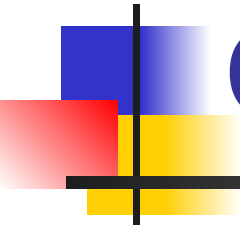
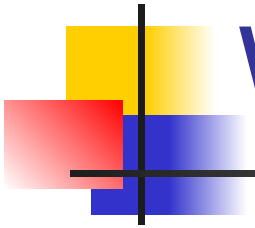


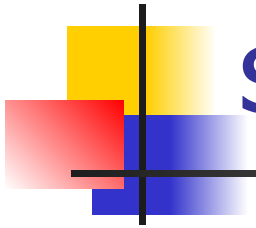
Context Management



What are Contexts?

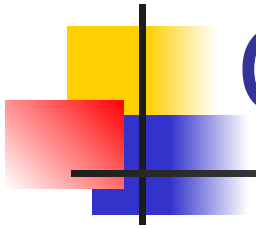
- Minifilter defined memory associated with Filter Manager objects
  - May be from Paged or Non-paged pool
- Which objects can have contexts:
  - Volume
  - Instance
  - File (not currently supported)
  - Stream
  - StreamHandle (FileObject)
- Filter Manager tracks when contexts should be deleted:
  - Object deletion
  - Instance detach
  - Filter unload

Contexts and 3rd party file systems



- To support Stream and StreamHandle contexts a file system must use the FSRTL_ADVANCED_FCB_HEADER
  - See **`FsRtlSetupAdvancedHeader()`** and **`FsRtlTeardownPerStreamContexts()`** in NTIFS.H
  - Also look at the FastFat source in the IFSKit
- All Microsoft file systems now use this

Context Registration



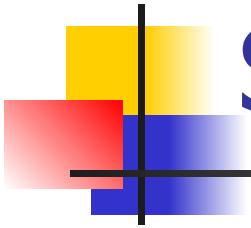
- Specify an array of **`FLT_CONTEXT_REGISTRATION`** structures in **`FLT_REGISTRATION`** structure
  - The order of the entries in this array does not matter
- At least one registration entry must be specified for each context type used by the filter
- **`CleanupContext()`** callback is optional
- 3 different allocation options:
  - Specify 0-3 explicit context sizes (for given context type). System internally uses Lookaside lists
    - Pool tag must be specified
  - Specify **`FLT_VARIABLE_SIZED_CONTEXTS`**, system allocates directly from pool (may be used in conjunction with explicit sizes)
    - Pool tag must be specified
  - Specify your own allocate/free callback routines

Creating Contexts

- Use:

```
NTSTATUS
FltAllocateContext (
    IN PFLT_FILTER Filter,
    IN FLT_CONTEXT_TYPE ContextType,
    IN SIZE_T ContextSize,
    IN POOL_TYPE PoolType,
    OUT PFLT_CONTEXT *ReturnedContext);
```

- Volume contexts must be allocated from non-paged pool
- Contexts limited to 64K in size
- A context size of zero is supported



Setting contexts

- Use:
  - **`FltSetVolumeContext()`**
  - **`FltSetInstanceContext()`**
  - **`FltSetFileContext()`**
  - **`FltSetStreamContext()`**
  - **`FltSetStreamHandleContext()`**
- Sample:

```
NTSTATUS
FltSetStreamHandleContext (
    IN PFLT_INSTANCE Instance,
    IN PFILE_OBJECT FileObject,
    IN FLT_SET_CONTEXT_OPERATION Operation,
    IN PFLT_CONTEXT NewContext,
    OUT PFLT_CONTEXT *OldContext OPTIONAL);
```

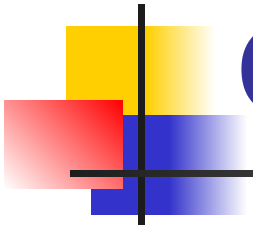
- **`FLT_SET_CONTEXT_OPERATION`** values:
  - **`FLT_SET_CONTEXT_REPLACE_IF_EXISTS`**
    - If specified, replaced context returned in OldContext parameter (must be dereferenced)
  - **`FLT_SET_CONTEXT_KEEP_IF_EXISTS`**
    - If specified, existing context returned in OldContext parameter (must be dereferenced)
- Can not set contexts at DPC level

Supports contexts

- Use:

```
BOOLEAN
FltSupportsFileContexts (
     IN PFILE_OBJECT FileObject);

BOOLEAN
FltSupportsStreamContexts (
     IN PFILE_OBJECT FileObject);

BOOLEAN
FltSupportsStreamHandleContexts (
     IN PFILE_OBJECT FileObject);
```

- Checks to see if contexts are supported on the given FileObject
  - Not supported on paging files
  - Not supported during pre-create
  - Not supported during post-close
  - Not supported during **`IRP_MJ_NETWORK_QUERY_OPEN`**
  - **`FltSupportsFileContexts()`** currently returns FALSE
  - Not supported on file systems that do not use the **`FSRTL_ADVANCED_FCB_HEADER`**



Getting Contexts

- Use:
  - `FltGetVolumeContext()`
  - `FltGetInstanceContext()`
  - `FltGetFileContext()`
  - `FltGetStreamContext()`
  - `FltGetStreamHandleContext()`
- Sample:

```
NTSTATUS
FltGetStreamContext (
    IN PFLT_INSTANCE Instance,
    IN PFILE_OBJECT FileObject,
    OUT PFLT_CONTEXT *Context);
```

- Designed to be retrieved during each operation
- Can not get contexts at DPC level – if a context is needed in a postOperation callback:
  - Get it during the preOperation callback and pass it to the postOperation callback
  - Do your postOperation processing at non-DPC level
    - Synchronize operation
    - Use `FltDoCompletionProcessingWhenSafe()`

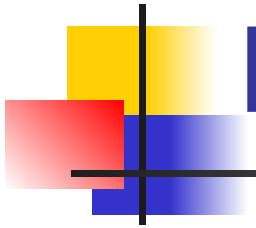
Referencing Contexts

- Use:

```
VOID
FltReferenceContext(
    IN PFLT_CONTEXT Context);
```

- This allows a filter to add their own reference to a context
- Call FltReleaseContext() to remove added reference

Releasing Contexts



- Use:

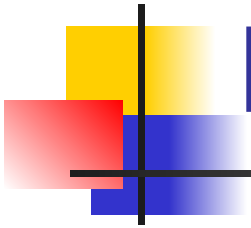
```
VOID
FltReleaseContext (
    IN PFLT_CONTEXT Context);
```

- Contexts need to be released following:
  - Getting (via **`FltGetXxxContext()`**)
  - Creating (Via **`FltAllocateContext()`**)
  - Referencing (via **`FltReferenceContext()`**)
  - A replaced or previous context returned from **`FltSetXxxContext()`** routines
- Contexts may be held from pre to post operations as well as across multiple operations
  - They must eventually be released or the memory will be leaked
- It is OK for a context to point to another context
  - Example: stream context contains a pointer to an instance context
- Contexts **can** be released at DPC level

Multiple contexts

- Use:

```
VOID
FltGetContexts (
    IN PFLT_RELATED_OBJECTS FltObjects,
    IN FLT_CONTEXT_TYPE DesiredContexts,
    OUT PFLT_RELATED_CONTEXTS Contexts);

VOID
FltReleaseContexts (
    IN OUT PFLT_RELATED_CONTEXTS Contexts);
```

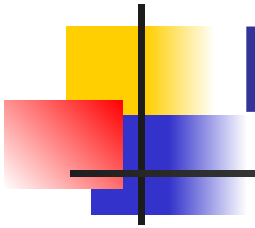


Deleting Contexts

- Use:
  - **`FltDeleteContext()`**
  - **`FltDeleteVolumeContext()`**
  - **`FltDeleteInstanceContext()`**
  - **`FltDeleteFileContext()`**
  - **`FltDeleteStreamContext()`**
  - **`FltDeleteStreamHandleContext()`**
- Sample:

```
VOID
FltDeleteContext (
    IN PFLT_CONTEXT Context);

NTSTATUS
FltDeleteVolumeContext (
    IN PFLT_FILTER Filter,
    IN PFLT_VOLUME Volume,
    OUT PFLT_CONTEXT *OldContext OPTIONAL);
```

- Only use if you have an explicit reason to delete an existing context
  - Filter Manager tracks when contexts should be deleted due to objects going away
- Can not delete contexts at DPC level

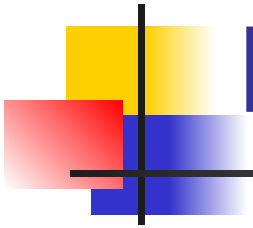


Freeing Contexts

- Contexts are freed after they are deleted and all outstanding references have been released
- **`PFLT_CONTEXT_CLEANUP_CALLBACK`**
  - Unique routine defined for each context type at registration time
  - May be NULL if you don't have any cleanup work to do when context is freed
  - Called by Filter Manager before context is freed
- Contexts are cleaned up in the following hierarchical order
  - StreamHandle
  - Stream
  - File (when implemented)
  - Instance
  - Volume

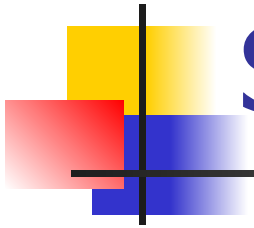
Example: Setting a stream context

```
status = FltAllocateContext( FilterHandle,
                             FLT_STREAM_CONTEXT,
                             sizeof(NCONTEXT),
                             PagedPool,
                             &ctx );
if (NT_SUCCESS(status)) {

    /* Initialize Context Here */

    status = FltSetStreamContext( FltObjects->Instance,
                                  FltObjects->FileObject,
                                  FLT_KEEP_CONTEXT_IF_EXISTS,
                                  ctx,
                                  NULL );

    / * Always release.  If set fails it will free it */
    FltReleaseContext( ctx );
}
```



Performance Suggestions

- If a filter only supports one instance per volume (which most filters do) use instance contexts instead of volume contexts
- Consider putting a pointer to your instance context inside your stream or streamhandle contexts



Sample

- Look at Ctx minifilter sample