SharpMedia Resources Design

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

Table Of Contents

[About 2](#_Toc167623225)

[Goals 2](#_Toc167623226)

[Interfaces 2](#_Toc167623227)

[IReference 2](#_Toc167623228)

[IPersistencyInfo 3](#_Toc167623229)

[IResource 3](#_Toc167623230)

[IResourceLoader 3](#_Toc167623231)

[Special Issues 4](#_Toc167623232)

[Reference communication (shared resource scenarios) 4](#_Toc167623233)

[Loader communication (remote resource scenarios) 4](#_Toc167623234)

[Resource multi-storage scenarios 4](#_Toc167623235)

[Resource mixing/composition 4](#_Toc167623236)

[Delayed loading 5](#_Toc167623237)

[Usage Case: Referencing Images 6](#_Toc167623238)

# About

This design is about resources, their creating, referencing, handling and deletion. We feel that resources are very important part. This specific document is about resource handling in general. The main resource interfaces should reside in SharpMedia library, implementations reside in assemblies that introduce new resource types.

# Goals

The following goals should be utilized:

* Easy resource loading;
* Automatic persistency for some resources;
* Resource mixing at runtime;
* Can point to not yet created resources (e.g. resources to be mixed);
* Can point to remote resources or not-loaded resources;
* Cache resources and share them in very generic way;
* Copy-on-write scenarios for some particular resources;
* Specific resource handling for different resource types.

# Interfaces

Interfaces describe the abstraction introduced by resource system.

## IReference

A reference is an interface that can point to any resource. Different resources can be accessed through **IReference** interface; however access through specialized reference class is preferred. Specialized class can offset additional properties and only through class’ methods/properties can the actual contents of reference be accessed. Note that reference access is **shared** way to accessing resources; e.g. multiple references can exist on the same resource. Resources can be modified using copy-on-write or locking techniques. Some reference types also support merging.

This interface can reference any resource, is it on remote machine or even not yet created. An example of that is pointing to uncreated (mixed) image on another machine. The resource is transferred only when requested. This means that you can mix resources that all reside on other machine there and get only the result. Furthermore, some operations use references, not images, and can execute on best CPU based on locality of resources (of course, the CPU must be available for that).

A reference can be through as “smart pointer”. Access to number of references is also available to any reference through **ReferenceCount** property.

The reference contains information about resource’s location (there can be several locations at once). It also contains **IPersistencyInfo**, if either loaded from persistent medium or if being prepared to reside there when persisted.

## IPersistencyInfo

Persistency information contains address (from process’ view) of node. It also contains method **Persist** that writes resource only if:

* The resource was changed;
* The resource changed its address (in such case, previous address can be cleaned).

Persistency can also be configured to affect several addresses on many computers. This way, all remote computer’s resources are updated when you update you texture. Additionally, you can write the resource as new version, thus still leaving previous version available. You can also disconnect the reference from the resource (by cloning the resource and not letting others know).

Persistency information types should be contained in specific namespaces; Database persistency in **SharpMedia.Database** namespace, other persistency types elsewhere.

## IResource

A resource has some events and properties that allow it to be referenced by a reference. Each resource implements **ICachable** interface and in essence, every resource can be cached (loader can cache resources). Besides that, resource implements the location property (having full location, which is the name of computer, name of the process and inside process addressing[[1]](#footnote-2)), reference count[[2]](#footnote-3) and changed property. Using this property, we can resolve reference resource locality and cost of transporting.

References can register delegates on changed property.

## IResourceLoader

A resource loader is abstract class and is in essence, a service. A resource loader can load resources from persistent storage, is required to **share** resources (two references that reference same resource at the same time *must* reference the same resource, but can be *split* later in process) and can cache resources (even if no more references, resource can stay alive a bit longer). At the same time, resources can be *unloaded* even before all references are disposed.

A resource loader must also be capable of communicating with other resource loaders of the same type on other machines. To support that, they must be properly configured and this is usually done in groupware scenarios. If certain resource cannot be located on current machine, a search is expanded to all remote machines (that is, if reference and configuration permits that).

# Special Issues

Special issues and scenarios are handled here so resources, loaders, references are properly coded.

## Reference communication (shared resource scenarios)

A direct reference-reference communication is not possible, only reference-resource and resource-reference communication is possible. At any given time, a number of references point to resource can be obtained. This is possible even if resource does not exist yet, because resource loader must create a small *resource proxy* class that tracks references (and is filled upon actually loading/creating/transporting the resource).

Reference also need communication when they want to obtain locks (e.g. to obtain write access without copying) and when events like loaded, unloaded, persisted are fired. All communication is done using events on proxy or resource.

## Loader communication (remote resource scenarios)

Remote references can be obtained in many ways. One way is that the reference is transferred using serialization to another application domain. Addressing of such reference does not change (however, methods are provided to re-link reference based on locality).

The other way to obtain resource is to specifically address resource remotely, e.g. connect to resource service on another computer.

The third and probably most useful way is to connect through local computer’s loader to another computer. This way, resource loader must be properly configured to locate resources on another computer. There are two possible scenarios:

* When loader locates remote resource, it is assumed to must exist on local hardware and resource is copied to local database at demand (that is, if resource is “shared”);
* Resource can be accessed using fully remote services.

## Resource multi-storage scenarios

Resources can exist in multiple storages at once. The loader is **not required** to enumerate all locations of resource; however resources that reside in RAM or Device memory must have those locations marked. At the same time, loader must never lie where the resource is.

When resource exists in multiple storages, the specific references must handle specific accesses to such resources. For example, texture can reside in RAM or GPU memory. The reference addresses this issue by allowing different accessing schemes for RAM (image) or GPU (textures) memories.

## Resource mixing/composition

Resource mixing is a powerful feature introduced by resource system. A resource reference is available before the resource is mixed. Mixing is done on demand, when first accessed or when explicitly requested.

Resource location is the **worst** location of any of the components that are needed to mix the resource.

## Delayed loading

Delayed loading is also a powerful feature. This means that you can construct the whole structure of you program and load only resources that are needed. There is almost no memory penalty for this.

# Usage Case: Referencing Images

This usage case demonstrates how easy it is to load images and reference them. You can fully inspect them before using them (e.g. query their types, compatible hardware formats …).

public void Reference()

{

// References an image, as texture.

ImageRef reference = new ImageRef("/MyGame/Textures/Woods/Cellar",/\* some other flags \*/);

// We can access it's current location(s).

Placement locations = reference.Locations;

// We check if it is already in device memory.

if ((locations & Placement.DeviceMemory) != 0)

{

// Obtains as 2D texture (if not 2D, it throws an exception). Texture

// can be already prepared if data already in Device memory.

Texture2D image = reference.Texture2D;

// Work with image ...

}

else

{

// This call will load the data into texture, usually

// directly in GPU memory.

Texture2D image = reference.Texture2D;

// Do something else ...

}

}

1. Resources that reside in service section and are not shared (most of them are not), belong to process that controls them, not service process. [↑](#footnote-ref-2)
2. The same as defined for references. [↑](#footnote-ref-3)