OpenWF Display Configuration Developer's Guide:

Modifying the Wfdcfg library

QNX Software Systems Limited
1001 Farrar Road
Ottawa, Ontario
K2K 0B3
Canada

Voice: +1 613 591-0931
Fax: +1 613 591-3579
Email: info@qnx.com
Web: http://www.qnx.com/

Electronic edition published: Wednesday, October 8, 2014

Table of Contents

About OpenWF Display Configuration

If you are integrating a new display, you must configure and provide the parameters of your display based on its specifications via the OpenWF Display Configuration API (Wfdcfg Library).

This table may help you find what you need in this guide:

To find out about:	Go to:
What you need	*Before you begin* (p. 11)
How to get the source code	*Getting the source code* (p. 15)
How to set timing parameters	*Setting timing parameters* (p. 19)
How to add extensions to the Wfdcfg library	*Updating Wfdcfg source (adding extensions)* (p. 23)
How to build the Wfdcfg library	*Building the Wfdcfg library* (p. 27)
How to update your target	*Updating your target* (p. 23)
How to configure your display	*Configuring Screen for your display* (p. 27)
Wfdcfg API	*Wfdcfg Library Reference*

Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications.

The following table summarizes our conventions:

Reference	Example
Code examples	`if( stream == NULL)`
Command options	`-lR`
Commands	`make`
Constants	`NULL`
Data types	`unsigned short`
Environment variables	***PATH***
File and pathnames	`/dev/null`
Function names	*exit()*
Keyboard chords	**Ctrl–Alt–Delete**
Keyboard input	`Username`
Keyboard keys	**Enter**
Program output	`login:`
Variable names	*stdin*
Parameters	*parm1*
User-interface components	**Navigator**
Window title	**Options**

We use an arrow in directions for accessing menu items, like this:

You'll find the Other... menu item under **Perspective** **Show View**.

We use notes, cautions, and warnings to highlight important messages:

Notes point out something important or useful.

Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we typically use a forward slash (/) as a delimiter in pathnames, including those pointing to Windows files. We also generally follow POSIX/UNIX filesystem conventions.

Technical support

Technical assistance is available for all supported products.

To obtain technical support for any QNX product, visit the Support area on our website (*www.qnx.com*). You'll find a wide range of support options, including community forums.

Chapter 1
About OpenWF Display Configuration

If you are integrating a new display, you must configure and provide the parameters of your display based on its specifications via the OpenWF Display Configuration API (Wfdcfg Library).

This table may help you find what you need in this guide:

To find out about:	Go to:
What you need	*Before you begin* (p. 11)
How to get the source code	*Getting the source code* (p. 15)
How to set timing parameters	*Setting timing parameters* (p. 19)
How to add extensions to the Wfdcfg library	*Updating Wfdcfg source (adding extensions)* (p. 23)
How to build the Wfdcfg library	*Building the Wfdcfg library* (p. 27)
How to update your target	*Updating your target* (p. 23)
How to configure your display	*Configuring Screen for your display* (p. 27)
Wfdcfg API	*Wfdcfg Library Reference*

Chapter 2
Before you begin

This guide is intended to describe how to integrate a new display by setting the appropripate parameters based on your display specifications.

You must have the following:

- Either one of:
  - QNX Software Development Platform 6.6.0 with QNX Software Development Platform 6.6.0 Graphics patch 3875 (available from our website, *http://www.qnx.com/*)

  or

  - QNX Software Development Platform 6.5.0 with Service Pack 1 and a recent, compatible Screen package

  installed on your development host
- A target that's compatible with QNX Neutrino 6.6.0 or QNX Neutrino6.5.0 SP1
- a processor-specific BSP archive that's available from our website, *http://www.qnx.com/* and is compatible with your target.
- a directory structure with a *bsp_working_dir* that's a result from unzipping the processor-specific BSP archive. For more information on BSPs, refer to *Building Embedded Systems*, or to the BSP User Guide. The BSP User Guide that is specific to your target is available from from our website, www.qnx.com.
- Supported display hardware
- Specifications for your display hardware
- the most recent Wfdcfg archive (e.g., `src-lib-wfdcfg-2014-07-31-Moonraker-B556.tgz`); archives that are specific for your target may be available

Chapter 3
Introduction to the Wfdcfg library

The Wfdcfg library provides the modes and attributes of your display hardware to your display driver and Screen.

Wfdcfg Library

Your display driver (WFD driver) is the primary user of the Wfdcfg library, but the composition manager component in Screen also uses the modes and attributes from the Wfdcfg library.

The composition manager accesses the OpenWFD modes and attributes from the Wfdcfg library through the WFD driver. Based on this information, the composition manager tells the WFD driver which OpenWFD mode to use.

For the most part, it'll be your WFD driver that interfaces with the Wfdcfg library. Examples of what your driver can do through the Wfdcfg library are:

- Apply timing parameters
- Control pixel clock frequency
- Provide output formats
- Control backlighting in Wfdcfg library through callbacks

What your WFD driver can control or provide through the Wfdcfg library is dependent on your specific display hardware.

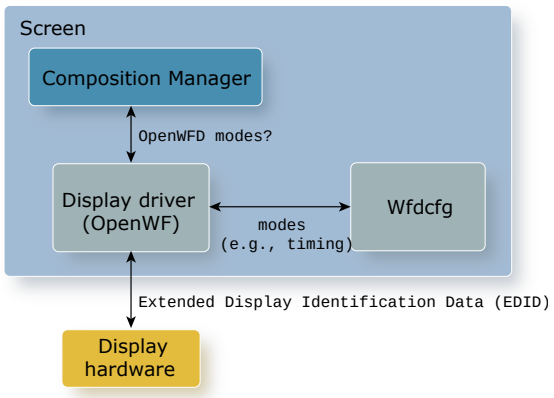


Figure 1: How Wfdcfg interacts with Screen

Some WFD drivers also probe the display hardware for extended display identification data (EDID). This information is consolidated and provided to the composition manager.

Chapter 4
Getting the source code

The files from the compressed Wfdcfg archive are extracted onto your host.

Compressed Wfdcfg archive (`.tgz`) file

The source code for the Wfdcfg library is provided as a compressed archive (`.tgz` file format). This compressed archive usually contains a single `.tar` archive.

The Wfdcfg archive is named according to the following convention:

`src-lib-wfdcfg-platform-`*`build_date-branch-build_number.file_format`*

where the variables in the filename are as follows:

File element	Description
platform	This variable is optional. If your archive is specific to your platform, you may see the platform identified in the filename.
build_date	Date and time stamp in the format of *yyyy-mm-dd*
branch	The repository branch
build_number	The identification number of the particular build
build_number	The identification number of the particular build
file_format	• *tgz* for the compressed archive • *tar* archive

For example, a Wfdcfg archive can be named: `src-lib-wfdcfg-2014-07-31-Moonraker-B556.tgz` or `src-lib-wfdcfg-imx6x-evk-wfdcfg-2014-07-31-Moonraker-B556.tgz`

Extracting files

You can extract the files from the Wfdcfg archive to a directory of your choice. The archive is self-contained and can be installed anywhere. The directory where you extract the Wfdcfg archive will be referred to as `wfdcfg_working_dir`.

From the command line on your development host:

1. Go to your *wfdcfg_working_dir*.

```
cd wfdcfg_working_dir
```

2. Extract the files from the Wfdcfg archive using the `tar` command.

   For more information on the `tar` command, refer to the *Utilities Reference*.

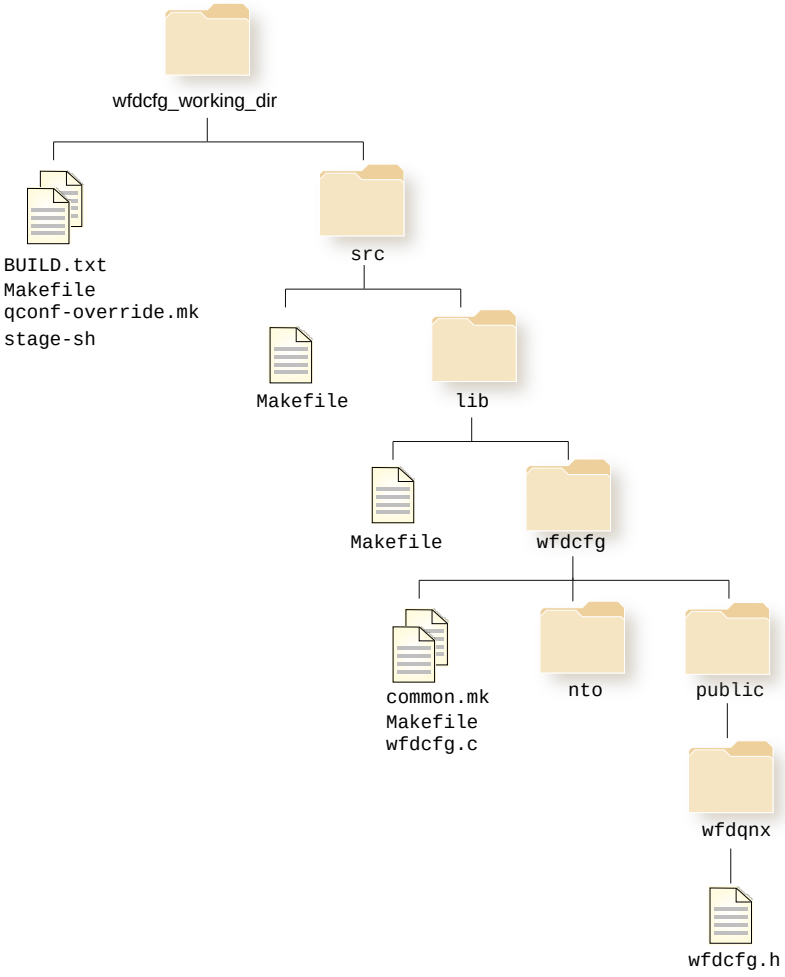
   ```
   tar -xvf wfdcfg-archive.tgz
   ```

If you are on a Windows development host, *and* you are not using the `tar` utility, ensure that you are using a compatible file archiver that properly handles symbolic links.

You can manually copy the appropriate files if you find an issue with your symbolic links. For a list of symbolic links in the archive, you can use a command similar to this:

```
tar -tvf wfdcfg_archive.tgz | grep ^l
```

After you've extracted the files from the wfdcfg archive, the resulting directory structure looks similar to this:



If you have a platform-specific archive, you may see some slight differences in the directory structure.

Figure 2: wfdcfg archive directory structure.

You'll update the files `wfdcfg.h` and `wfdcfg.c` to add extensions that are specific for your display hardware. Refer to *Updating Wfdcfg source (adding extensions)* (p. 23) for more information on how to modify the Wfdcfg library.

`BUILD.txt` provides you with some simple directions on how to build your source. `stage-sh` is a helper script that creates a stage area from where you can build your source. Refer to *Building the Wfdcfg library* (p. 27) for information on how to build the Wfdcfg library.

Chapter 5
Setting timing parameters

Timing is mandatory component of a mode.

The timing parameters are related to the horizontal and vertical blanking intervals. These intervals refer to a part of the process of displaying images on a computer monitor or television screen via raster scanning.

The horizontal blanking interval occurs once per line of image information and is composed of: a horizontal sync pulse, a front porch, and a back porch.

The vertical blanking interval is the time betweeen the end of an active image and the start of the next; it is composed of: a vertical sync pulse, a front porch, and a back porch.

Timing parameters

You'll define the timing parameters based on your display hardware using the `wfd cfg_timing` structure in Wfdcfg. You'll need the following timing parameters to be set:

`pixel_clock_kHz`

The frequency (in kHz) that pixels are transmitted at. The clock remains active throughout the entire horizontal and vertical blanking intervals, even when pixels are not being transmitted.

`hpixels`

The width (in pixels) of the display. Together with `vlines`, `hpixels` indicates the resolution of the display.

`vlines`

The height (in lines) of the display. Together with `hpixels`, `vlines` indicates the resolution of the display.

`hsw`

The width of the horizontal synchronization pulse. This width refers to the amount of time that the horizontal sync pulse is active. The horizontal sync pulse is transmitted at the beginning of each video scanline. Its purpose is to keep start of the horizontal video scanline in the display in sync with the video source that created it. That is, when the scanline reaches the right side of your display, the horizontal sync pulses indicates that it is time to

return and start the next scanline at the left side of the display again. The width of this horizontal synchronization pulse is measured in pixels.

vsw

The vertical synchronization pulse width. This width refers to the amount of time that the vertical sync pulse is active. The vertical sync pulse is transmitted at the beginning of each field and frame. Its purpose is to ensure that the display scan starts at the top of the picture at the right time. That is, when the last scanline on the bottom of the display has been reached, the display must return and start the next scanline back at the top of the screen for the next vertical cycle. The width of this vertical synchronization pulse is measured in lines.

hfp

The horizontal front porch is the amount of time between the end of the horizontal active time and the start of the horizontal synchronization pulse. This time allows for the image to settle and to prevent the image from interfering with sync extraction. The horizontal front porch is measured in pixels.

vfp

The vertical front porch is the amount of time between the end of the vertical active time and the start of the vertical synchronization pulse. The vertical front porch is measured in lines.

hbp

The horizontal back porch is the amount of time between the end of the horizontal sync pulse and the start of the next horizontal active time. The horizontal back porch is measured in pixels.

vbp

The vertical back porch is the amount of time between the end of the vertical sync pulse and the start of the next vertical active time. The vertical back porch is measured in lines.

flags

A bitmask of `wfdcfg_flags`. You can use this field to configure settings that are supported by your display driver. Appropriate settings for this flag are likely based on your display and/or bridge specifications.

Setting timing parameters

The timing parameters that Wfdcfg requires are available in, or can be derived from, the product specfication of your display hardware. Below is a table of typical timings based on display resolutions and refresh rates:

Display	pixel_clock_kHz	hpixels	vlines	hsw	vsw	hfp	vfp	hbp	vbp
800x400 @ 60 Hz	29760	800	400	72	10	24	3	96	7
1024x768 @ 60 Hz (CVT)	63500	1024	768	104	4	48	3	152	23
1280x1024 @ 60 Hz (CVT)	109000	1280	1024	128	7	88	3	216	29
1080p @ 60 Hz (1920x1080)	148500	1920	1080	44	5	88	4	148	36
720p @ 60 Hz (1280x720)	74250	1280	720	40	5	110	5	220	20

You'll need to configure these timing parameters, based on your product specification, in your Wfdcfg source (`wfdcfg.c`) within a `mode` structure. The `mode` may also include extensions, but the timing is the most important part. It's possible that in the following circumstances, you may have multiple entries in your mode array:

- when you have more than one physical display connected
- when your display supports multiple modes
- when you're switching between displays, but only have one connected at a time

For example,

```
struct mode {
 const struct wfdcfg_timing timing;
 const struct wfdcfg_keyval *ext_list;
};

static const struct mode sample_timings[] = {
 {
  // 800x480 @ 60 Hz
  .timing = {
   .pixel_clock_kHz =  29760,
   .hpixels =  800, .hfp= 24, .hsw= 72, .hbp= 96,  //  992 total
   .vlines  =  480, .vfp=  3, .vsw= 10, .vbp=  7,  //  500 total
   .flags = WFDCFG_INVERT_HSYNC,
  },
  .ext_list = (const struct wfdcfg_keyval[]){
   { "ext_1_example", .i = 1 },
   { "ext_2_example", .i = 2 },
   { NULL }  // marks end of list
  },
 },
 {
```

```
 // 1024x768 @ 60 Hz (CVT)
 .timing = {
  .pixel_clock_kHz =  63500,
  .hpixels = 1024, .hfp= 48, .hsw=104, .hbp=152,  // 1328 total
  .vlines  =  768, .vfp=  3, .vsw=  4, .vbp= 23,  //  798 total
  .flags = WFDCFG_INVERT_VSYNC,
 },
 .ext_list = NULL,
},
{
 // 1280x1024 @ 60 Hz (CVT)
 .timing = {
  .pixel_clock_kHz = 109000,
  .hpixels = 1280, .hfp= 88, .hsw=128, .hbp=216,  // 1712 total
  .vlines  = 1024, .vfp=  3, .vsw=  7, .vbp= 29,  // 1063 total
  .flags = WFDCFG_INVERT_VSYNC,
 },
 .ext_list = NULL,
},
{
 // 1080p @ 60 Hz (1920x1080)
 .timing = {
  .pixel_clock_kHz = 148500,
  .hpixels = 1920, .hfp= 88, .hsw= 44, .hbp=148,  // 2200 total
  .vlines  = 1080, .vfp=  4, .vsw=  5, .vbp= 36,  // 1125 total
  .flags = 0,
 },
 .ext_list = NULL,
},
{
 // 720p @ 60 Hz (1280x720)
 .timing = {
  .pixel_clock_kHz =  74250,
  .hpixels = 1280, .hfp=110, .hsw= 40, .hbp=220,  // 1650 total
  .vlines  =  720, .vfp=  5, .vsw=  5, .vbp= 20,  //  750 total
  .flags = 0,
 },
 .ext_list = NULL,
},
{
 // marks end of list
 .timing = {.pixel_clock_kHz = 0},
},
};
```

Chapter 6
Updating Wfdcfg source (adding extensions)

You may want to update your Wfdcfg source if your display supports specific options that you want to control.

For most part, other than the timing parameters, there's no need to update the Wfdcfg source. However, some displays on some platforms may support specific options that require extensions on the structures or implementation of callbacks. Not all display drivers support the use of callback functions.

Extensions are added by defining an array of `wfdcfg_keyval` structures. Each extension is identified through a name (`key`) and data that is associated with it.

```
struct wfdcfg_keyval {
    const char *key;    /**< Identifier of extension */
    long i;             /**< Data associated to extension */
    void *p;            /**< Data associated to extension */
};
```

Usually, when there are extensions that are platform-specific, these extensions are declared in a separate header file that is not `wfdcfg.h`. Conventionally, the header file is named `wfdcfg_`*`platform`*`.h` where *platform* refers to the platform of your target hardware.

Declaring extensions

If your display can support additional configuration or settings, then you'll declare these extensions as constants in the source header file. Extensions can be added for:

- devices (`wfdcfg_device`)
- ports(`wfdcfg_port`)
- modes (`wfdcfg_timing`)
- mode lists (`wfdcfg_mode_list`)

Callback functions

An extension that provides the display driver a function to call when appropriate. An extension for a callback function needs declaration of the following:

- the name of the extension
- the prototype of the callback function

For example, the following is a declaration of a callback function extension. This function initializes the port and is to be called when the `wfdcfg_port` is created.

```
/**
 * Port initialisation.  If this port extension exists, the WFD driver
 * will call the given function when the port is created.
 *  .p  (of is a pointer to a function of type wfdcfg_ext_fn_port_init_t
 *      (which returns EOK on success or another errno code on failure)
 *  .i must be zero
 */
#define WFDCFG_EXT_FN_PORT_INIT "port_init"
typedef int (wfdcfg_ext_fn_port_init_t)(struct wfdcfg_port*);
```

Attributes

An extension that provides information for the configuration of the display hardware.

The following are examples of such an extension:

```
/**
 *  The HSP clock frequency in kHz.  This is a device extension.
 *  .p must be NULL
 *  .i gives the clock speed in kHz.
 */
#define WFDCFG_EXT_HSP_CLOCK_KHZ "hsp_clock_kHz"

/**
 * Specifies the output format.  This is a port extension.
 *  .p must be NULL
 *  .i is a value from enum imx5x_output_formats (default: RGB888)
 */
#define WFDCFG_EXT_OUTPUT_FORMAT "output_format"
enum imx5x_output_formats {
    /* 24 bits (use for 18-bit SPWG too) */
    WFDCFG_OUTPUT_FORMAT_RGB888 = 24,

    /* 18 bits (parallel LCD panels, not LVDS) */
    WFDCFG_OUTPUT_FORMAT_RGB666_PACKED = 18,

    /* 18 bits (LVDS) */
    WFDCFG_OUTPUT_FORMAT_RGB666_SPWG18 = 19,

    /* 16 bits (parallel LCD panels) */
    WFDCFG_OUTPUT_FORMAT_RGB565 = 16,
};
```

Defining extensions

Definitions of the extensions are usually added to the source file `wfdcfg.c`. However, you can define them where you see fit, as long as you maintain binary compatibility.

It's generally expected that there are separate lists for the different types of extensions. That is, you'll have one list for each of device, port, mode, and mode list extensions.

The following are examples of different extension definitions:

```
static const struct wfdcfg_keyval device_exts[] = {
    { WFDCFG_EXT_HSP_CLOCK_KHZ, .i = 200000 },
    { NULL },   // marks end of list
};
```

```
static int port_init(struct wfdcfg_port*);

static const struct wfdcfg_keyval port_exts[] = {
    { WFDCFG_EXT_FN_PORT_INIT, .p = WFDCFG_FNPTR(
            &port_init, wfdcfg_ext_fn_port_init_t*) },
    { NULL },   // marks end of list
};

static int
port_init(struct wfdcfg_port *port)
{
    (void)port;
    return EOK;
}

static const struct mode modes[] = {
    {
        // 800x480 @ 60.49 Hz
        .timing = {
            .pixel_clock_kHz = 32264,
            .hpixels = 800, .hfp= 60, .hsw=124, .hbp= 32,  // 1016 total
            .vlines  = 480, .vfp= 33, .vsw=  2, .vbp= 10,  //  525 total
            .flags = WFDCFG_INVERT_VSYNC | WFDCFG_INVERT_HSYNC,
            },
        .ext_list = NULL,
    },
    { .timing = { .pixel_clock_kHz = 0 } }  // marks end of list
};
```

Chapter 7
Building the Wfdcfg library

After extracting the files from the Wfdcfg archive and modifying your source, you're ready to rebuild the Wfdcfg library.

Remember that the directory where you extracted the Wfdcfg archive is referred to as `wfdcfg_working_dir`.

Building from an existing stage area

If you've already configured a stage area, then:

1. From the command line on your development host, go to the directory where you unzipped your Wfdcfg archive.

   ```
   cd wfdcfg_working_dir
   ```

2. Ensure your environment is clean.

   ```
   make clean
   ```

3. Use the `make` command to build.

   ```
   make hinstall install
   ```

   Your stage directory should now be populated with the library that you've just built.

Building without an existing stage area

A stage area is a directory location that mimics the local installation path(s) that you would find under the ***QNX_TARGET*** variable of your development host. The stage area is built up based on the content from the tree when `make hinstall` (headers) or `make install` (binaries) is performed.

The stage area directory path is selected by setting the ***INSTALL_ROOT_nto*** makefile variable to the base path where your headers and libraries are to be installed. You also need to set the ***USE_INSTALL_ROOT*** macro. This macro tells the makefiles to search the ***INSTALL_ROOT_nto*** directory tree when the compiler and linker are seaching for headers and libraries. It is cumbersome to set these values each time you perform an installation, so the build environment facilitates the setting of these variables through the use of a single override makefile that is specified using the `QCONF_OVERRIDE` environment variable.

You can use the `stage-sh` script that is included in the Wfdcfg archive to create a stage area for you.

1. From the command line, go to your the directory where you unzipped your Wfdcfg archive.

   ```
   cd wfdcfg_working_dir
   ```

2. Run the provided script to create a stage area for building your source.

   ```
   stage-sh
   ```

3. From within the shell created by `stage-sh`, ensure your environment is clean.

   ```
   make clean
   ```

4. From within the shell created by `stage-sh`, use the `make` command to build.

   ```
   make hinstall install
   ```

   The stage directory that was created by running the `stage-sh` script should now be populated with the library that you've just built.

For more information on building OS source, refer to *Building Embedded Systems*, or to our community forums from our website, www.qnx.com.

Chapter 8
Updating your target

After you've completed updating and building your Wfdcfg library, you are ready to rebuild your QNX IFS and transfer it to your target.

Updating your buildfile

1. Navigate to the buildfile for your BSP under your `bsp_working_dir`:

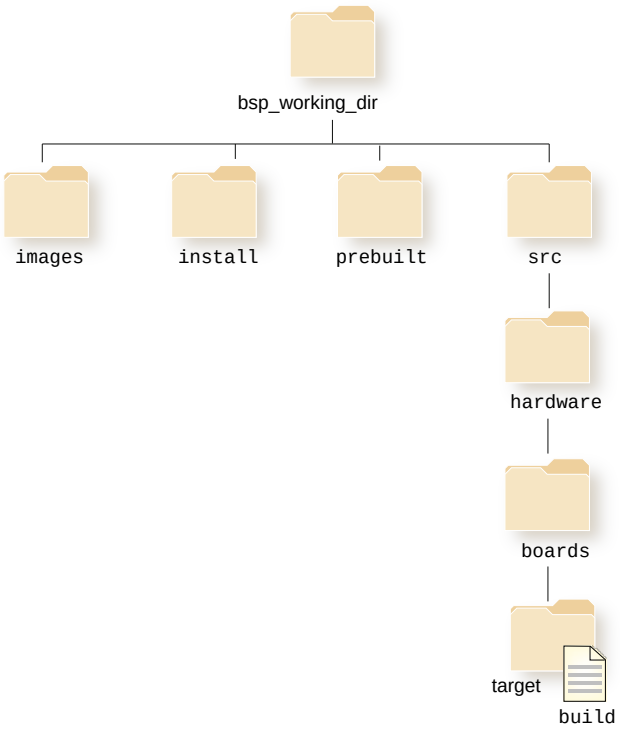


Figure 3: `buildfile`

2. Add the Wfdcfg library to the buildfile.

```
/usr/lib/graphics/platform/libwfdcfg-sample.so=graphics/platform/libwfdcfg-sample.so
```

   Acceptable paths for your built Wfdcfg library are:

   - ***$GRAPHICS_ROOT***
   - Your default library search path

For more information on changing buildfiles, refer to *Building Embedded Systems*, or to the BSP User Guide. The BSP User Guide that's specific to your target is available from from our website, www.qnx.com.

Building your QNX IFS

1. Go to the root directory for your BSP (`bsp_working_dir`).
2. Ensure your environment is clean.

   ```
   make clean
   ```

3. Use the `make` to build your QNX IFS.

   ```
   make
   ```

For more information on building your QNX IFS, refer to *Building Embedded Systems*, or to the BSP User Guide. The BSP User Guide that's specific to your target is available from from our website, www.qnx.com.

Transferring the QNX IFS to your target

From *Building Embedded Systems*, follow the instructions on how to transfer your QNX IFS to your target.

Chapter 9
Configuring Screen for your display

You need to configure your display in `graphics.conf` and restart Screen.

Configuring `wfd device` section

In your configuration file (`graphics.conf`), on your target, you need to specify the Wfdcfg library (or libraries) and any parameters related to your display device. In the `wfd device` section, ensure that you have the appropriate libraries specified for your display. For example,

```
begin wfd device 1
  wfd-dlls = libwfdcfg-imx6x-okaya.so libimx6xCSCgamma-generic.so libWFDimx6x.so
  grpx0 = lcd
  grpx1 = hdmi
  grpx2 = hdmi
  video-layer0 = lcd
  video-layer1 = hdmi
end wfd device
```

You need to configure one `wfd device` section for each display you are using.

For more information on configuring Screen, refer to the *Screen Developer's Guide*.

Restarting Screen

Ensure the following:

- Your target hardware is running QNX Neutrino RTOS.
- On your target, you can run a shell and commands such as `pidin`.

To apply your new display configuration:

1. On your target, from the command line, stop `screen` by using the following command:

   ```
   slay screen
   ```

   You can verify that the `screen` process is no longer running by using the following command:

   ```
   pidin ar
   ```

2. If not already set, ensure that your ***GRAPHICS_ROOT*** and ***LD_LIBRARY_PATH*** are set to correct paths. For more information on setting these environment variables, refer to the *Screen Developer's Guide*.

3. Restart `screen`.

   ```
   screen
   ```

4. Verify that there were no warnings generated from your new configuration by using the following command:

   ```
   sloginfo
   ```

5. Use any of the available sample Screen applications to verify that your display has been correctly integrated for use with your target. For example:

   ```
   sw-vsync
   ```

For more information on applying your Screen, configuration, refer to the *Screen Developer's Guide*.

Chapter 10
OpenWF Display Configuration Library Reference

The Wfdcfg library provides the modes and attributes of your display hardware to your display driver and to Screen. Your display driver (WFD driver) is the primary user of the Wfdcfg library, but the composition manager component in Screen also uses the modes and attributes from the Wfdcfg library.

Definitions in *wfdcfg.h*

Preprocessor macro definitions for the wfdcfg.h header file in the library.

Definitions:

`#define WFDCFG_EXT_PHYS_SIZE_MM "phys_size_mm"`

A port extension that describes the physical size (in millimetres) of the display port.

Members of wfdcfg_keyval structure used for this extension are used as follows:

- `p:` pointer to an array of float (float size[2]={width, height};)
- `i:` set to 0

`#define WFDCFG_EXT_FN_SET_POWER_MODE "set_power_mode"`

A port extension function that's called to set the power mode on a port.

Members of wfdcfg_keyval structure used for this extension are used as follows:

- `p:` pointer to the function of type wfdcfg_ext_fn_set_power_mode_t
- `i:` set to 0

Library:

`libwfdcfg`

WFDCFG_FNPTR(FN, TYP)

Validate the type of a function pointer at compile time.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

WFDCFG_FNPTR(FN,TYP)
```

Arguments:

FN

A function pointer that you're validating

TYP

A function type that you're validating your function pointer against

Library:

`libwfdcfg`

Description:

This macro is used to validate that whether the specified function pointer (*FN*) is the same type as the specified function type (*TYP*).

Returns:

The function pointer FN, if it is compatible with the function type specified; 0 otherwise.

wfdcfg_device

Opaque data type representing an OpenWF device

Synopsis:

```
struct wfdcfg_device;
```

Library:

`libwfdcfg`

Description:

This device is an abstraction of a display controller that supports one or more ports. The device may be associated with device-extensions.

wfdcfg_device_create()

Create a wfdcfg device.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

int wfdcfg_device_create(struct wfdcfg_device **device,
                         int deviceid,
                         const struct wfdcfg_keyval *opts)
```

Arguments:

device

A handle to the device that is to be created

deviceid

The identification number of the OpenWF device

opts

An array of optional parameters that is terminated by .key=NULL

Library:

`libwfdcfg`

Description:

This function creates one device. You must create at least one device. Failure to create a device results in the OpenWF display driver reporting an error.

Returns:

0 if device was successfully created; `*device` is set to an opaque pointer. A code from errno.h if device failed to be created; `*device` remains unchanged. Possible error codes include:

- `ENOMEM`: Unable to allocate a device
- `ENOENT`: Invalid/Unknown device ID

wfdcfg_device_destroy()

Destroy a wfdcfg device.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

void wfdcfg_device_destroy(struct wfdcfg_device *device)
```

Arguments:

device

A handle to the device to be destroyed; if NULL, no action will be taken

Library:

`libwfdcfg`

Description:

Memory allocated that was allocated by create wfdcfg_device_create() is released. The device's extension pointers are not valid after calling this function.

Returns:

Nothing.

wfdcfg_device_get_extension()

Retrieve an extension identified by a key (string) from a device.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

const struct wfdcfg_keyval* wfdcfg_device_get_extension(
            const struct wfdcfg_device *device,
            const char *key)
```

Arguments:

device

A handle to the device whose extension(s) you are retrieving

key

Identifier of extension to retrieve

Library:

`libwfdcfg`

Description:

The extension is valid between the time you create and destroy the device.

Returns:

Pointer to wfdcfg_keyval if the extension was found; NULL if the extension was not found. It's considered acceptable for a device to have no extensions.

wfdcfg_ext_fn_set_power_mode_t

A port extension function that's called to set the power mode on a port.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

typedef int( wfdcfg_ext_fn_set_power_mode_t)
               (struct wfdcfg_port *port,
                enum wfdcfg_power_mode power_mode);
```

Arguments:

port

A handle to the display port on which the power mode is to be set

power_mode

The power mode (of type wfdcfg_power_mode) that the display port is to be set to

Library:

`libwfdcfg`

Description:

Returns:

A code from @c errno.h; possible codes include:

- `EINVAL:` when a invalid power mode is passed
- `EOK:` on success

wfdcfg_flags

Flags used in the `flags` field of `wfdcfg_timing` structure.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>


 enum wfdcfg_flags{
      WFDCFG_INVERT_HSYNC =_b(0)
      WFDCFG_INVERT_VSYNC =_b(1)
      WFDCFG_INVERT_DATA_EN =_b(2)
      WFDCFG_INVERT_CLOCK =_b(3)
      WFDCFG_INVERT_DATA =_b(4)
      WFDCFG_INVERT_HV_SYNC_RF =_b(5)
      WFDCFG_INTERLACE =_b(8)
      WFDCFG_DOUBLESCAN =_b(9)
      WFDCFG_DOUBLECLOCK =_b(10)
      WFDCFG_PREFERRED =_b(31)
};
```

Data:

WFDCFG_INVERT_HSYNC

Use an active-high horizontal sync pulse.

WFDCFG_INVERT_VSYNC

Use an active-high vertical sync pulse.

WFDCFG_INVERT_DATA_EN

Invert the "data enable" signal.

WFDCFG_INVERT_CLOCK

Invert the pixel clock signal.

WFDCFG_INVERT_DATA

Invert data.

WFDCFG_INVERT_HV_SYNC_RF

Drive HSYNC and VSYNC on the opposite edge of the pixel clock.

WFDCFG_INTERLACE

Use interlacing.

WFDCFG_DOUBLESCAN

Enable scanline doubling.

WFDCFG_DOUBLECLOCK

Use CEA-861-D, double clock modes.

WFDCFG_PREFERRED

Use as default mode.

Library:

`libwfdcfg`

Description:

wfdcfg_keyval

Array(s) of this structure are used to allow for extensions.

Synopsis:

```
struct wfdcfg_keyval {
    const char * key ;
    long i ;
    void * p ;
};
```

Data:

const char * key

Identifier of extension.

long i

Data associated with extension.

void * p

Data associated with extension.

Library:

`libwfdcfg`

Description:

Extensions can exist on each of the following:

- device: access these via *wfdcfg_device_get_extension()* (p. 39).
- port: access these via *wfdcfg_port_get_extension()* (p. 53).
- mode: access these via *wfdcfg_mode_get_extension()* (p. 44).
- mode_list: access these via *wfdcfg_mode_list_get_extension()*.

Several Wfdcfg library functions take this structure as an optional argument; certain drivers pass data to the Wfdcfg library using this interface.

The `i` and `p` members of `this` structure depend on the key (when part of an array, an element with `key==NULL` marks the end of that array). Unused `i` and/or `p` fields should be set to 0 or NULL.

wfdcfg_mode_get_extension()

Retrieve an extension from the specified mode.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

const struct wfdcfg_keyval* wfdcfg_mode_get_extension(
            const struct wfdcfg_timing *mode,
            const char *key)
```

Arguments:

mode

A handle to the mode (timing) whose extension(s) you are retrieving

key

Identifier of extension to retrieve

Library:

`libwfdcfg`

Description:

Returns:

Pointer to wfdcfg_keyval if the extension was found; NULL if the extension was not found. It's considered acceptable for a list to have no extensions.

wfdcfg_mode_list

Opaque data type representing a list of video modes

Synopsis:

```
struct wfdcfg_mode_list;
```

Library:

`libwfdcfg`

Description:

A mode is a set of attributes and extensions that can be set on a display port.

wfdcfg_mode_list_create()

Create a list of video modes associated with specified port.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

int wfdcfg_mode_list_create(struct wfdcfg_mode_list **list,
                            const struct wfdcfg_port *port,
                            const struct wfdcfg_keyval *opts)
```

Arguments:

list

A handle to the list to be created

port

The port associated with the list to be created

opts

An array of optional parameters that is terminated by .key=NULL

Library:

`libwfdcfg`

Description:

Once created, use wfdcfg_mode_list_get_next() to retrieve mode (timing) entries in the list.

Returns:

0 if port was successfully created; `*list` is set to an opaque pointer. A code from errno.h if device failed to be created; `*list` remains unchanged. Possible error codes include:

- `ENOMEM`: Unable to allocate a list
- `ENOENT`: Invalid port

wfdcfg_mode_list_destroy()

Destroy a list of video modes.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

void wfdcfg_mode_list_destroy(struct wfdcfg_mode_list *list)
```

Arguments:

list

A handle to the list to be destroyed; if NULL, no action will be taken.

Library:

`libwfdcfg`

Description:

Memory allocated that was allocated by create wfdcfg_mode_list_create() is released. The list's extension pointers are not valid after calling this function.

Returns:

Nothing.

wfdcfg_mode_list_get_next()

Retrieve a mode (timing) from the specified list of video modes.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

const struct wfdcfg_timing* wfdcfg_mode_list_get_next(
            const struct wfdcfg_mode_list *list,
            const struct wfdcfg_timing *prev_timing)
```

Arguments:

list

A handle to the list to retrieve the mode from

prev_timing

A handle to the mode (timing) in the list that precedes the one to be retrieved

Library:

`libwfdcfg`

Description:

Timing is mandatory component of a mode. The mode may also include extensions. If `prev_timing` is NULL, this function returns the first mode (timing) in the specified list. Otherwise, this function returns the next mode (timing) in the list after `prev_timing`.

Returns:

A pointer to the mode (wfdcfg_timing) in the list that follows the argument `prev_timing`; a return value of NULL indicates that the end of the list has been reached.

wfdcfg_port

Opaque data type representing an OpenWF display port

Synopsis:

```
struct wfdcfg_port;
```

Library:

```
libwfdcfg
```

Description:

This port is usually associated with an ID and a list of modes that include timing and optional associated mode-extensions.

wfdcfg_port_create()

Create a wfdcfg port.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

int wfdcfg_port_create(struct wfdcfg_port **port,
                       const struct wfdcfg_device *device,
                       int portid,
                       const struct wfdcfg_keyval *opts)
```

Arguments:

port

A handle to the port that is to be created

device

A handle to the device that is associated with the port to be created

portid

The identification number of the OpenWFD port

opts

An array of optional parameters that is terminated by .key=NULL

Library:

`libwfdcfg`

Description:

There must be at least one port created. Otherwise, the OpenWF display driver will report an error.

Returns:

0 if port was successfully created; `*port` is set to an opaque pointer. A code from errno.h if device failed to be created; `*port` remains unchanged. Possible error codes include:

- `ENOMEM`: Unable to allocate a port
- `ENOENT`: Invalid/Unknown port ID

wfdcfg_port_destroy()

Destroy a `wfdcfg` port.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

void wfdcfg_port_destroy(struct wfdcfg_port *port)
```

Arguments:

port

A handle to the device to be destroyed; if NULL, no action will be taken

Library:

`libwfdcfg`

Description:

Memory allocated that was allocated by create wfdcfg_port_create() is released. The port's extension pointers are not valid after calling this function.

Returns:

Nothing.

wfdcfg_port_get_extension()

Retrieve an extension identified by a key (string) from a port.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>

const struct wfdcfg_keyval* wfdcfg_port_get_extension(
            const struct wfdcfg_port *port,
            const char *key)
```

Arguments:

port

A handle to the port whose extension(s) you are retrieving

key

Identifier of the extension to retrieve

Library:

`libwfdcfg`

Description:

The extension is valid between the time you create and destroy the port.

Returns:

Pointer to wfdcfg_keyval if the extension was found; NULL if the extension was not found. It's considered acceptable for a port to have no extensions.

wfdcfg_power_mode

Power modes for display.

Synopsis:

```
#include <wfdqnx/wfdcfg.h>


 enum wfdcfg_power_mode{
      WFDCFG_POWER_MODE_OFF = 0x7680
      WFDCFG_POWER_MODE_SUSPEND = 0x7681
      WFDCFG_POWER_MODE_LIMITED_USE = 0x7682
      WFDCFG_POWER_MODE_ON = 0x7683
};
```

Data:

WFDCFG_POWER_MODE_OFF

No power - frames lost.

WFDCFG_POWER_MODE_SUSPEND

Faster recovery than WFDCFG_POWER_MODE_OFF.

WFDCFG_POWER_MODE_LIMITED_USE

Frames maintained in hardware.

WFDCFG_POWER_MODE_ON

Fully operational.

Library:

`libwfdcfg`

Description:

Some power modes may not be possible for your specific display hardware. Recovery time to `WFDCFG_POWER_MODE_ON` decreases from `WFDCFG_POWER_MODE_OFF` to `WFDCFG_POWER_MODE_SUSPEND` to `WFDCFG_POWER_MODE_LIMITED_USE` and the power consumption increases.

wfdcfg_timing

Structure that describes the video timing parameters for the display driver settings.

Synopsis:

```
struct wfdcfg_timing {
    _Uint32t pixel_clock_kHz ;
    _Uint32t hpixels ;
    _Uint32t vlines ;
    _Uint16t hsw ;
    _Uint16t vsw ;
    _Uint16t hfp ;
    _Uint16t vfp ;
    _Uint16t hbp ;
    _Uint16t vbp ;
    _Uint32t flags ;
};
```

Data:

_Uint32t pixel_clock_kHz

Frequency (in kHz) that pixels are transmitted at.

_Uint32t hpixels

Width (in pixels) of the display.

_Uint32t vlines

Height (in lines) of the display.

_Uint16t hsw

Width (in pixels) of horizontal sync pulse.

_Uint16t vsw

Width (in lines) of vertical sync pulse.

_Uint16t hfp

Horizontal front porch (in pixels)

_Uint16t vfp

Vertical front porch (in lines)

_Uint16t hbp

Horizontal back porch (in pixels)

_Uint16t vbp

Vertical back porch (in lines)

_Uint32t flags

Bitmask of wfdcfg_flags values.

Library:

`libwfdcfg`