size (maintain a high-water mark).
Copy the values to the noticeboard.
Increment the item's modified count again.
Increment the noticeboard modified count.

The item and noticeboard modified counts will not be incremented if INCREMENT_MODIFY is FALSE.

Note that this routine only alters the specified number of bytes starting at the specified offset — all bytes are initially zero and bytes above and below those that are being altered will not be affected. The actual size of the item will be adjusted upwards if the new data extends past the previous end of the data but the size of the item cannot be decreased by this routine. To alter the size of an item, use the NBS_PUT_SIZE routine.

Invocation:

```
(Int) = NBS_PUT_VALUE (ID,OFFSET,NBYTES,BYTE_ARRAY,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item which the value is to be put.

OFFSET = INTEGER (Given)
Byte offset into item data.

NBYTES = INTEGER (Given)
Number of bytes to put.

BYTES = BYTE(∗) (Given)
Bytes to be put.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__NOTOWNER`	Caller does not own the noticeboard
`NBS__BADOFFSET`	Negative offset specified
`NBS__TOOMANYBYTES`	Not room to put all the data

Notes:

In versions prior to {V2.4.0} this routine could be used to write character strings on the VAX using the %REF() mechanism to pass the character data. A new routine, NBS_PUT_CVALUE, has been provided to remove the need for %REF and make the method portable.

NBS_PUT_CVALUE
Put a character string into a slice of a primitive item associated with a specified identifier

Description:

Simply extract the string data pointer and length and pass this information to NBS_PUT_VALUE.

Invocation:

```
(Int) = NBS_PUT_CVALUE (ID,OFFSET,STRING,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item which the value is to be put.

OFFSET = INTEGER (Given)
Byte offset into item data.

STRING = CHARACTER∗(∗) (Given)
The string to be put.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__NOTOWNER`	Caller does not own the noticeboard
`NBS__BADOFFSET`	Negative offset specified
`NBS__TOOMANYBYTES`	Not room to put all the data

Notes:

Replaces NBS_PUT_VALUE when writing character strings to noticeboards.

NBS_PUT_SHAPE
Put a new shape to a primitive item associated with a specified identifier

Description:

Check that the ID is not NIL and that it pertains to a primitive item.
Check that the caller owns the noticeboard (or WORLD_WRITE is TRUE).
Check that the item has enough potential dimensions to accept all of the supplied dimensions.
Increment the modified count for this item.
Copy the dimensions to the noticeboard.
Increment the modified count once more.
Increment the noticeboard modified count.

The item and noticeboard modified counts will not be incremented if INCREMENT_MODIFY is FALSE.

Invocation:

```
(Int) = NBS_PUT_SHAPE (ID,NDIMS,DIMS,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item which the shape is to be put.

NDIM = INTEGER (Given)
Number of dimensions to be put.

DIMS = INTEGER(∗) (Given)
Dimensions to be put.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__NOTOWNER`	Caller does not own the noticeboard
`NBS__TOOMANYDIMS`	NDIMS is greater than item max dims

NBS_PUT_SIZE
Put a new size to a primitive item associated with a specified identifier

Description:

Check that the ID is not NIL and that it pertains to a primitive item.
Check that the caller owns the noticeboard (or WORLD_WRITE is TRUE).
Check that the item is large enough to be the proposed size.
Increment the item's modified count.
Update the internal record of the item's size.
Increment the item's modified count again.
Increment the noticeboard modified count.

The item and noticeboard modified counts will not be incremented if INCREMENT_MODIFY is FALSE.

Invocation:

```
(Int) = NBS_PUT_SIZE (ID,NBYTES,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item which the shape is to be put.

NBYTES = INTEGER (Given)
new item size in bytes.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__NOTOWNER`	Caller does not own the noticeboard
`NBS__TOOMANYBYTES`	NBYTES is greater than item size

NBS_INC_MODIFIED
Increment the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

Description:

Check that the ID is not NIL.
Check that the caller owns the noticeboard (or WORLD_WRITE is TRUE).
If the item is structured increment the noticeboard modified count.
If the item is primitive increment the item's modified count.

Note that this is a very dangerous routine when called on behalf of primitive items. Calls to it *must* be paired. Any reader of an item will time out if the modified count for the item being read is an odd number.

Invocation:

```
(Int) = NBS_INC_MODIFIED (ID,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item whose modified count is be incremented.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTOWNER`	Caller does not own the noticeboard

NBS_PUT_TRIGGER
Specify a routine to be called whenever a primitive item is updated

Description:

Check that the ID is not NIL and that it pertains to a primitive item.
Check that the caller owns the noticeboard (or WORLD_WRITE is TRUE).
Copy the address of the routine to be called on item update.

In this context `"update"` means any change to the item`'`s shape, data, size or modified count. The supplied routine is called with the following calling sequence:

TRIGGER (ID,STATUS)

where ID is the identifier of the item which has been altered and STATUS is as usual. Any bad status returned by the trigger routine will be passed back to the caller.

Invocation:

```
(Int) = NBS_PUT_TRIGGER (ID,TRIGGER,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for which a trigger routine is to be specified.

TRIGGER = EXTERNAL (Given)
The address of the routine to call whenever the item is updated. From FORTRAN, declare it as EXTERNAL. Pass zero (requires %VAL(0) from FORTRAN) to disable the facility).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTOWNER`	Caller does not own the noticeboard

NBS_GET_VALUE
Get a byte array from a slice of a primitive item associated with the specified identifier

Description:

```
Check that the ID is not NIL and that it pertains to a primitive item.
Check that the offset into the data is not negative.
Repeat
    {
    Read the modified count for this item.
    Copy as many bytes as there is room for in the user's buffer from the noticeboard starting
    at the specified offset and return the actual number of bytes in the item.
    Read the modified count for this item once more.
    }
Until time out or the two modified counts are equal and even
    (which means that the values were not updated whilst they were being read).
```

If CHECK_MODIFY is FALSE, the item's modified count is not checked at all and a timeout cannot occur.

If the specified offset is greater than the current size of the item's data, no error status will be returned and no data will be copied, but the returned number of bytes (ACTBYTES) will be less than the offset (OFFSET) and this case should always be checked for.

Invocation:

```
(Int) = NBS_GET_VALUE (ID,OFFSET,MAXBYTES,BYTE_ARRAY,ACTBYTES,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item from which thew value is to be got.

OFFSET = INTEGER (Given)
Byte offset into item data.

MAXBYTES = INTEGER (Given)
Size in bytes of the user's buffer.

BYTE_ARRAY = BYTE(∗) (Returned)
User's buffer into which bytes will be got.

ACTBYTES = INTEGER (Returned)
Actual number of values associated with the item. This may be greater than OFFSET + MAXBYTES but no more than MAXBYTES bytes will be copied into the user's buffer.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

NBS__NILID	NIL ID
NBS__NOTPRIMITIVE	Item is not primitive
NBS__BADOFFSET	Negative offset specified
NBS__TIMEOUT	Timeout awaiting valid data

NBS_GET_CVALUE
Get a character string from a slice of a primitive item associated with the specified identifier

Description:

Extracts the string data pointer and string length from the FORTRAN argument list, and passes this information to NBS_GET_VALUE. Thus, the number of bytes read is at most the length the string supplied.

Invocation:

```
(Int) = NBS_GET_CVALUE (ID,OFFSET,STRING,ACTBYTES,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item from which thew value is to be got.

OFFSET = INTEGER (Given)
Byte offset into item data.

STRING = CHARACTER$*$($*$) (Returned)
Users string buffer into which item bytes will be got.

ACTBYTES = INTEGER (Returned)
Actual number of values associated with the item. This may be greater than OFFSET + LEN(STRING) but no more than LEN(STRING) bytes will be copied into the user's buffer.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__BADOFFSET`	Negative offset specified
`NBS__TIMEOUT`	Timeout awaiting valid data

Notes:

No C version of this routine is supplied because C strings are by convention null terminated. As NBS_GET_CVALUE writes new string data, its correct functioning in the C case would rely on the unwarranted assumption that sufficient space existed in the destination string for the data to be written. By forcing the use of NBS_GET_VALUE in this case, the user must at least state the destination length explicitly.

NBS_GET_SHAPE
Get the shape of a primitive item associated with the specified identifier

Description:

```
Check that the ID is not NIL and that it pertains to a primitive item.
Repeat
    {
    Read the modified count for this item.
    Copy as many dimensions as there is room for in the user's buffer from the noticeboard and
    return the actual number of dimensions in the item.
    Read the modified count for this item once more.
    }
    Until time out or the two modified counts are equal and even
    (which means that the values were not updated whilst they were being read).
```

If CHECK_MODIFY is FALSE, the item's modified count is not checked at all and a timeout cannot occur.

Note also that the MAXDIMS parameter to this routine is a MODIFIED parameter.

Invocation:

```
(Int) = NBS_GET_SHAPE (ID,MAXDIMS,DIMS,ACTDIMS,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item from which the shape is to be got.

MAXDIMS = INTEGER (Given and returned)
On entry, size of the DIMS array. On exit the maximum number of dimensions that this item can have.

DIMS = INTEGER($*$) (Returned)
Returned dimensions.

ACTDIMS = INTEGER (Returned)
Actual number of dimensions associated with the item. This may be greater than MAXDIMS but no more than MAXDIMS values will be copied into the DIMS array.

MAXBYTES = INTEGER (Given)
Size in bytes of the user's buffer.

BYTE_ARRAY = BYTE($*$) (Returned)
User's buffer into which bytes will be got.

ACTBYTES = INTEGER (Returned)
Actual number of values associated with the item. This may be greater than OFFSET + MAXBYTES but no more than MAXBYTES bytes will be copied into the user's buffer.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive
`NBS__TIMEOUT`	Timeout awaiting valid data

NBS_GET_MODIFIED
Get the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

Description:

Check that the ID is not NIL.
If the item is structured get the noticeboard modified count.
If the item is primitive get the item's modified count.

For structured items, this value is incremented each time an item in the noticeboard is updated.

For primitive items, if this value is even then the associated values are not currently being updated. If it is odd then they are currently being updated. The total number of updates to this item is half the value of the modified count.

An "update" is an update of an item's value, shape or size.

Note that when item data is accessed directly via pointer then the modified count is not updated unless this is done explicitly using the NBS_INC_MODIFIED routine.

Invocation:

```
(Int) = NBS_GET_MODIFIED (ID,MODIFIED,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item from which the modified count is to be got.

MODIFIED = INTEGER (Returned)
The current value of the noticeboard or item's modified count.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_MODIFIED_POINTER
Get a pointer to the noticeboard modified count or an item modified count depending on whether this is a structured or primitive item

Description:

Check that the ID is not NIL.
If the item is structured get the noticeboard modified count.
If the item is primitive get the item's modified count.
Return a pointer to the appropriate modified count.

For structured items, this value is incremented each time an item in the noticeboard is updated.

For primitive items, if this value is even then the associated values are not currently being updated. If it is odd then they are currently being updated. The total number of updates to this item is half the value of the modified count.

An "update" is an update of an item's value, shape or size.

Note that when item data is accessed directly via pointer then the modified count is not updated unless this is done explicitly using the NBS_INC_MODIFIED routine.

Invocation:

```
(Int) = NBS_GET_MODIFIED_POINTER (ID,POINTER,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for which the pointer to its modified count is to be got.

POINTER = INTEGER (Returned)
The address of the item's modified count.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_UPDATED
Determine whether a primitive item or the noticeboard has been updated since the noticeboard was found or this routine was last called

Description:

Check that the ID is not NIL.
If the item is structured get the noticeboard modified count.
If the item is primitive get the item's modified count.
Return TRUE (1) if the modified count is greater than the count the last time that this routine was called.
Return FALSE (0) otherwise.
Remember the modified count for next time.

For structured items, always use the same ID, since the remembered count is associated with the ID and not with the noticeboard.

Invocation:

```
(Int) = NBS_GET_UPDATED (ID,UPDATED,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for which to determine whether it has been updated since the last call on its behalf.

UPDATED = INTEGER (Returned)
Whether updated (1) or not (0).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_POINTER
Return a pointer to the first byte of the data of a primitive item associated with the specified identifier

Description:
Check that the ID is not NIL and that it pertains to a primitive item.
Return the address of the start of the item's noticeboard data.

Invocation:

```
(Int) = NBS_GET_POINTER (ID,POINTER,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for which the pointer to its noticeboard data is to be got.

POINTER = INTEGER (Returned)
The address of the first byte of this item's noticeboard data.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOTPRIMITIVE`	Item is not primitive

NBS_GET_NAME
Get the name of an item associated with the specified identifier

Description:

Check that the ID is not NIL.
Return the item's name.

Invocation:

```
(Int) = NBS_GET_NAME (ID,NAME,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for whose name is to be got.

NAME = CHARACTER*(*) (Returned)
The item's name.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_TYPE
Get the type of an item associated with the specified identifier

Description:

Check that the ID is not NIL.
Return the item's type.

Invocation:

```
(Int) = NBS_GET_TYPE (ID,TYPE,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for whose type is to be got.

TYPE = CHARACTER*(*) (Returned)
The item's type.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_SIZE
Get the maximum and actual sizes of a primitive item associated with the specified identifier

Description:

Check that the ID is not NIL and that it pertains to a primitive item.
Return the maximum and actual sizes of the item's noticeboard data.

Invocation:

```
(Int) = NBS_GET_SIZE (ID,MAXBYTES,ACTBYTES,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item for whose size is to be got.

MAXBYTES = INTEGER (Returned)
Maximum size in bytes.

ACTBYTES = INTEGER (Returned)
Actual size in bytes.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__NOPRIMITIVE`	Item is not primitive

Prior Requirements :

NBS_FIND_NOTICEBOARD must have been called.

NBS_GET_PRIMITIVE
Determine whether or not an item is primitive

Description:

Check that the ID is not NIL.
Return FALSE (0) if the item is a structure and TRUE (1) if it is primitive.

Invocation:

```
(Int) = NBS_GET_PRIMITIVE (ID,PRIMITIVE,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item concerned.

PRIMITIVE = INTEGER (Returned)
Whether primitive (1) or structured (0).

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_PARENT
Get the identifier of an item's parent structure

Description:

Check that the ID is not NIL.
Return the identifier of the item's parent.

If the item has no parent (ie, if it pertains to a noticeboard), then a zero ID will be returned.

Invocation:

```
(Int) = NBS_GET_PARENT (ID,PARENT,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item whose parent is to be got.

PARENT = INTEGER (Returned)
Identifier of item's parent. If the item has no parent then a NIL ID will be returned.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID

NBS_GET_CHILDREN
Get the number of children of a structured item

Description:

Check that the ID is not NIL and does not pertain to a primitive item.
Return the number of children that it has.

Invocation:

```
(Int) = NBS_GET_CHILDREN (ID,CHILDREN,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of the item whose number of children is to be got.

CHILDREN = INTEGER (Returned)
Number of children.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__PRIMITIVE`	Parent is primitive

NBS_GET_INFO
Get general non-character information on a given noticeboard

Description:

Check that the item name is legal.
Copy the current value of the item from the relevant noticeboard.

There are currently eight items which can be returned. Most are from a common noticeboard area but GLOBAL_BASE is an address within the address space of the caller. Unless otherwise stated, all are integers.

CHAN => Channel to open noticeboard file (zero if not open) DEFN_SIZE => Size of definition part of noticeboard FILE_SIZE => Size of noticeboard definition file GLOBAL_BASE => Address of noticeboard start MODIFIED => Total number of times values have been modified PID => PID of owner of this noticeboard SECTION_SIZE => Total size of noticeboard including data VERSION => Software version creating file / noticeboard

Invocation:

```
(Int) = NBS_GET_INFO (ID,NAME,VALUE,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of noticeboard or of any item in the relevant noticeboard.

NAME = CHARACTER∗(∗) (Given)
The name of the item to obtain. See the above list. Can be abbreviated so long as it remains unambiguous but this is not recommended because new items may be supported in the future. Case is not significant.

VALUE = Depends on NAME (Returned)
The item's value. Declared as pointer to integer, but may be coerced to pointer to real.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__BADOPTION`	Illegal item name

Prior Requirements :

NBS_FIND_NOTICEBOARD must have been called.

NBS_GET_CINFO
Get general character information on a given noticeboard

Description:

Check that the item name is legal.
Copy the current value of the item from the relevant noticeboard.

There is currently only one character item which can be returned.

SAVE_NAME => Name of noticeboard file (character)

Invocation:

```
(Int) = NBS_GET_CINFO (ID,NAME,VALUE,STATUS)
```

Arguments:

ID = INTEGER (Given)
Identifier of noticeboard or of any item in the relevant noticeboard.

NAME = CHARACTER*(*) (Given)
The name of the item to obtain. See the above list. Can be abbreviated so long as it remains unambiguous but this is not recommended because new items may be supported in the future. Case is not significant.

VALUE = CHARACTER*(*) (Returned)
The item's value.

STATUS = INTEGER (Given and returned)
The global status. Possible return values are,

`NBS__NILID`	NIL ID
`NBS__BADOPTION`	Illegal item name

Prior Requirements :

NBS_FIND_NOTICEBOARD must have been called.

B NBS Error Codes

This is edited from the source file that is processed by the MESSAGE utility and the (slightly modified) ADAM ERRGEN utility to generate the C (NBS_DIR:NBS_ERR.H) and Fortran (NBS_DIR:-NBS_ERR.INC) versions of the error code INCLUDE files.

Severity	Name	Description
Information	`SECTIONEXISTED`	Noticeboard already existed
Warning	`TOOMANYDIMS`	More dimensions than maximum allowed
	`TOOMANYBYTES`	More bytes than maximum allowed
	`BADOFFSET`	Offset is less than zero
	`BADOPTION`	Illegal parameter / item name
	`DATANOTSAVED`	Data part of noticeboard not saved — cannot restore it
Error	`DEFINING`	Currently defining noticeboard contents
	`NOTDEFINING`	Not currently defining noticeboard contents
	`NILSID`	`NIL` static `ID`
	`NILID`	`NIL` item `ID`
	`PRIMITIVE`	Item is primitive
	`NOTPRIMITIVE`	Item is not primitive
	`ITEMNOTFOUND`	Item does not exist
	`SECTIONNOTFOUND`	Noticeboard does not exist
	`CANTOPEN`	Can't open noticeboard definition file
	`CANTWRITE`	Can't write noticeboard definition file
	`CANTREAD`	Can't read noticeboard definition file
	`NOTOWNER`	Non-owner attempted to alter noticeboard
	`TIMEOUT`	Time out finding noticeboard or getting item value or shape
	`DATASAVED`	Data part of noticeboard saved — cannot restore definition
	`DATANOTRESTORED`	Data was not restored from noticeboard file — cannot save it
	`HASIDS`	Item / noticeboard has items derived from it — cannot lose it
	`NOTTOPLEVEL`	Item is not top-level (ie not noticeboard) — cannot lose it
	`TOPLEVEL`	Item is top-level (ie noticeboard) — cannot lose it
	`NEVERFOUND`	Parent has no items derived from it — cannot lose it
Fatal	`INITALLOCFAILED`	Couldn't initialise storage allocator
	`NOMOREROOM`	Couldn't get memory — increase `MAX_DEFN_SIZE` if when defining

C Demonstration Programs

Several demonstration programs are shipped with the system and are described here (it is assumed that the symbol "`program`" has been set up to run the program `program`). All of these programs are written in C.

C.1 `NBTRACE` — Trace Noticeboard Contents

`NBTRACE` lists the contents of a noticeboard definition file or of an active noticeboard.

The name of the definition file (assumed extension `.NBD`) or of a noticeboard item can be given as a command line parameter and will be prompted for if it is not given. The program first attempts to find the noticeboard and if that fails (because the noticeboard doesn't exist) it attempts to restore the noticeboard from the definition file. Then it locates the specified item and lists it and all items below it. VMS-style wild cards can be used. Thus

```
$ nbtrace gct.ifl.enq*
```

might result in the following

```
Noticeboard data not restored because it was not saved
STRUC    IFL (140)
  _CHAR    ENQ_DEV_DESCR   (0/132/0)
  _CHAR    ENQ_DEV_TYPE    (0/132/0)
  _CHAR    ENQ_VER_DATE    (0/132/0)
  _CHAR    ENQ_VER_NUM     (0/132/0)
```

and

```
$ nbtrace gct.fits.*.x
```

is a useful trick to suppress listing of lower level items (on the assumption that none of them are called `X`).

```
Noticeboard data not restored because it was not saved
STRUC   FITS (15)
  DRT_STRUCT   ALT_OBS (3)
  DRT_STRUCT   COMMENT (2)
  DRT_STRUCT   HAEND (3)
  DRT_STRUCT   HASTART (3)
  DRT_STRUCT   INSTRUME (3)
  DRT_STRUCT   LAT_OBS (3)
  DRT_STRUCT   LONG_OBS (3)
  DRT_STRUCT   OBJECT (3)
  DRT_STRUCT   ORIGIN (3)
  DRT_STRUCT   RUN (3)
  DRT_STRUCT   SPEED (3)
  DRT_STRUCT   UTDATE (3)
  DRT_STRUCT   WINDOW (3)
  DRT_STRUCT   ZDEND (3)
  DRT_STRUCT   ZDSTART (3)
```

When no item name is given, general information about the noticeboard's size and owner is given, as in

```
$ nbtrace words
Software version   = 5 (5)
Size of section    = 1080 (438)
Size of definition = 1080 (438)
Noticeboard owner  = 2959 (b8f)
Modified count     = 0 (0)

NOTICEBOARD WORDS (9)
  WORD     BROWN    (0/0/0)
  WORD     DOG      (0/0/0)
  WORD     FOX      (0/0/0)
  WORD     JUMPS    (0/0/0)
  WORD     LAZY     (0/0/0)
  WORD     OVER     (0/0/0)
  WORD     QUICK    (0/0/0)
  WORD     THE      (0/0/0)
  WORD     THE      (0/0/0)
```

C.2 TIME — Time Noticeboard Operations

TIME times various noticeboard operations. It produced the timings listed in Section 6.

The user chooses how many items the noticeboard should contain, which item should be used for timing NBS_FIND_ITEM, how many iterations to perform and the values of the INCREMENT_MODIFY and CHECK_MODIFY flags. The program produces a report in TIME.LIS. For example:

```
Noticeboard system timing program at Thu Mar 31 17:49:00 1988
-------------------------------------------------------------

Number of iterations   = 10000
Number of items        = 100
Item used for searches = 100
Increment modify flag  = 1
Check modify flag      = 1

Test 0, define, save and find noticeboard
       cpu microseconds = 2330000

Test 1, scalar assignment
       cpu microseconds per iteration = 3

Test 2, put scalar
       cpu microseconds per iteration = 124

Test 3, get scalar
       cpu microseconds per iteration = 90
.
.
```

C.3 EXERCISE — Exercise Noticeboard Routines

EXERCISE calls all NBS routines and triggers all reasonable errors. It produces a report in EXERCISE.LIS in which any unexpected results have an asterisk in the first column. The total error count is reported at the bottom and this should always be zero. If it is non-zero there could be a resource-related problem (or even a bug in the NBS routines). For example:

```
Noticeboard system exercise program at Thu Mar 31 10:28:55 1988
---------------------------------------------------------------

NBS_TUNE
--------
nbs_tune max_defn_size: ok
nbs_tune of doesnt_exist: illegal parameter / item name

NBS_DEFINE_* errors
-------------------
nbs_define_structure: not currently defining noticeboard contents
.
.

NBS_PUT
-------
nbs_get_primitive: NIL item ID
nbs_get_parent: NIL item ID
nbs_get_children: NIL item ID
nbs_get_children: item is primitive
nbs_get_info: NIL item ID
nbs_get_info: illegal parameter / item name

Error count = 0
----------------
```

C.4 WORDS — Generate Tree-structured Noticeboard

WORDS generates a tree-structured noticeboard with one item for each word found in a file provided by the user. The user also specifies a “cluster size”, which is a measure of how deep the tree is (the tree has about $\log_2$(cluster size) levels). Once the noticeboard has been created the user can type in words and is told whether they appeared in the file. Exit with ^Z. The example given for NBTRACE is of a noticeboard created using WORDS.