Natural Buffer Pool - General

The buffer pool is a storage area into which Natural programs are placed in preparation for their execution. Programs are moved into and out of the buffer pool as Natural users request Natural objects. Conceptually, it serves a function similar to that of an operating system in loading programs in and out of a reentrant area. The Natural buffer pool is an integral part of Natural in all supported environments.

The following topics are covered:

- Natural Buffer Pool Principle of Operation
- Buffer-Pool Monitoring and Maintenance
- Natural Global Buffer Pool

Natural Buffer Pool Principle of Operation

Natural generates reentrant Natural object code. A compiled program is loaded into the buffer pool and executed from the buffer pool. Thus, it is possible that a single copy of a Natural program can be executed by more than one user at the same time.

This section covers the following topics:

- Objects in the Buffer Pool
- Directory Entries
- Text Pool
- Buffer Pool Hash Table
- Buffer Pool Initialization
- Buffer Pool Search Methods
- Local and Global Buffer Pools
- Buffer Pool Cache

Objects in the Buffer Pool

Objects in the buffer pool can be programs, subprograms, maps and global data areas. Global data areas are placed in the buffer pool only for compilation. In this case, two objects with the same name are loaded in the buffer pool: the GDA itself and the corresponding symbol table.

Directory Entries

When a Natural object is loaded into the buffer pool, a control block called a directory entry is allocated to this object.

A directory entry contains such information as the name of the object, what library it belongs to, what database ID and Natural system file number the object was retrieved from, and some statistical information (for example, the number of users who are concurrently executing the program at a given point in time).

When a user executes a program, Natural checks the directory entries to see if the program has already been loaded into the buffer pool. If it is not already in the buffer pool, a copy of the program is retrieved from the appropriate Natural system file and loaded into the buffer pool.

When an object is loaded in the buffer pool, one or more other Natural objects which are currently not being executed may be deleted from the buffer pool in order to make room for the newly loaded object. When the new object is loaded, a new directory entry is created in order to identify this object.

When an object is deleted from the system file, it will also be deleted from the buffer pool as soon as it is no longer being used. When an object is newly cataloged or stowed, its old version will also be deleted from the buffer pool as soon as it is no longer being used; when it is requested for execution again, the new version will then be loaded from the system file into the buffer pool.

Text Pool

The actual object code of a program that is loaded into the buffer pool is placed into an area called the text pool and must be allocated as a contiguous piece of memory within this text pool. This text pool is divided into a number of 4 KB buffers. This is an arbitrary size and can be changed at the Natural administrator’s discretion. When an object is loaded, one or more text buffers that are contiguous to each other are allocated to store the object code of the object.

Buffer Pool Hash Table

This section applies to buffer pools of `TYPE=NAT` only.

To speed up the search time for looking up an object in the buffer pool directory, a hash table is used. The number of entries in the hash table is twice the number of directory entries, rounded up to the next prime number. This will ensure that only half of the table is filled at any point in time and that the probability of collisions is near zero. As a consequence, the average number of tests to find an existing object in the hash table is theoretically less than 2.

The hash criterion is the eight character long program name. If more than one program name is hashed to the same location in the hash table, an overflow technique resolves the collisions.

The storage required for the hash table is roughly 16 bytes per text block. Thus, the available storage in the text pool is reduced by between 1.6% (1 KB text blocks) and 0.1% (16 KB text blocks).

Buffer Pool Initialization

In case of a global buffer pool the initialization occurs during start of the global buffer pool.

In case of a local buffer pool the initialization time varies depending on the environment.

- In batch mode, TSO, TIAM and VM/CMS, the initialization occurs when you begin the execution of the Natural session.

- In a TP monitor environment, the initialization generally occurs when the first user invokes Natural under this TP monitor. Under Com-plete and CICS, it is also possible to initialize the local buffer pool when the TP monitor is started.

Buffer Pool Search Methods

As mentioned earlier and explained below, there are different search methods for allocating space in the buffer pool.

To select a search method, use

- The Natural profile parameters `BPMETH` and `BPI`.
  Or the macro `NTBPI` in the Natural parameter module.
  Or the function parameter `METHOD` of the global buffer pool.

For a description of these parameters and the macro `NTBPI` refer to the Natural *Parameter Reference* documentation.

Below is information on the search methods:

- `METHOD=S`
- `METHOD=N`
- Choosing Search Methods

METHOD=S

`METHOD=S` indicates that a selection process is used as search algorithm for allocating storage in the buffer pool in order to obtain the space required to accomplish a new load.

The selection process used is a combination of search Algorithms 1 and Algorithm 2:

- **Algorithm 1**
  Search Algorithm 1 attempts to find storage in the buffer pool that is either free space or space occupied by an unused (active but not used) object.

  If free space of the exact object size required is found, the selection process ends immediately. Otherwise, the search continues by browsing the buffer pool from top to bottom and comparing the entries in the buffer pool for best size fit. Additionally, in the case of unused objects, the search also considers the last attached time of the object, that is, the time the object was last referenced at a load or locate.

  When the selection process has finished, either free space or the space of an unused object with a size greater than or equal to the size requested is selected. The rule of precedence that applies to the search is: free space is taken first and space of unused objects next. In the case of unused objects, the oldest objects are removed first.

  If the selection process of Algorithm 1 was not successful, Algorithm 2 is invoked.

- **Algorithm 2**
  Search Algorithm 2 starts if Algorithm 1 fails. Algorithm 2 starts searching from a position in the buffer pool which is passed by Algorithm 1 and attempts to combine two or more entities (free storage and/or space occupied by unused objects) in order to obtain the necessary storage for a new load. However, the age of the object is not taken into account.

  Algorithm 2 continues processing to the bottom of the text record section and, if necessary, wraps around to the top of the text record section to make one final pass from top to bottom. If space is still unavailable, Algorithm 2 fails, the object cannot be loaded and Natural issues a corresponding error message.

METHOD=N

`METHOD=N` indicates that the next available free or unused space is used in order to obtain the space required to accomplish a new load. Unused space is space that is occupied by an active but not used object.

The search for the next available space starts from a pointer that moves through the buffer pool in a wrap-around fashion. Any next available buffer pool entries that are free or contain unused objects can be used and possibly chained together to obtain the amount of storage requested.

If the bottom of the buffer pool is reached during an allocation request, the pointer is wrapped around to the top of the buffer pool and one final search is performed through the buffer pool from top to bottom. If the bottom of the buffer pool is reached again and the object cannot be loaded, the load fails and Natural issues a corresponding error message.

`METHOD=N` can especially be considered for large buffer pools in combination with the buffer pool cache function. For details, see also Choosing Search Methods below.

Choosing Search Methods

By default, `METHOD=S` is used. The advantage of this method is, that a diligent search is performed to allocate space, taking into account the size and the age of objects and guaranteeing that the most dispensable unused objects are removed from the buffer pool to provide space for a new load.

A disadvantage of `METHOD=S` can be the high CPU time that is consumed by the selection process when browsing the buffer pool from top to bottom.

The advantage of `METHOD=N` is the short selection process and, usually, little browsing that require less CPU time for allocating space. This can be significant to large buffer pools.

The disadvantage of `METHOD=N` is that objects are selected less carefully for removal from the buffer pool. To avoid an increase in Adabas I/Os for reloading removed objects, we recommend that you use `METHOD=N` in combination with the buffer pool cache function.

Local and Global Buffer Pools

Local Buffer Pool

Using Natural as supplied on the installation tape, you are running a *local* buffer pool. This is a buffer pool area that is allocated in the same partition (or region or address space) of the particular environment in use.

For example, in a batch, TSO or CMS environment, each user has his/her own local buffer pool. In a TP monitor environment such as Com-plete, CICS or IMS TM, there is one buffer pool per TP monitor from which all TP users execute.

Global Buffer Pool

In a z/OS environment, a global buffer pool is allocated from CSA or ECSA storage. In such an environment, all TSO users, batch users and TP monitor users could be executing from one common global area.

In a z/VSE environment, a global buffer pool is allocated from System GETVIS Area (below or above). In such an environment, all batch users and TP monitor users could be executing from one common global area.

In a VM/CMS environment, a global buffer pool can be installed as a writeable Discontiguous Shared Segment that is dynamically attached to the user's virtual machine when Natural/CMS is invoked; see also the sections *Installing Natural under CMS* and *Preparing the VM System for Natural* in the Natural *Installation* documentation.

In a BS2000/OSD environment, a global buffer pool is a common memory pool, see *Natural Global Buffer Pool under BS2000/OSD*.

For further information on the global buffer pool, see *Natural Global Buffer Pool*.

Buffer Pool Cache

This section applies to global buffer pools of `TYPE=NAT` and local buffer pools of `TYPE=NAT` or `TYPE=SWAP`.

The buffer pool cache is available in conjunction with global and local buffer pools. It is not available with z/VM. It is used only for Natural programming objects (programs, subprograms, maps, etc.), whereas it is not used for example for objects generated by Natural for DL/I.

When a buffer pool is not large enough to take up all objects requested by the different users, special overload strategies are used to replace existing objects with requested objects. The number of overload situations has a direct relation to the efficiency of the buffer pool. The best and most efficient way to reduce the disliked overloads, hence to improve the buffer pool performance, is simply to increase its size.

However, this choice is not applicable at most customer sites, due to a lack of available storage in the primary address space and/or the z/OS (E)CSA, z/VSE system GETVIS area or BS2000/OSD Common Memory Pool.

In order to improve the situation described above, a buffer pool cache is used. The main purpose of this mechanism is to prevent a loss of all objects which were deleted from the buffer pool due to "short-on-buffer-pool-storage" situations. This means, that an object delete results in a "swap out to buffer pool cache". The intended benefit of this feature is a reduction of database calls used for object loading and consequently a performance improvement.

Note for Global Buffer Pools:
The buffer pool cache area is allocated in a data space. When a data space is created for a buffer pool (profile parameter `BPCSIZE` specified for z/OS or z/VSE, or `DATA` parameter specified for BS2000/OSD), the ownership is assigned to the creator task. If this task terminates, the system automatically deletes the data space. Therefore, the creator task will stay alive in this case, regardless of

whether RESIDENT=Y has been set or not.

Note for Local Buffer Pools: (z/OS and z/VSE only, not for Com-plete and not for IMS TM)
The buffer pool cache is allocated in a data space or in a memory object "above the bar", that is, in 64-bit memory (z/OS only). When a data space or memory object is created for a buffer pool (see profile parameters BPCSIZE and BPC64), the ownership is assigned to the creator task. If this task terminates, the system automatically deletes the data space or the memory object.

Buffer-Pool Monitoring and Maintenance

The Natural utility SYSBPM (described in the Natural *Utilities* documentation) provides statistical information on the current status of the buffer pool. SYSBPM also allows you to adjust the buffer pool to your requirements.

The following topics are covered below:

- Preload List
- Blacklist
- Propagation of Buffer-Pool Changes
- Performance Considerations

Preload List

A preload list is a list of objects that will be loaded into the buffer pool and remain there as resident. When a user requests such an object for execution, it is always already in the buffer pool and need not be loaded from the system file.

This may improve performance, may avoid buffer pool fragmentation, or may be useful to ensure that central error transactions are always available, even if the database containing the system file is not active.

At the start of the Natural session, Natural checks which of the objects on the preload list are already in the buffer pool. Those which are not will then be loaded from the system file into the buffer pool. This checking and loading is also performed whenever the buffer pool is connected, re-connected and re-initialized using the SYSBPM utility. If a global buffer pool is re-initialized by a REFRESH command, no checking takes place for existing Natural sessions. That is, as long as no new Natural session is started that accesses this buffer pool, the objects on the preload list are not loaded.

The load of the preload list is not serialized. That means, if multiple Natural sessions start concurrently, each session tries to load all objects named in the preload list into the buffer pool. This may lead to duplicate entries of the same Natural object in the buffer pool (see also hint below).

A preload list is identified by name, and is attached to a specific buffer pool by specifying its name as startup parameter (for a global buffer pool) or in the NTBPI macro (for a local buffer pool). Thus, a different preload list may be defined for each buffer pool; or the same preload list may be used for different buffer pools.

If the specified preload list cannot be located, or if objects contained on the preload list cannot be loaded, Natural will issue a corresponding warning message at session initialization. In either case, the preloading will be repeated for each subsequent session initialization.

As the objects on the preload list are the first to be loaded, they are located at the beginning of the buffer pool (except if the initial preloading could not load all objects, in which case the objects may be located anywhere in the buffer pool).

To maintain preload lists, you use the SYSBPM utility, see *SYSBPM - Preload List Maintenance* in the Natural *Utilities* documentation.

Tip:
To avoid problems with the load of the objects on a preload list by user sessions the following procedure is recommended:

- **For a global buffer pool:**
  Immediately after the allocation or refresh of the global buffer pool, start a batch Natural session that accesses the global buffer pool and that executes a FIN.
- **For a local buffer pool under CICS and Com-plete:**
  During startup of the TP system, start an asynchroneous Natural session that access the local buffer pool, and put a FIN command on the Natural stack. Ensure that this Natural session references the name of the preload list in its NTBPI macro.

Blacklist

To prevent a Natural object from being executed, you can put it on a so-called "blacklist": the object can then not be loaded into the buffer pool; and if it is already in the buffer pool, it cannot be executed. A user requesting such an object to be executed will then receive an appropriate error message.

You can put not only individual objects on the blacklist, but also entire libraries.

Examples:

- The blacklist may be useful, when you upgrade a Natural application and do not wish users to continue to work with that application until you have finished the upgrade. Without the blacklist, a user might execute a new module which in turn would invoke an old module - which might lead to an abnormal termination of the Natural session. With the blacklist, the user can will receive a message that the requested object can currently not be executed, and can then continue his/her Natural session.
- Performance aspects may be another reason for using the blacklist to prevent certain resource-consuming objects from being executed in a specific environment.
- You may use the blacklist to prevent the execution of test programs in a production environment.

To maintain the blacklist, you use the SYSBPM utility. See *SYSBPM - Blacklist Maintenance* in the Natural *Utilities* documentation.

Propagation of Buffer-Pool Changes

Note:
Under z/OS, the propagation of buffer-pool changes is always restricted to the Natural subsystem in which the change has occurred (for details on the Natural subsystem, see *Natural Subsystem (z/OS)* or *Natural Subsystem (z/VSE)*. Thus, "all global buffer pools" in this context means "all global buffer pools within the same subsystem".

In some environments, it is important that changes which occur in one (local or global) buffer pool are also propagated to all other global buffer pools - that is, the same changes are also automatically made in the other global buffer pool - so as to ensure the consistency of the buffer pools and the Natural applications being used. This is particularly important in a z/OS Parallel Sysplex environment.

For example, if a Natural program is newly cataloged in one z/OS image, the propagation will cause the program to be deleted from all other global buffer pools in the z/OS Parallel Sysplex environment, so that its new version has to be loaded from the system file when the program is to be executed again.

The following changes are propagated:

- an object is deleted from the buffer pool,
- the buffer pool's blacklist is modified,
- the buffer pool is re-initialized.

Changes can be propagated to all other global buffer pools within the current z/OS image, or within the entire z/OS Parallel Sysplex environment, or all other global buffer pools of the same name within the z/OS Parallel Sysplex environment.

The propagation does not affect those objects in another global buffer pool which are defined as resident in that buffer pool.

The propagation is activated and its scope controlled by the Natural profile parameter BPPROP.

Note:
As the propagation is performed asynchronously and an object is only deleted from a buffer pool when it is not longer being used, it may take some time until the current version of an object is available in all buffer pools.

Propagation to other *local* buffer pools is not possible.

Performance Considerations

For general advice on performance-related issues regarding the buffer pool and the BP cache, see *Performance Considerations* in the section SYSBPM of the Natural *Utilities* documentation.

Natural Global Buffer Pool

The Natural global buffer pool is an optional Natural component.

It is available for the following operating systems

- z/OS (refer to *Global Buffer Pool under z/OS*)
- z/VSE (refer to *Global Buffer Pool under z/VSE*)
- BS2000/OSD (refer to *Global Buffer Pool under BS2000/OSD*).

Profile Parameters Used

The following Natural profile parameters are used to establish a global buffer pool:

BPNAME	Specifies the name of the global buffer pool (see BPNAME). BPNAME=' ' (blank) is used to establish a connection to the *local* buffer pool.
SUBSID	Specifies the ID of the Natural subsystem to be used (see profile parameter SUBSID; applies only under z/OS and z/VSE).

During Natural startup, Natural attempts to locate the global buffer pool using these parameters.

Buffer Pool Opening / Closing Procedure

With the NTBPI macro of the Natural parameter module or the corresponding profile parameter BPI, you can define more than one buffer pool.

At session initialization, Natural attempts to establish a connection to the first buffer pool defined. If this fails, Natural attempts to establish a connection to the second buffer pool defined. If that fails, too, it tries the next buffer pool defined, etc. Whenever an attempt to establish a connection to a buffer pool fails, Natural will issue a corresponding message.

The same procedure applies when a buffer pool is stopped: if you close the currently connected buffer pool while a Natural session is still active, Natural attempts to connect to another buffer pool (in the order in which they are defined) and continue the session. Thus, it is possible for the Natural administrator to close a global buffer pool without having to terminate all active Natural sessions.