**Portable Systems Group**

# Windows NT Argument Validation Specification

**Author:** *David N. Cutler*

*Original Draft, May 4, 1989*
*Revision 1.1, May 5, 1989*
*Revision 1.2, May 10, 1989*
*Revision 1.3, July 15, 1989*

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

## 1. Overview

This document describes the argument probing and capture requirements to which all system services must adhere.

System services must be written such that they are robust and provide protection against malicious attack and inadvertent program bugs. It must not be possible to crash or corrupt the system by passing an invalid argument value, a pointer to memory that is not accessible to the caller, or by dynamically altering or deleting the memory occupied by an argument in a simultaneously executing thread.

## 2. Requirements

Every system service must ensure that the arguments on which it operates are valid (i.e., values are correct). This is essential to robust system operation and involves the capturing of values and the probing of argument addresses at appropriate points.

In general, a system service should capture all arguments on entry to the procedure. This ensures that the caller or one of its cohorts (buddy threads) cannot dynamically alter the value of the argument after it has been read and verified, or delete the memory in which it is contained.

In some cases, it is not necessary to capture the value of an argument immediately. Such is the case for I/O buffers and name strings. However, all pointers **MUST** be captured and the addresses to which they point **MUST** be probed for accessibility.

Fortunately, most arguments do not need explicit capture since they are passed in registers. Arguments that are passed in memory are probed and captured by the system service dispatcher as necessary.

## 3. Operation

The address space layout of **Windows NT** contains a boundary that delineates user address space from system address space. All addresses above the boundary are considered system addresses and all addresses below the boundary are considered user addresses.

Pages in the system part of the address space are owned by kernel mode and are not accessible to the user unless they are double mapped into the user part of the address space. Pages in the user part of the address space are owned by user mode and the access for kernel mode is identical to that for user mode.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

The executive **NEVER** creates a page in the user part of the address space that is owned by kernel mode. Furthermore, at the boundary between user address space and system address space, there are **64K** bytes that are inaccessible to all modes. This address space layout makes it possible to determine whether an address is a valid user address simply by doing a boundary comparison.

When a system service is called, the trap handler gets control, saves state, and transfers control to the system service dispatcher. The system service dispatcher determines which system service is being called, and obtains the address of the appropriate function and the number of in-memory arguments from a dispatch table. If the previous processor mode is user mode and there is one or more in-memory arguments, then the in-memory argument list is probed and then copied to the kernel stack. If an access violation occurs during the copy, then the system service is completed with a status of access violation. If an access violation does not occur, then the the pointer to the in-memory argument list is changed to point to the copy of the arguments on the kernel stack. The system service dispatcher sets up a catchall condition handler, and then calls the system service function.

The first thing the system service should do is establish a condition handler. This handler should be prepared to handle access violations that may occur as argument pointers are dereferenced to read or write actual argument values.

Next, the system service code should obtain the previous processor mode. If the previous processor mode was kernel, then there is no need to probe any arguments. The executive does not call itself with bad arguments.

If the previous processor mode was user, then any argument values that are read or written by dereferencing a pointer must be probed for accessibility. Probing is accomplished by first ensuring that the address of the variable is within the user's address space and then reading or writing the variable as appropriate. The code that actually probes pointer-related arguments does not set up a condition handler. It merely does the boundary check and then reads or writes the indicated location. If the boundary check fails, an access violation condition is raised. If the memory is inaccessible, an access violation is raised by hardware. Thus probes are extremely cheap.

The complete code at the beginning of a system service should be constructed as follows:

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
// set up condition handler to catch access violations
            .
            .
            .

if (GetPreviousMode() != KernelMode) {
            .
            .
            .
      // probe and capture reference arguments
            .
            .
            .
}
```

At this point in the execution of a system service, all input values have been captured and all output variables have been probed for writeability. The system service performs its function, writes output values as necessary, and returns a status that indicates whether the service succeeded or failed.

During the writing of output values, an access violation can occur because another thread or user altered the address space of the calling thread. Access violations that occur at this time are silent and do not cause the service to fail. If this were not the case, then it would be very difficult to actually complete a system service since code would have to be added to back out and undo the service right up until the very last output value is written. If the caller receives a success status under such conditions, it is likely that the caller will attempt to access one of the output values and get an access violation.

## 4. Interfaces

The following sections describe the interfaces that are provided to probe arguments for read and write accessibility.

### 4.1 Probe for Readability and Read Argument Value

The following functions provide the capability to probe a primitive data type for readability and to read an argument value.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
CHAR
ProbeAndReadChar (
    IN PCHAR Address
    );

UCHAR
ProbeAndReadUchar (
    IN PUCHAR Address
    );

SHORT
ProbeAndReadShort (
    IN PSHORT Address
    );

USHORT
ProbeAndReadUshort (
    IN PUSHORT Address
    );

LONG
ProbeAndReadLong (
    IN PLONG Address
    );

ULONG
ProbeAndReadUlong (
    IN PULONG Address
    );

QUAD
ProbeAndReadQuad (
    IN PQUAD Address
    );

UQUAD
ProbeAndReadUquad (
    IN PUQUAD Address
    );
```

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
HANDLE
ProbeAndReadHandle (
    IN PHANDLE Address
    );

BOOLEAN
ProbeAndReadBoolean (
    IN PBOOLEAN Address
    );
```

The previous functions are used to probe and read a value pointed to by a safe pointer. A safe pointer is one that has either been captured on procedure entry or which has been previously captured with one of the these functions. The functions compare the pointer value to the user/system address boundary, read the appropriate data-type value, and return the value as the function value. If the value is not of consequence, then the function value is simply not assigned to a variable. Note that both signed and unsigned data types are provided.

### 4.2 Probe for Writeability and Read Argument Value

The following functions provide the capability to probe a primitive data type for writeability and read an argument value.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
CHAR
ProbeForWriteChar (
    IN PCHAR Address
    );

UCHAR
ProbeForWriteUchar (
    IN PUCHAR Address
    );

SHORT
ProbeForWriteShort (
    IN PSHORT Address
    );

USHORT
ProbeForWriteUshort (
    IN PUSHORT Address
    );

LONG
ProbeForWriteLong (
    IN PLONG Address
    );

ULONG
ProbeForWriteUlong (
    IN PULONG Address
    );
```

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
QUAD
ProbeForWriteQuad (
    IN PQUAD Address
    );

UQUAD
ProbeForWriteUquad (
    IN PUQUAD Address
    );

HANDLE
ProbeForWriteHandle (
    IN PHANDLE Address
    );

BOOLEAN
ProbeForWriteBoolean (
    IN PBOOLEAN Address
    );
```

The previous functions are used to probe for writeability and read a value pointed to by a safe pointer. A safe pointer is one that has either been captured on procedure entry or which has been previously captured with one of these functions. The functions compare the pointer value to the user/system address boundary, read the appropriate data type value, write the value that was read back into memory, and return the original value as the function value. If the value is not of consequence, then the function value is simply not assigned to a variable. Note that both signed and unsigned data types are provided.

### 4.3 Probe for Writeability and Read/Write Argument Value

The following functions provide the capability to probe a primitive data type for writeability, read an argument value, and write a specified value.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
CHAR
ProbeAndWriteChar (
      IN PCHAR Address,
      IN CHAR Value
      );

UCHAR
ProbeAndWriteUchar (
      IN PUCHAR Address,
      IN UCHAR Value
      );

SHORT
ProbeAndWriteShort (
      IN PSHORT Address,
      IN SHORT Value
      );
```

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

```
USHORT
ProbeAndWriteUshort (
        IN PUSHORT Address,
        IN USHORT Value
        );

LONG
ProbeAndWriteLong (
        IN PLONG Address,
        IN LONG Value
        );

ULONG
ProbeAndWriteUlong (
        IN PULONG Address,
        IN ULONG Value
        );

QUAD
ProbeAndWriteQuad (
        IN PQUAD Address,
        IN QUAD Value
        );

UQUAD
ProbeAndWriteUquad (
        IN PUQUAD Address,
        IN UQUAD Value
        );

HANDLE
ProbeAndWriteHandle (
        IN PHANDLE Address,
        IN HANDLE Value
        );

BOOLEAN
ProbeAndWriteBoolean (
        IN PBOOLEAN Address,
        IN BOOLEAN Value
        );
```

The previous functions are used to probe a primitive data type for writeability and read a value pointed to by a safe pointer. In addition, the value that is to be written is specified as an argument to the function. A

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

safe pointer is one that has either been captured on procedure entry or which has been previously captured with one of these functions. The functions compare the pointer value to the user/system address boundary, read the appropriate data-type value, write the specified value to memory, and return the original memory contents as the function value. If the value is not of consequence, then the function value is simply not assigned to a variable. Note that both signed and unsigned data types are provided.

### 4.4 Probing An Aggregate Value

The following functions provide the capability to probe aggregate data types (i.e., structures, arrays, strings, etc.) for read and write accessibility.

```
VOID
ProbeForRead (
    IN PCHAR Address,
    IN ULONG Length
    );

VOID
ProbeForWrite (
    IN PCHAR Address,
    IN ULONG Length
    );
```

The previous functions are used to probe an aggregate for read or write accessibility using a safe pointer. A safe pointer is one that has either been captured on procedure entry or which has been previously captured with one of the preceding functions. The functions compare the starting and ending addresses of the specified aggregate for read or write accessibility and then read or write one character from each page that is spanned by the aggregate. Note that these functions do not capture the aggregate value.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License

**Revision History:**

Original Draft 1.0, May 4, 1989

Revision 1.1, May 5, 1989

1. Add capturing of reference arguments to sample system service code.

2. Change data type definitions to make Portable System Group conventions.

Revision 1.2, May 10, 1989

1. Move the capturing and probing of the in-memory argument list into the system service dispatcher.

Revision 1.3, July 15, 1989

1. Add functions to probe handle and boolean values.

Copyright (c) Microsoft Corporation - Use subject to the Windows Research Kernel License