Registry Callbacks

© Copyright Microsoft Corporation, 2001. All Rights Reserved 1

1. Introduction 3

2. Implementation Details 4

2.1 Callback prototype 4

2.2 Invocation of callbacks 4

2.3 Callback Registration and Deregistration 5

2.4 Writing a callback routine 6

3. Compatibility Notes 7

4. Appendix 8

1. Introduction

This feature adds callback support to the Registry. Callback support is needed by virus filters and other components that need to control access to the machine's registry. In Windows XP, the system service table is marked read-only as part of a general effort to keep the system from being destabilized by errors in drivers. It was then discovered that a number of kernel-mode components from third parties had been patching the system service table in order to monitor and control application invocation of system services. The registry APIs were one of the key areas where third parties needed to intercept calls, so this new facility is intended to provide an alternative to patching the system service table.

Registry callbacks provide a way for kernel-mode components to get control whenever an application attempts to access the Registry. The callbacks provide a way to monitor registry access as well as fail calls. Registry callbacks are a kernel-only mechanism.

The remainder of this document describes the logic by which the Registry calls the registered callbacks, registration/deregistration routines and data structures used to pass call parameters between the registry and the callbacks.

2. Implementation Details

2.1 Callback prototype

Due to the large number of registry APIs, the registry uses a single callback for all registry APIs rather than a distinct callback for each API. The callback is based on the general executive callback mechanism. The prototype the registry filters should implement for the callback procedure is defined in ntddk.h as:

```
typedef NTSTATUS (*PEX_CALLBACK_FUNCTION ) (
   IN PVOID CallbackContext,
   IN PVOID Argument1,
   IN PVOID Argument2
   );
```

- *CallbackContext* is for the callback's internal use. It is passed in at registration time and then passed to the callback procedure every time the callback is invoked.
- *Argument1* is a type selector and specifies the current registry call for which the callback is invoked. It is of type REG_NOTIFY_CLASS (see below).
- *Argument2* is a pointer to a structure where the actual parameters are packed together. It is of one of the PREG_*_INFORMATION types (see below).

The difference between the parameters passed to a registry API and the parameters to the corresponding callback procedure is that instead of a handle, the callback procedure is passed the object pointer. Handles are not safe to use in the callbacks because they can be closed and reused asynchronously with the callbacks. Object pointers avoid this problem because the object manager takes a reference on the object before passing the pointer to the callback, and releases the reference after the callback returns. The callback should not itself maintain a reference on the object.

2.2 Invocation of callbacks

Callbacks will be invoked before any internal registry work is done, providing the callback the opportunity to fail the API before the registry operation takes place..

The registry will probe the API parameters before passing them to the callback. However, the registry does not capture them, so the callback implementers should be aware that data buffers could be un-trusted user mode addresses and access them only inside a try/except block.

The registry does not hold any locks within the current thread while invoking the callbacks. This is to reduce the possibility of deadlock or vulnerability to denial of service attacks.

For each registry operation, callback routine will be invoked twice. First before any operation is performed (the ‘pre’ callback notification) and second after the operation is performed (the ‘post’ callback notification). Operation types are prefixed with ‘Pre’ and ‘Post’ accordingly (see appendix - `REG_POST_OPERATION_INFORMATION`).

Below is the pseudo code for Registry invocation of ‘pre’ callbacks:

```
If (  AreCallbacksRegistered() ) {
        for (all registered callbacks) {
                status = callback(...);
                if ( !NT_SUCCESS(status ) ) {
                        return status to the registry API caller without doing
                        any registry work
                }
        }
}
```

For ‘post’ notification, the operation type, status of the operation and the object upon which it was performed will be fed to the callback. Callback return code will be ignored. Callback writers can match pre and post calls by recording the thread ID in the pre notification and match against it in the post notification.

Callback procedures may use any registry API. However, callbacks will not be invoked for registry APIs called from inside a callback.

Note: For NtCreateKey and NtOpenKey post callbacks the Object field is NULL on error. Callback writers should first check the status code to make sure it is safe to dereference the Object.

2.3 Callback Registration and Deregistration

Callback register and un-register APIs are declared in ntddk.h. To register a callback, callback writers would call:

```
NTSTATUS
CmRegisterCallback(IN PEX_CALLBACK_FUNCTION Function,
                   IN PVOID              Context,
                   IN OUT PLARGE_INTEGER    Cookie
                  );
```

Function is the actual callback routine, handling the registry APIs of interest.

Context is an opaque value to be passed back to the callback every time the callback is involved. Its interpretation is up to the callback routine.

Cookie is used to un-register the callback.

To un-register a callback:

```
NTSTATUS
CmUnRegisterCallback(IN LARGE_INTEGER    Cookie);
```

Cookie should be the same as returned by the register API.

2.4 Writing a callback routine

A callback routine should immediately return STATUS_SUCCESS for any Registry operation that is not of interest. After considering the parameters to operations that it wants to filter, a callback routine should return an error code rather than STATUS_SUCCESS for any operation it intends to fail.

When an error status is returned the corresponding registry API call is immediately failed without any changes being made to the registry. (Except for the status code for post-create operations, which is ignored: the create/open registry APIs can only be failed in the pre-create callback).

Here is how a simple callback routine should look (see Appendix or ntddk.h for type definitions):

```
NTSTATUS RegistryCallback(
    IN PVOID CallbackContext,
    IN PVOID Argument1,
    IN PVOID Argument2
    )
{
    REG_NOTIFY_CLASS Type;
    Type = (REG_NOTIFY_CLASS)Argument1;
    switch( Type ) {
    case RegNtPreCreateKeyEx:
        {
            REG_ CREATE_KEY_INFORMATION pCreate =
                                    (REG_ CREATE_KEY_INFORMATION)Argument2;
            //
            // Code to handle Pre - CreateKey
            //
        }
        break;
    case RegNtPreDeleteKey:
        {
            PREG_DELETE_KEY_INFORMATION  pDelete
                                                    = (PREG_DELETE_KEY_INFORMATION)Argument2;
            //
            // Code to handle NtDeleteKey
            //
        }
        break;
    case RegNtPreSetValueKey:
        {
            PREG_SET_VALUE_KEY_INFORMATION pSetValue =
(PREG_SET_VALUE_KEY_INFORMATION)Argument2;
            //
            // Code to handle NtSetValueKey
            //
        }
        break;
    case RegNtPreDeleteValueKey:
        {
            PREG_DELETE_VALUE_KEY_INFORMATION  pDeteteValue
                                                  =  (PREG_DELETE_VALUE_KEY_INFORMATION)Argument2;
            //
            // Code to handle NtDeleteValueKey
            //
        }
        break;
    default:
        //
        // we are not interested to hook these APIs. Let the call pass through and return STATUS_SUCCESS
        //
        break;
    }

    return STATUS_SUCCESS;
}
```

3. Compatibility Notes

This documents refers to .Net server. Differences in prior versions are outlined below:

- Windows XP gold and service packs send only pre notifications with the exception of NtCreateKey and NtOpenKey where both pre and post notifications are sent. The following pairs are used:
    - (RegNtPreCreateKey, REG_PRE_CREATE_KEY_INFORMATION)
    - (RegNtPreOpenKey,REG_PRE_OPEN_KEY_INFORMATION)
    - (RegNtPostCreateKey, REG_POST_CREATE_KEY_INFORMATION)
    - (RegNtPostOpenKey,REG_POST_OPEN_KEY_INFORMATION)
- Windows .Net server doesn't send the above pre/post notifications in the create/open case (for backward compatibility reasons). Instead it send the new **Ex** versions, as below:
    - (RegNtPreCreateKeyEx, REG_CREATE_KEY_INFORMATION)
    - (RegNtPreOpenKeyEx, REG_OPEN_KEY_INFORMATION)
    - (RegNtPostCreateKeyEx, REG_POST_OPERATION_INFORMATION)
    - (RegNtPostOpenKeyEx, REG_POST_OPERATION_INFORMATION)
- In the post create/open notifications Windows XP sets the Object filed to the address of the pointer to the object instead of the pointer to the object (PVOID * instead of PVOID). Callbacks can work around this by doing an extra dereference on the Object address. This has been fixed in .Net.

4. Appendix

The following definitions are from from ntddk.h. Consult that file for the most up-to-date versions. The typedefs describe how the parameters are passed to the callback for each registry API. ntddk.h is the only header a callback writer needs to include in order to get these definitions.

```
//
// Registry kernel mode callbacks
//

//
// Hook selector
//
typedef enum _REG_NOTIFY_CLASS {
    RegNtDeleteKey,
    RegNtPreDeleteKey = RegNtDeleteKey,
    RegNtSetValueKey,
    RegNtPreSetValueKey = RegNtSetValueKey,
    RegNtDeleteValueKey,
    RegNtPreDeleteValueKey = RegNtDeleteValueKey,
    RegNtSetInformationKey,
    RegNtPreSetInformationKey = RegNtSetInformationKey,
    RegNtRenameKey,
    RegNtPreRenameKey = RegNtRenameKey,
    RegNtEnumerateKey,
    RegNtPreEnumerateKey = RegNtEnumerateKey,
    RegNtEnumerateValueKey,
    RegNtPreEnumerateValueKey = RegNtEnumerateValueKey,
    RegNtQueryKey,
    RegNtPreQueryKey = RegNtQueryKey,
    RegNtQueryValueKey,
    RegNtPreQueryValueKey = RegNtQueryValueKey,
    RegNtQueryMultipleValueKey,
    RegNtPreQueryMultipleValueKey = RegNtQueryMultipleValueKey,
    RegNtPreCreateKey,
    RegNtPostCreateKey,
    RegNtPreOpenKey,
    RegNtPostOpenKey,
    RegNtKeyHandleClose,
    RegNtPreKeyHandleClose = RegNtKeyHandleClose,
    //
    // .Net only
    //
    RegNtPostDeleteKey,
    RegNtPostSetValueKey,
    RegNtPostDeleteValueKey,
    RegNtPostSetInformationKey,
    RegNtPostRenameKey,
    RegNtPostEnumerateKey,
    RegNtPostEnumerateValueKey,
    RegNtPostQueryKey,
    RegNtPostQueryValueKey,
    RegNtPostQueryMultipleValueKey,
    RegNtPostKeyHandleClose,
    RegNtPreCreateKeyEx,
    RegNtPostCreateKeyEx,
    RegNtPreOpenKeyEx,
    RegNtPostOpenKeyEx
} REG_NOTIFY_CLASS;

//
// Parameter description for each notify class
//
typedef struct _REG_DELETE_KEY_INFORMATION {
    PVOID               Object;                      // IN
} REG_DELETE_KEY_INFORMATION, *PREG_DELETE_KEY_INFORMATION;

typedef struct _REG_SET_VALUE_KEY_INFORMATION {
    PVOID               Object;                         // IN
    PUNICODE_STRING     ValueName;                      // IN
    ULONG               TitleIndex;                     // IN
    ULONG               Type;                           // IN
    PVOID               Data;                           // IN
    ULONG               DataSize;                       // IN
} REG_SET_VALUE_KEY_INFORMATION, *PREG_SET_VALUE_KEY_INFORMATION;

typedef struct _REG_DELETE_VALUE_KEY_INFORMATION {
    PVOID               Object;                         // IN
    PUNICODE_STRING     ValueName;                      // IN
} REG_DELETE_VALUE_KEY_INFORMATION, *PREG_DELETE_VALUE_KEY_INFORMATION;

typedef struct _REG_SET_INFORMATION_KEY_INFORMATION {
    PVOID                       Object;                 // IN
    KEY_SET_INFORMATION_CLASS   KeySetInformationClass; // IN
    PVOID                       KeySetInformation;      // IN
    ULONG                       KeySetInformationLength;// IN
} REG_SET_INFORMATION_KEY_INFORMATION, *PREG_SET_INFORMATION_KEY_INFORMATION;

typedef struct _REG_ENUMERATE_KEY_INFORMATION {
    PVOID                       Object;                 // IN
    ULONG                       Index;                  // IN
    KEY_INFORMATION_CLASS       KeyInformationClass;    // IN
    PVOID                       KeyInformation;         // IN
    ULONG                       Length;                 // IN
    PULONG                      ResultLength;           // OUT
} REG_ENUMERATE_KEY_INFORMATION, *PREG_ENUMERATE_KEY_INFORMATION;

typedef struct _REG_ENUMERATE_VALUE_KEY_INFORMATION {
    PVOID                           Object;                     // IN
    ULONG                           Index;                      // IN
    KEY_VALUE_INFORMATION_CLASS     KeyValueInformationClass;   // IN
    PVOID                           KeyValueInformation;        // IN
    ULONG                           Length;                     // IN
    PULONG                          ResultLength;               // OUT
} REG_ENUMERATE_VALUE_KEY_INFORMATION, *PREG_ENUMERATE_VALUE_KEY_INFORMATION;

typedef struct _REG_QUERY_KEY_INFORMATION {
    PVOID                       Object;                 // IN
    KEY_INFORMATION_CLASS       KeyInformationClass;    // IN
    PVOID                       KeyInformation;         // IN
    ULONG                       Length;                 // IN
    PULONG                      ResultLength;           // OUT
} REG_QUERY_KEY_INFORMATION, *PREG_QUERY_KEY_INFORMATION;

typedef struct _REG_QUERY_VALUE_KEY_INFORMATION {
    PVOID                           Object;                     // IN
    PUNICODE_STRING                 ValueName;                  // IN
    KEY_VALUE_INFORMATION_CLASS     KeyValueInformationClass;   // IN
    PVOID                           KeyValueInformation;        // IN
    ULONG                           Length;                     // IN
    PULONG                          ResultLength;               // OUT
} REG_QUERY_VALUE_KEY_INFORMATION, *PREG_QUERY_VALUE_KEY_INFORMATION;

typedef struct _REG_QUERY_MULTIPLE_VALUE_KEY_INFORMATION {
    PVOID               Object;                 // IN
    PKEY_VALUE_ENTRY    ValueEntries;           // IN
    ULONG               EntryCount;             // IN
    PVOID               ValueBuffer;            // IN
    PULONG              BufferLength;           // IN OUT
    PULONG              RequiredBufferLength;   // OUT
}                                        REG_QUERY_MULTIPLE_VALUE_KEY_INFORMATION,
*PREG_QUERY_MULTIPLE_VALUE_KEY_INFORMATION;

typedef struct _REG_RENAME_KEY_INFORMATION {
    PVOID            Object;    // IN
```

```
            PUNICODE_STRING  NewName;   // IN
        } REG_RENAME_KEY_INFORMATION, *PREG_RENAME_KEY_INFORMATION;


        typedef struct _REG_KEY_HANDLE_CLOSE_INFORMATION {
            PVOID               Object;         // IN
        } REG_KEY_HANDLE_CLOSE_INFORMATION, *PREG_KEY_HANDLE_CLOSE_INFORMATION;

        /* .Net Only */
        typedef struct _REG_CREATE_KEY_INFORMATION {
            PUNICODE_STRING     CompleteName;   // IN
            PVOID               RootObject;     // IN
        }                                                     REG_CREATE_KEY_INFORMATION,
REG_OPEN_KEY_INFORMATION,*PREG_CREATE_KEY_INFORMATION, *PREG_OPEN_KEY_INFORMATION;

        typedef struct _REG_POST_OPERATION_INFORMATION {
            PVOID               Object;         // IN
            NTSTATUS            Status;         // IN
        } REG_POST_OPERATION_INFORMATION,*PREG_POST_OPERATION_INFORMATION;
        /* end .Net Only */

        /* XP only */
        typedef struct _REG_PRE_CREATE_KEY_INFORMATION {
            PUNICODE_STRING     CompleteName;   // IN
        }                                                 REG_PRE_CREATE_KEY_INFORMATION,
REG_PRE_OPEN_KEY_INFORMATION,*PREG_PRE_CREATE_KEY_INFORMATION,
*PREG_PRE_OPEN_KEY_INFORMATION;;

        typedef struct _REG_POST_CREATE_KEY_INFORMATION {
            PUNICODE_STRING     CompleteName;   // IN
            PVOID               Object;         // IN
            NTSTATUS            Status;         // IN
        }                   REG_POST_CREATE_KEY_INFORMATION,REG_POST_OPEN_KEY_INFORMATION,
*PREG_POST_CREATE_KEY_INFORMATION, *PREG_POST_OPEN_KEY_INFORMATION;
        /* end XP only */
```