# Filter (Intermediate)

### From Crypto++ Wiki

This page examines filter behavior and custom filters in detail. Generally, a user defined filter will derive from `class Filter`. Wei Dai recommends examining `SignerFilter` in filters.h for a filter example.

When creating a custom filter, two functions must be implemented: `Put2` and `IsolatedFlush`. These two functions allow the filter to operate properly whether using an iterative `Put` or streaming through a pipeline in later version of the Crypto++ library. `Put2` is a recent addition to the library, enjoying incorporation around version 5.0. If reusing existing filter code (for example, from a Crypto++ 3.2 filter), `AttachedTransformation()->Put2` may cause compatibility issues since `Put2` was not available when the earlier filter was written.

## Contents



### UselessFilter

In the `UselessFilter` below, the new filter simply forwards the call to `Put2` the underlying `BufferedTransformation`. Note that if there is no underlying filter, 'this' object's `BufferedTransformation` will create a default `MessageQueue` to consume the data. So calls through `AttachedTransformation` are always safe. That is, a call through the pointer returned from `AttachedTransformation` can be made without regards to a `NULL` pointer.

```
class UselessFilter : public Filter
{
public:
    UselessFilter(BufferedTransformation* attachment = NULL)
        : Filter(attachment) { };

    // The new Put interface (circa Crypto++ 5.0)
    size_t Put2(const byte * inString,
        size_t length, int messageEnd, bool blocking )
    {
        return AttachedTransformation()->Put2(
            inString, length, messageEnd, blocking );
    }

    bool IsolatedFlush(bool hardFlush, bool blocking)
    {
        return false;
    }
```

```
};
```

According to Wei, `IsolatedFlush` should not call the base filter's `IsolatedFlush` because the filter's implementation of `Flush` will perform the action as required. See *BufferedTransormation and CreatePutSpace (http://groups.google.com/group/cryptopp-users/browse_thread/thread/b518a39d44b4f92a)* .

Finally, copying and assignment are not allowed on the `UselessFilter`. This is expected since `BufferedTransformation` inherits from `NotCopyable` which hides its copy constructor.

```
UselessFilter useless;
UselessFilter copy(useless);     // Compile error
UselessFilter assign = useless;  // Compile error
```

## BufferlessUselessFilter

`UselessFilter` can be enhanced to provide bufferless uselessness as follows. By deriving from a `Bufferless<Filter>`, the new filter will use the Crypto++ library implementation of `IsolatedFlush`. The library's version of `IsolatedFlush` is also a simple `return false`.

```
class BufferlessUselessFilter : public Bufferless<Filter>
{
public:
    BufferlessUselessFilter(BufferedTransformation* attachment = NULL)
        : Filter(attachment) { };

    size_t Put2(const byte * inString,
        size_t length, int messageEnd, bool blocking )
    {
        return AttachedTransformation()->Put2(
            inString, length, messageEnd, blocking );
    }
};
```

## Anchor Filter

It turns out that `BufferlessUselessFilter` is useful after all. The code below defines an `Anchor` from which other filters can be `Attach`'d and `Detach`'d.

```
class Anchor : public Bufferless<Filter>
{
public:
    Anchor(BufferedTransformation* attachment = NULL)
        { Detach(attachment); };

    // The new Put interface (circa Crypto++ 5.0)
    size_t Put2(const byte * inString,
        size_t length, int messageEnd, bool blocking )
    {
        return AttachedTransformation()->Put2(
            inString, length, messageEnd, blocking );
    }
};
```

`Anchor` offers both `Put(byte, bool)` and `Put(const byte*, size_t, bool)` in addition to `Put2`. This is done to maintain maximum compatibility with down level objects which might call `Put` rather than

`Put2`.

## Attach and Detach

Recall that the `Attach` method is used to attach a `BufferedTransformation` object to the end of the current chain (it does not detach a previously chained object). The next example will examine behavior when attaching a filter to a chain with `Attach`.

In the code below, an `Anchor` is created. Then a `HexEncoder` is attached and data is `Put`. Note that the data is encoded using **lower case** hexadecimal digits. The `HexEncoder` is the removed from the chain. The exercise is then repeated, except that the second `Attach` uses **upper case** encoding.

```
byte data[] = { 0x0A, 0x0B, 0x0C };

string encoded;
Anchor anchor;

anchor.Attach(
    new HexEncoder(
        new StringSink( encoded ),
        false /*LCase*/, 2 /*Group*/, " " /*Separator*/
    )
);

anchor.Put( data, sizeof( data ) );
anchor.MessageEnd();

anchor.Detach();

/* Separator is not a trailer */
encoded += " ";

anchor.Attach(
    new HexEncoder(
        new StringSink( encoded ),
        true /*UCase*/, 2 /*Group*/, " " /*Separator*/
    )
);

anchor.Put( data, sizeof( data ) );
anchor.MessageEnd();

cout << encoded << endl;
```

Output: **`0a 0b 0c 0A 0B 0C`**

Note that `encoded.clear` was not called when the `Anchor` detached its filter, so the data was displayed twice.

## Attach and Attach

In the previous example, `anchor.Detach` was called before `Attach` was called a second time. The next example forgoes the call to `anchor.Detach`.

```
Anchor anchor;

anchor.Attach(
    new HexEncoder(
        new StringSink( encoded ),
        false /*LCase*/, 2 /*Group*/, " " /*Separator*/
```

```
    )
);

anchor.Attach(
    new HexEncoder(
        new StringSink( encoded ),
        true /*UCase*/, 2 /*Group*/, " " /*Separator*/
    )
);

anchor.Put( data, sizeof( data ) );
anchor.MessageEnd();

cout << encoded << endl;
```

Output: **30 61 20 30 62 20 30 63**

The example above has the same effect of creating a pipline as shown below.

```
anchor.Attach(
    new HexEncoder(
        new HexEncoder(
            new StringSink( encoded ),
            true /*UCase*/, 2 /*Group*/,
            " " /*Separator*/
        ),
        false /*LCase*/, 2 /*Group*/,
        " " /*Separator*/
    )
);
```

Output: **30 61 20 30 62 20 30 63**

## MultiByteToWideCharFilter

The `MultiByteToWideCharFilter` performs character conversions from either an ANSI code page or OEM code page to UTF16. It is a wrapper for the Windows function MultiByteToWideChar (http://msdn.microsoft.com/en-us/library/dd319072(VS.85).aspx) . MultiByteToWideChar was chosen because the standard C++ library does not offer a means for determining the required size before conversion. And assumming that the narrow string is 8-bit clean (i.e., only the low 7-bits are significant) will surely cause buffer management problems.

The single constructor for `MultiByteToWideCharFilter` is shown below. `codePage` should be either `CP_ACP` or `CP_OEM` (defined in `windows.h`). There is one defined flag - `THROW_EXCEPTION`. By default, `DEFAULT_FLAGS` = `THROW_EXCEPTION`, so the filter will throw a Crypto++ exception if an error condition is detected.

```
MultiByteToWideCharFilter(BufferedTransformation* attachment = NULL,
    unsigned codePage = CP_ACP, int flags=DEFAULT_FLAGS )
```

This `MultiByteToWideCharFilter` requires that `Put` convert - but **not** buffer - the data. So the filter's `Put` method allocates an appropriately sized buffer, performs a standard character conversion into the buffer, and then calls `Put` on its attached filter using the new buffer.

The downloadable code also demonstrates how to probe memory before a read operation so that an Access Violation is not incurred at runtime. Unfortunately, the technique is only appropriate for `DEBUG` builds.

However, it is an aid during debugging.

```
size_t MultiByteToWideCharFilter::Put2( const byte * inString,
        size_t length, int messageEnd, bool blocking )
{
    // Nothing to process for us. But we will pass it on to the lower
    //  filter just in case...
    if( 0 == length )
    {
        return AttachedTransformation()->Put2( inString, length,
            messageEnd, blocking );
    }

    // Length is not 0. Therefore, the pointer must be valid.
    assert( NULL != inString );
    if( NULL == inString )
    {
        if( THROW_EXCEPTION == (m_flags & THROW_EXCEPTION) )
        {
            throw InvalidArgument(
                "MultiByteToWideCharFilter: inString is NULL" );
        }
        else
        {
            return 0;
        }
    }

    // The first call determines the size of the buffer. See
    //  http://msdn.microsoft.com/en-us/library/dd319072(VS.85).aspx
    unsigned size = (unsigned)MultiByteToWideChar( m_codePage,
        MB_COMPOSITE, (LPCSTR)inString, (int)length, NULL, 0 );
    assert( 0 != size );

    // MultiByteToWideChar returns the required size, in Wide Characters.
    //  So if 0 is returned, there is an ERROR.
    if( 0 == size )
    {
        if( THROW_EXCEPTION == (m_flags & THROW_EXCEPTION) )
        {
            throw OS_Error( OTHER_ERROR,
                "MultiByteToWideCharFilter: Unable to convert multi-byte string",
                "MultiByteToWideChar", GetLastError() );
        }
        else
        {
            return 0;
        }
    }

    // MultiByteToWideChar returns the required size, in
    // Wide Characters (not bytes). If inString included a trailing
    // NULL, the conversion will include a trailing NULL. If inString
    // _did not_ contain a trailing NULL, the conversion _will not_
    // include a trailing NULL
    size_t cb = size * sizeof(wchar_t);
    SecByteBlock buffer( cb );

    if( NULL == buffer.data() || 0 == buffer.size() )
    {
        if( THROW_EXCEPTION == (m_flags & THROW_EXCEPTION) )
        {
            throw Exception( Exception::OTHER_ERROR,
                "MultiByteToWideCharFilter: Out of memory" );
        }
        else
        {
            return 0;
        }
    }

    // Do it for real this time
```

```
    unsigned result = (unsigned)MultiByteToWideChar( m_codePage, MB_COMPOSITE,
        (LPCSTR)inString, (int)length, (LPWSTR)buffer.data(), size );
    assert( result == size );

    // If all characters _were not_ converted, throw...
    if( result != size )
    {
        if( THROW_EXCEPTION == (m_flags & THROW_EXCEPTION) )
        {
            throw OS_Error( OTHER_ERROR,
                "MultiByteToWideCharFilter: Failed to convert multi-byte string",
                "MultiByteToWideChar", GetLastError() );
        }
        else
        {
            return 0;
        }
    }

    return AttachedTransformation()->Put2( buffer,
        buffer.size(), messageEnd, blocking );
}
```


Categories: Sample | Download | Filter