Devicetree Overlay Notes

This document describes the implementation of the in-kernel device tree overlay functionality residing in drivers/of/overlay.c and is a companion document to Documentation/devicetree/dynamic-resolution-notes.rst[1]

How overlays work

A Devicetree's overlay purpose is to modify the kernel's live tree, and have the modification affecting the state of the kernel in a way that is reflecting the changes. Since the kernel mainly deals with devices, any new device node that result in an active device should have it created while if the device node is either disabled or removed all together, the affected device should be deregistered.

Lets take an example where we have a foo board with the following base tree:

```
---- foo.dts ---------------------------------------------------------------
    /* FOO platform */
    /dts-v1/;
    / {
            compatible = "corp,foo";

            /* shared resources */
            res: res {
            };

            /* On chip peripherals */
            ocp: ocp {
                    /* peripherals that are always instantiated */
                    peripheral1 { ... };
            };
    };
---- foo.dts ---------------------------------------------------------------
```

The overlay bar.dts,

```
---- bar.dts - overlay target location by label ----------------------------
    /dts-v1/;
    /plugin/;
    &ocp {
            /* bar peripheral */
            bar {
                    compatible = "corp,bar";
                    ... /* various properties and child nodes */
            };
    };
---- bar.dts ---------------------------------------------------------------
```

when loaded (and resolved as described in [1]) should result in foo+bar.dts:

```
---- foo+bar.dts -----------------------------------------------------------
    /* FOO platform + bar peripheral */
    / {
            compatible = "corp,foo";

            /* shared resources */
            res: res {
            };

            /* On chip peripherals */
            ocp: ocp {
                    /* peripherals that are always instantiated */
                    peripheral1 { ... };

                    /* bar peripheral */
                    bar {
                            compatible = "corp,bar";
                            ... /* various properties and child nodes */
                    };
            };
    };
---- foo+bar.dts -----------------------------------------------------------
```

As a result of the overlay, a new device node (bar) has been created so a bar platform device will be registered and if a matching device driver is loaded the device will be created as expected.

If the base DT was not compiled with the -@ option then the "&ocp" label will not be available to resolve the overlay node(s) to the proper location in the base DT. In this case, the target path can be provided. The target location by label syntax is preferred because the overlay can be applied to any base DT containing the label, no matter where the label occurs in the DT.

The above bar.dts example modified to use target path syntax is:

```
---- bar.dts - overlay target location by explicit path --------------------
    /dts-v1/;
    /plugin/;
    &{/ocp} {
            /* bar peripheral */
            bar {
                    compatible = "corp,bar";
                    ... /* various properties and child nodes */
            }
    };
---- bar.dts ---------------------------------------------------------------
```

Overlay in-kernel API

The API is quite easy to use.

1. Call of_overlay_fdt_apply() to create and apply an overlay changeset. The return value is an error or a cookie identifying this overlay.
2. Call of_overlay_remove() to remove and cleanup the overlay changeset previously created via the call to of_overlay_fdt_apply(). Removal of an overlay changeset that is stacked by another will not be permitted.

Finally, if you need to remove all overlays in one-go, just call of_overlay_remove_all() which will remove every single one in the correct order.

In addition, there is the option to register notifiers that get called on overlay operations. See of_overlay_notifier_register/unregister and enum of_overlay_notify_action for details.

Note that a notifier callback is not supposed to store pointers to a device tree node or its content beyond OF_OVERLAY_POST_REMOVE corresponding to the respective node it received.