SharpMedia Shell Design

© 2007 Žiga Osolin – SharpMedia project, document version: DRAFT

Table Of Contents

[About 2](#_Toc168026883)

[Goals 2](#_Toc168026884)

[Concepts 3](#_Toc168026885)

[Interfaces 4](#_Toc168026886)

[IWindowManager 4](#_Toc168026887)

[IDesktop 4](#_Toc168026888)

[ICursor 4](#_Toc168026889)

[IWindow 4](#_Toc168026890)

[Process communications 4](#_Toc168026891)

[Implementation notes 4](#_Toc168026892)

[Invalidate Scenarios 4](#_Toc168026893)

[Usages 4](#_Toc168026894)

# About

Windows is a service and library, based on vector graphics. It implements swap chain sharing by allowing processes to create windows on desktop.

# Goals

Service should contain the following:

* Window manager;
* Window effects (rotation, blending, minimization/maximization, …);
* Desktop (multiple desktops);
* Easy creation of windows, easy removal;
* Fast rendering, dirty sections, invalidation;
* Event processing and dispatching;
* Easy to use client side classes.

Unresolved:

* Desktop manipulation (start, quick lunch);
* Desktop icon functionality (a link, something more sophisticated?);
* Library API and classes.

# Concepts

Windows service is based on certain GUI theme. It works using render target sharing technique introduced by Graphics library. A window is defined by rectangle region (shape can be changed using blending techniques, described later). Windows service is always the owner of device (has swap chain).

Desktop

Window Common Part (close button, magnify, fullscreen)

Window region

The desktop is fully controlled by Window Services. This means that it is rendered to swap chain by this service automatically. It also controls the following attributes of internal window:

* Windows focus and z-order;
* Window Common Part (fires close, magnify, full screen events);
* Window resizing (and fires event for render target resizing);
* Window invalidation, re-rendering;
* Window post effects.

The Window Region is rendered by process that owns that window to a special, shared render target. This render target can be use as a texture source for rendering in swap chain. What is more, additional effects may be applied to such windows (rotation, blending). Note that Windows Service does not require the process that contains window rendering a window because all required contents already exist in the specified render target.

# Service Interfaces

## IWindowManager

A window manager is the service type. It can be used to create new windows, delete existing, invalidate some windows (usually only those that you have access to), configure the desktops etc. It can be obtained through the OS service API, as any other service.

On the client side, you can inspect different windows, request termination (if you have privileges). You can also configure window effects through this interface[[1]](#footnote-2).

## IDesktop

A desktop in SharpMedia is not only one. You can have as many desktops as you want. Desktop is never bound to certain directory; it is simply a (serializable) list of icons and other configurations of desktop. Through this interface, you can:

* Create a new icon;
* Delete an existing icon;

Desktops support inheritance; this means that some desktop can have all icons of parent desktops plus its own.

## DesktopIcon

A desktop icon is a simple element on desktop. It is a link to some node. It also has an icon link.

## IPointer

A pointer is usually bound to a mouse, but not necessarily. A pointer is somewhat different from mouse; it has its inside desktop position. Clicking any of keys that are bound as pointer events will add some sort of processing that depends on cursor position. It may even affect the windows.

## IWindowBackend

This interface can be obtained through **IWindowManager**. It is used to send data to Window Service. However, many times, window service needs to notify the window of some event (mouse click etc.). This is why **IWindow** must be registered in window backend to allow back communication.

A window backend can configure the window rendering. It can only change title of window and available controls (close, minimize, maximize, toggle full screen) but not the window’s size or position. Furthermore, it can force window to render itself (this should be done when something is changed). A render target can also be changed.

Control over window effects is not exposed to window. These effects are global and can be only configured through **IWindowManager**.

## IWindow

A window is implemented by client side. A user can use custom window event trigger, however, we suggest accessing windows through the **ClientWindow** (Windows Library Classes), which implements both.

# Library Classes

## ClientWindow

Client window implements **IWindow** interface and allows both way service<->process communication. It has helper methods to add GUI hierarchy to it. Besides all that, it implements (if not disabled) automatic render target resizing.

# Implementation notes

## Invalidate Scenarios

On window invalidation, we first request window re-rendering. We then wait for window event for some time (e.g. 1 second) and if there is none, we know that window is not responding. In such scenarios, “white window” is used and we are still pending for the rewrite of current window. We also notify the system that a process is not responding.

## Window Effects

Window effects are almost free and make the windows more attractive. Instead of just copying data on top of swap chain, we can mix them quite freely. Windows blending is one of such effects, where we draw windows in correct order and you can see through each window.

There are also animation effects of windows. For example, window can do some animation when being minimized or maximized. Furthermore, we can draw windows in 3D. For example, there is sphere mapping effect, or 3D window preview effect.

# Usages

1. More about effects in Implementation Notes. [↑](#footnote-ref-2)