Handling Sequence Data in After Effects Effect Plug-ins

by Tobias Fleischer (reduxFX Productions) *v1.2*

In this document, I describe my experiences and test results for handling sequence data in the Adobe After Effects Plug-in C++ API/SDK.

Sequence data plays an important role in an After Effects/Premiere Pro plug-ins, as it allows the developer to attach any custom data to the current effect instance that is also persistently copied, saved and reloaded with the project (contrary to frame or global data of a plug-in, which is not saved and restored).

However, usage and handling of this sequence data is not straightforward and both the SDK documentation and the provided examples are often not as clear, and sometimes even contradicting. So here is my try to shed a little light on that topic.

DISCLAIMER: Please note that I am not affiliated with Adobe and this document is based on my own experiments (and partly guesswork), so no guarantees that everything is correct. If you however find some wrong information here or can contribute by providing additional information, feel free to contact me: tobias.fleischer@reduxfx.com

1. The Sequence Data Commands

There are four API commands that directly influence sequence data:

- PF_Cmd_SEQUENCE_SETUP (when the effect is first applied to a layer)
- PF_Cmd_SEQUENCE_SETDOWN (when the effect is deleted)
- PF_Cmd_SEQUENCE_FLATTEN (when effect is saved, copied or duplicated)
- PF_Cmd_SEQUENCE_RESETUP (when effect is loaded or duplicated)

Each of these commands will be described in detail in this document.

Generally, when the host application calls a command in the effect plug-in, it will also include pointers to PF_InData and PF_OutData data structures as calling parameters in the entry point function like this:

```
DllExport PF_Err EntryPointFunc(
 PF_Cmd cmd,
 PF_InData* in_data, PF_OutData* out_data,
 PF_ParamDef* param[], PF_LayerDef* outputP, void* extraP)
{
...
```

According to comments in the *AE_Effect.h* header file: *"The in_data contains read only information that the effect can use. The out_data contains write only information through which the effect communicates back to the calling program."*

This read-only design is however not enforced and you can actually modify the in_data values from your code (which is also done in some cases, as we will see later).

The in_data and out_data pointers also contain handles to an effect plug-in's global, frame and sequence data, with the last one being of particular interest to us. It is important to keep in mind that the host does lock and unlock these *in_data->sequence_data* and *out_data->sequence_data* (as well as global data and frame data) handles automatically before and after calling the plug-in.

2. The Sequence Data Structure

You are free to store any value you like in your sequence data. In fact, you only allocate memory of a certain size (using AE's internal HandleSuite() and then fill it with your data. This is most commonly done by casting it to developer-defined struct.

Whenever you are storing handles or pointers in your data, you will of course run into trouble when such data is used in a different context, e.g. when a previously saved project with your plug-in applied is loaded - the memory locations of the pointers in your struct will then most certainly be invalid.

Adobe introduced the concept of flat and unflat sequence data to conquer this, giving you the change to alter your sequence data before it is written out to disk or memory. "Flattening" is basically a serialisation process where you can set all fields of your data to self-contained or safe values (no pointers!). The reverse process is called "unflattening" (sometimes also "inflating") and allows you to recover your unflat data from the saved flat one (or re-create it from scratch).

The problem is that the After Effects API does not tell you automatically if your data is "flat" or "unflat". It is therefore recommended behaviour to include a boolean flag as part of your sequence data struct that allows you to discern between these two cases.

Here is a simple example based on the *Pathmaster* sample from the SDK:

```
// struct for flat (fixed) sequence data
struct flatSequenceData
{
    bool isFlatB; // indicator for type of data (flat/unflat)

    int val; // demo: simple int variable
    A_char string[100]; // demo: 100 character string
};
```

```
// struct for unflat (dynamic) sequence data
struct unflatSequenceData
{
    bool isFlatB; // indicator for type of data (flat/unflat)

    int val; // demo: simple int variable
    A_char* stringP; // demo: dynamic length string
};
```

Here the sequence data contains an integer value and a character string with length 100. While the int variable can be opaquely copied from one sequence data to the other, the string is (for demonstration purposes) dynamically allocated in the unflat version of the sequence data and needs to be "flattened" to the fixed-size char array when being written to disk.

Please note that the boolean variable for indicating if the data is flat or unflat (here: *isFlatB*) needs to be at the same position in both structs (e.g. the very first variable), and definitely before any dynamic data! This is required to be able to cast any sequence data you receive to a pointer to one of your structs and then determine the type of data.

This is however only needed if your sequence data actually contains pointers/handles/references - if not, you are in luck and most of the complicated stuff in this document does not really apply to you. Then your sequence data would maybe just look like this:

```
// struct for simple sequence data (no pointers/handles/references)
struct flatSequenceData
{
    bool isFlatB; // indicator for type of data (flat/unflat)

    int val; // demo: simple int variable
};
```

Note that there is a global OutFlag (used for setting global plug-in behaviour) called PF_OutFlag_SEQUENCE_DATA_NEEDS_FLATTENING. According to *AE_Effect.h*, both After Effects and Premiere Pro assume this flag is set in PF_Cmd_GLOBAL_SETUP when the plug-in's sequence data needs flattening, although it is not consistently used in all examples across all SDK versions. Actually, the Adobe hosts seem to ignore this flag (assuming it to be always set) and therefore always send the commands to flatten and unflatten the data.

3. Sequence Setup (PF_Cmd_SEQUENCE_SETUP)

When a plug-in is applied for the first time on a layer, it receives the command PF_Cmd_SEQUENCE_SETUP. It allows the plug-in to allocate and initialize its (unflat) sequence data.

Note: This command is NOT called when loading a saved project or duplicating an effect, in these cases PF_Cmd_SEQUENCE_RESETUP (see next chapter) is used!

Here is my reference implementation of the function handling this command, with lines in *italics* using the variables from the structs above for demonstration purposes:

```
static PF_Err sequenceSetup(PF_InData* in_data, PF_OutData* out_data)
{
    // acquire suite handler
    AEGP_SuiteHandler suites(in_data->pica_basicP);

    // create a new handle, allocating the size of your (unflat)
    // sequence data for it
    PF_Handle unflatSequenceDataH =
           suites.HandleSuite1()->host_new_handle(sizeof(unflatSequenceData));

    // if the handle could not be created, bail out with error
    if (!unflatSequenceDataH) return PF_Err_OUT_OF_MEMORY;

    // lock the handle
    unflatSequenceData* unflatSequenceDataP =
           static_cast<unflatSequenceData*>
                  (suites.HandleSuite1()->host_lock_handle(unflatSequenceDataH));

    // if we could successfully acquire the lock
    if (unflatSequenceDataP)
    {
           // init struct by filling it with zeros
           AEFX_CLR_STRUCT(*unflatSequenceDataP);

           // IMPORTANT: set flag for being "unflat"
           unflatSequenceDataP->isFlatB = false;

           // demo: set int variable
           unflatSequenceDataP->val = 123;

           // demo: allocate dynamic string
           unflatSequenceDataP->stringP = new A_char[101];
           if (unflatSequenceDataP->stringP) {
                  // copy text to dynamic string in memory
                  suites.ANSICallbacksSuite1()->strcpy(
                         unflatSequenceDataP->stringP, "This is a test");
           }

           // IMPORTANT: notify AE that this is your sequence data handle!
           out_data->sequence_data = unflatSequenceDataH;

           // unlock handle again
           suites.HandleSuite1()->host_unlock_handle(unflatSequenceDataH);
    }
    return PF_Err_NONE;
}
```

Initially both *in_data->sequence_data* and *out_data->sequence_*data are NULL when PF_Cmd_SEQUENCE_SETUP is called, at least in all my test cases with After Effects and Premiere Pro. You are expected to create/allocate/initialize your sequence data in this function and write the handle of the allocated data to *out_data->sequence_data* to inform the host that this is the plug-in's current sequence data.

Note that we lock and unlock the newly created unflat sequence data in this code snippet, although in that case it is not strictly necessary (since we just created the data ourselves and no other thread/process knows yet about it). You could directly get the data pointer associated with a handle by using the DH macro and doing a static cast like this:

```
sequenceData* sequenceDataP = static_cast<sequenceData*>(DH(sequenceDataH));
```

The same logic applies to some other functions in this document – but I always find it good practice to lock and unlock data pointers before accessing/modifying its values. The only exception to this are the *in_data* and *out_data* sequence data pointers, as they are automatically locked and unlocked by the host, as described in chapter 1.

There is a field in the out_data struct called "*flat_sdata_size*" that some examples from the SDK set to the size of the unflat sequence data. This is however obsolete and unneeded behaviour as far as I know. The *AE_Effect.h* header even contains a comment that says: *"This is obsolete. Because you used its allocation functions to get the memory in the first place, the host can find out the size of the handle without asking."*

In relation to this, another interesting thing to keep in mind is that since we have to re-allocate our unflat as well as flat sequence data in each flattening/deflattening process, we can even use arbitrarily sized *flat* data, as long as we are able to recover the relevant unflat data from it. Since the host knows the size of the handle (which equals to the size of the memory bock associated with this handle), we can create and return a handle that has exactly the size we need to store the flattened/serialized data. When unflattening, we can query the size of a handle with PF_HandleSuite1->host_get_handle_size(), *or* we could store the the size of the flat sequence data size somewhere in our struct (before the actual dynamic data). This is especially useful for storing data where the serialized length is unknown beforehand and the developer does not want to always allocate the "worst-case" maximum length for each flat sequence data.

4. Sequence Setdown (PF_Cmd_SEQUENCE_SETDOWN)

The command PF_Cmd_SEQUENCE_SETDOWN is sent when the sequence data is to be freed/deleted. This is usually the case when the user removes an effect or closes the project/quits the application.

Note that as of After Effects 6.0, if an effect's sequence data has recently been flattened, the effect may be deleted without receiving an additional PF_Cmd_SEQUENCE_SETDOWN command. In this case, After Effects will dispose of your flat sequence data automatically.

Here is my reference implementation of the function handling this command, with lines in *italics* using the variables from the structs above for demonstration purposes:

```
static PF_Err sequenceSetdown(PF_InData* in_data, PF_OutData* out_data)
{
    // acquire suite handler
    AEGP_SuiteHandler suites(in_data->pica_basicP);

    // if sequence data exists
    if (in_data->sequence_data)
    {
           // get its handle ...
           PF_Handle unflatSequenceDataH = in_data->sequence_data;

           // ... and from that its actual data pointer
           unflatSequenceData* unflatSequenceDataP =
                  static_cast<unflatSequenceData*>(DH(unflatSequenceDataH));

           // check if it is flat or unflat data
           // it should always be unflat, but just to make sure
           if (!unflatSequenceDataP->isFlatB)
           {
                  // delete any dynamically allocated pointers here

                  // demo: delete the dynamic string
                  delete [] unflatSequenceDataP->stringP;
           }

           // IMPORTANT: dispose the handle, freeing your sequence data
           suites.HandleSuite1()->host_dispose_handle(unflatSequenceDataH);
    }

    // invalidate the sequence_data pointers in both AE's input
    // and output data fields (to signal that we have properly
    // disposed of the data).
    in_data->sequence_data = NULL;
    out_data->sequence_data = NULL;

    // return without error
    return PF_Err_NONE;
}
```

Note that the *in_data->sequence_data* gets set to NULL, as in most included example plugins. This works without an error, although the *AE_Effect.h* header contains a comment that claims *"The in_data contains read only information"*. Setting the sequence_data pointer there to NULL most probably has no effect, at least in AE and PPro.

5. Sequence Flatten (PF_Cmd_SEQUENCE_FLATTEN)

The command PF_Cmd_SEQUENCE_FLATTEN is sent when the sequence data is to be flattened. This is usually the case when the current data for a plug-in is copied to the clipboard, the effect is duplicated, or - the most common case - when the project is saved and the effect's sequence data is supposed to be written on disk.

If your sequence data does not contain any pointers or handlers, you can simply skip the implementation of this function. This works because *out_data->sequence_data* is (or should be) initialized by the host to be pointing to the same handle as *in_data->sequence_data*, so if nothing is done with this data, the host can assume that the data there is "flat-safe".

Otherwise, when handling this command, you are expected to allocate a new structure for your flat sequence data, populate it with the data from the unflat sequence data (given to you in *in_data->sequence_data*) while getting rid of any pointers. Then delete the (unflat) *in_data->sequence_data* and store the handle to your new (flat) sequence data in *out_data->sequence_data*.

Interestingly, in the AE SDK doc, it says in the description for PF_Cmd_SEQUENCE_FLATTEN on what you are supposed to do in this command: *"**Free the unflat data** and set the out_data->sequence_data to point to the new flattened data"*

If we however look for example at the *Pathmaster* example from the SDK, within the *SequenceFlatten()* function, there is the following comment:

// Make a flat copy of whatever is in the unflat seq data handed to us.
// Do NOT delete the unflat one!

(And the unflat data is of course not deleted here.)

Only in the most recent CC2014 SDK, in the *PathMaster* sample this comment was removed and the unflat data is actually disposed here, so we can assume the example has been incorrect all these years and you are actually supposed to get rid of the unflat data.

Note that if you set the *out_data->sequence_data* to NULL, you will NOT get the sequence resetup call to unflatten it. Instead, you may just get a RENDER call with NULL data. Don't do that unless you know what you are doing.

Here is my reference implementation of the function handling this command, with lines in *italics* using the variables from the structs above for demonstration purposes:

```
static PF_Err sequenceFlatten(PF_InData* in_data, PF_OutData* out_data)
{
    PF_Err err = PF_Err_NONE;

    // acquire the suite handler
    AEGP_SuiteHandler suites(in_data->pica_basicP);

    // if sequence data is present
    if (in_data->sequence_data)
    {
          // assume it's always unflat data and get its handle ...
          PF_Handle unflatSequenceDataH = in_data->sequence_data;

          // ... then get its actual data pointer
          unflatSequenceData* unflatSequenceDataP =
                static_cast<unflatSequenceData*>(DH(unflatSequenceDataH));

          if (unflatSequenceDataP)
          {
                // create a new handle, allocating the size of your (flat)
                // sequence data for it
                PF_Handle flatSequenceDataH =
                      suites.HandleSuite1()->host_new_handle(
                            sizeof(flatSequenceData));

                if (flatSequenceDataH)
                {
                      // lock and get actual data pointer for flat data
                      flatSequenceData* flatSequenceDataP =
                            static_cast<flatSequenceData*>(
                                  suites.HandleSuite1()->host_lock_handle(
                                        flatSequenceDataH));

                      if (flatSequenceDataP)
                      {
                            // init struct by filling it with zeros
                            AEFX_CLR_STRUCT(*flatSequenceDataP);

                            // IMPORTANT: set flag for being "flat"
                            flatSequenceDataP->isFlatB = true;

                            // demo: directly copy int value unflat -> flat
                            flatSequenceDataP->val = unflatSequenceDataP->val;

                            // demo: copy dynamic string to static string
#ifdef AE_OS_WIN
                            strncpy_s(flatSequenceDataP->string,
                                  unflatSequenceDataP->stringP, 100);
#else
                            strncpy(flatSequenceDataP->string,
                                  unflatSequenceDataP->stringP, 100);
#endif

                            // IMPORTANT: notify AE of new flat sequence data
                            out_data->sequence_data = flatSequenceDataH;

                            // unlock flat sequence data handle
                            suites.HandleSuite1()->host_unlock_handle(
                                  flatSequenceDataH);
```

```
                        }
                }
                else
                {
                        err = PF_Err_INTERNAL_STRUCT_DAMAGED;
                }


                // IMPORTANT: delete any dynamically allocated pointers
                // in the unflat data before disposing it!
                // Also, set the pointers to null pointers just in case

                // demo: delete and nullify the dynamic string
                delete [] unflatSequenceDataP->stringP;
                unflatSequenceDataP->stringP = NULL;

                // IMPORTANT: dispose unflat sequence data!
                suites.HandleSuite1()->host_dispose_handle(
                        unflatSequenceDataH);
                in_data->sequence_data = NULL;
        }
    }
    else
    {
        err = PF_Err_INTERNAL_STRUCT_DAMAGED;
    }
    return err;
}
```

Please note that just like in PF_Cmd_SEQUENCE_SETDOWN, the *in_data->sequence_data* is here also commonly set to NULL, although originially designed to be read-only. It does not seem to cause any problem though.

6. Sequence Resetup (PF_Cmd_SEQUENCE_RESETUP)

From the SDK: *"This command is sent to re-create (usually unflatten) sequence data. Sent after sequence data is read from disk, during pre-composition, or when the effect is copied; After Effects flattens sequence data before duplication. During duplication, PF_Cmd_SEQUENCE_RESETUP is sent for both the old and new sequences. Don't expect a PF_Cmd_SEQUENCE_FLATTEN between PF_Cmd_SEQUENCE_RESETUPs."*

Again, good news if you don't have any problematic unflat data in your sequence data, as then you can just skip the implementation of this command, just like PF_Cmd_SEQUENCE_FLATTEN.

Otherwise, in PF_Cmd_SEQUENCE_RESETUP you should check for the boolean indicating that the data is flattened, and if so, you should unflatten the data, free the flattened data handle, and set the *out_data->sequence_data* handle.

Here is my reference implementation of the function handling this command, with lines in *italics* using the variables from the structs above for demonstration purposes:

```
static PF_Err sequenceResetup(PF_InData* in_data, PF_OutData* out_data)
{
    PF_Err err = PF_Err_NONE;

    // acquire suite handler
    AEGP_SuiteHandler suites(in_data->pica_basicP);

    // if sequence data is present
    if (in_data->sequence_data)
    {
           // get handle to flat data ...
           PF_Handle flatSequenceDataH = in_data->sequence_data;

           // ... then get its actual data pointer
           flatSequenceData* flatSequenceDataP =
                  static_cast<flatSequenceData*>(DH(flatSequenceDataH));

           if (flatSequenceDataP && flatSequenceDataP->isFlatB)
           {
                  // create a new handle, allocating the size of your (unflat)
                  // sequence data for it
                  PF_Handle unflatSequenceDataH =
                         suites.HandleSuite1()>host_new_handle(
                                sizeof(unflatSequenceData));

                  if (unflatSequenceDataH)
                  {
                         // lock and get actual data pointer for unflat data
                         unflatSequenceData* unflatSequenceDataP =
                                static_cast<unflatSequenceData*>(
                                       suites.HandleSuite1()->host_lock_handle(
                                              unflatSequenceDataH));

                         if (unflatSequenceDataP)
                         {
```

```
                                // init struct by filling it with zeros
                                AEFX_CLR_STRUCT(*unflatSequenceDataP);

                                // IMPORTANT: set flag for being "unflat"
                                unflatSequenceDataP->isFlatB = false;


                                // demo: directly copy int value unflat -> flat
                                unflatSequenceDataP->val = flatSequenceDataP->val;

                                // demo: copy static string to dynamic string
                                A_short lengthS =
                                        (A_short)strlen(flatSequenceDataP->string);
                                unflatSequenceDataP->stringP =
                                        new A_char[lengthS + 1];
                                if (unflatSequenceDataP->stringP)
                                {
                                        suites.ANSICallbacksSuite1()->strcpy(
                                                unflatSequenceDataP->stringP,
                                                flatSequenceDataP->string);
                                }

                                // IMPORTANT: notify AE of unflat sequence data
                                out_data->sequence_data = unflatSequenceDataH;

                                // IMPORTANT: dispose flat sequence data!
                                suites.HandleSuite1()->host_dispose_handle(
                                        flatSequenceDataH);
                                in_data->sequence_data = NULL;
                        }
                        else
                        {
                                err = PF_Err_INTERNAL_STRUCT_DAMAGED;
                        }

                        // unlock unflat sequence data handle
                        suites.HandleSuite1()->host_unlock_handle(
                                unflatSequenceDataH);
                }
        }
        else
        {
                // use input unflat data as unchanged output
                out_data->sequence_data = in_data->sequence_data;
        }
    }
    else
    {
        // no sequence data exists? create one!
        err = sequenceSetup(in_data, out_data);
    }

    return err;
}
```

7. Sequence of Sequence Data Commands

WARNING: The following information applies to After Effects versions prior and including CC 2014! With AE CC 2015/v13.5, significant architectural changes were made that results in some slightly different command order. This documetn will be updated at a later time to reflect the current AE versions.

Here is a little test procedure that I applied to determine the actual sequence of commands called by After Effects relevant to the handling of flat/unflat sequence data:

1. start AE, create new project, add layer, apply effect plug-in
2. duplicate effect plug-in
3. save project
4. close application

And here is a log of the sequence of commands concerning the sequence data for this test, using the code from the snippets above:

// action: start application, create new project, apply effect on new layer
cmd PF_Cmd_SEQUENCE_SETUP (in_data->sequence_data: 0)
create new unflat sequence data, store in handle H1
out_data->sequence_data = H1
// active handles: H1 (unflat)

// action: duplicate effect (Ctrl-D)
cmd PF_Cmd_SEQUENCE_FLATTEN (in_data->sequence_data: H1)
create new flat sequence data, copy data from H1, store in handle H2
dispose unflat sequence data with handle H1
out_data->sequence_data = H2
cmd PF_Cmd_SEQUENCE_RESETUP (in_data->sequence_data: H3)
// H3 is the flat data of the new effect (created by the host by copying H2)
create new unflat sequence data, copy data from H3, store in handle H4
dispose flat sequence data with handle H3
out_data->sequence_data = H4
cmd PF_Cmd_SEQUENCE_RESETUP (in_data->sequence_data: H2)
// H2 is the flat data of the original effect
create new unflat sequence data, copy data from H2, store in handle H5
dispose flat sequence data with handle H2
out_data->sequence_data = H5
// active handles: H4 (unflat), H5 (unflat)

// action: save project (CTRL-S)
cmd PF_Cmd_SEQUENCE_FLATTEN (in_data->sequence_data: H5)
create new flat sequence data, copy data from H5, store in handle H6
dispose unflat sequence data with handle H5
out_data->sequence_data = H6
cmd PF_Cmd_SEQUENCE_RESETUP (in_data->sequence_data: H6)
create new unflat sequence data, copy data from H6, store in handle H7
dispose flat sequence data with handle H6
out_data->sequence_data = H7
cmd PF_Cmd_SEQUENCE_FLATTEN (in_data->sequence_data: H4)
create new flat sequence data, copy data from H4, store in handle H8
dispose unflat sequence data with handle H4
out_data->sequence_data = H8
cmd PF_Cmd_SEQUENCE_RESETUP (in_data->sequence_data: H8)
create new unflat sequence data, copy data from H8, store in handle H9
dispose flat sequence data with handle H8
out_data->sequence_data = H9
// active handles: H7 (unflat), H9 (unflat)

// action: close project, quit application
cmd PF_Cmd_SEQUENCE_SETDOWN (in_data->sequence_data: H9)
dispose unflat sequence data with handle H9
out_data->sequence_data = 0
cmd PF_Cmd_SEQUENCE_SETDOWN (in_data->sequence_data: H7)
dispose unflat sequence data with handle H7
out_data->sequence_data = 0
// active handles: -

8. Premiere Pro particularities

While Premiere Pro also allows the user to load and apply After Effects effect plug-ins, its underlying structure is in some parts fundamentally different to AE. I will concentrate on the differences relevant to sequence data only here, although there are many other things for a developer to keep in mind if he wants a plugin to work correctly in both AE and PPro.

The biggest difference is the internal rendering process of Premiere Pro. Premiere Pro renders AE effects in a multithreaded environment by running various rendering threads and feeding them exact copies (clones) of the effect's current unflat sequence data. For the plug-in it is however not possible to determine if the sequence data was cloned, therefore sequence data needs to be considered read-only during rendering! Any changes made to the sequence data during rendering will be discarded. Additionally, you will run into problems if your sequence data contains data that is not thread-safe, e.g. a pointer to an instance of an object that is not protected/synchronized.

A new output flag has therefore been added to the AE API starting from CS4.1: PF_OutFlag2_PPRO_DO_NOT_CLONE_SEQUENCE_DATA_FOR_RENDER

This flag is only supported in Premiere Pro and not in After Effects and it allows a plug-in to opt out of Premiere Pro's multithreaded rendering of AE effects. From the SDK: *"When the flag is set, we don't clone the sequence data across all the threads and we only call into the plug-in on one thread at a time. Premiere Pro will still render using multiple threads, but the effect will only render on one thread at a time, and the same sequence data will be used. This flag is useful for plug-ins that provide their own internal multithreading, or plug-ins that render frames based on previous frames, such as image stabilizers."*

Please note however that in the AE SDK documentation it also says: ***"We advise against setting this flag, as it has been found to cause parameter UI problems."***

I stumbled upon these "parameter UI problems" very quickly: when the flag is set, updating, renaming, disabling or hiding of UI parameters in Premiere Pro does no longer work reliably, so that is a pretty big trade-off if you are manipulating the appearance of your UI parameters, although there are some workarounds concerning parameter handling. It is apparently logged as bug #3197343 at Adobe, although apparently still present in the current versions.

Another thing to keep in mind when working with Premiere Pro is that on some Premiere Pro versions (namely CS5 and CS5.5), you get a PF_Cmd_RENDER call before a call to PF_Cmd_SEQUENCE_SETUP or PF_Cmd_SEQUENCE_RESETUP, meaning you don't get any valid sequence data for your first render. It is usually a good idea in that case to bail out of your render function if no valid sequence data is present.

Additionally, if an effect is deleted or a project is closed, Premiere Pro does not always send a PF_Cmd_SEQUENCE_SETDOWN command, although the logic (bug?) or behaviour behind that is still unknown.