



# **Marlin Broadband Architecture Overview**

**for Marlin Adopters**

Draft of November 5, 2007

---

**Notice**

This documentation is provided to you pursuant to your agreement with Intertrust Technologies Corporation (“Intertrust”). This documentation may be used only in accordance with the terms of the agreement.

Copyright (c) 2006-2007 by Intertrust. All rights reserved.

# Table of Contents

|  |           |
|--|-----------|
| <b>1 Introduction.....</b>                                   | <b>5</b>  |
| <b>2 Audience .....</b>                                      | <b>5</b>  |
| <b>3 Marlin DRM Architecture Big Picture.....</b>            | <b>5</b>  |
| 3.1 Key Components .....                                     | 6         |
| 3.1.1 Web Store .....  | 6         |
| 3.1.2 Client Application .....                               | 6         |
| 3.1.3 Marlin Client .....                                    | 7         |
| 3.1.4 Marlin Server.....                                     | 7         |
| 3.2 Basic Interactions .....                                 | 7         |
| <b>4 Marlin DRM Architecture More Detailed Picture .....</b> | <b>9</b>  |
| 4.1 Optional Components .....                                | 10        |
| 4.1.1 Back Office.....                                       | 10        |
| 4.1.2 Content Server .....                                   | 10        |
| 4.2 Incorporated Technologies .....                          | 10        |
| 4.2.1 Octopus DRM.....                                       | 10        |
| 4.2.2 NEMO Framework .....                                   | 11        |
| 4.3 Communication Protocols.....                             | 11        |
| 4.4 Key Players.....   | 13        |
| 4.4.1 Users.....   | 13        |
| 4.4.2 Service Providers .....                                | 13        |
| 4.4.3 Content Providers .....                                | 13        |
| 4.5 Key Objects .....  | 14        |
| 4.5.1 Tokens .....   | 14        |
| 4.5.2 Content and Licenses .....                             | 15        |
| 4.5.3 Nodes and Links .....                                  | 15        |
| 4.5.4 Agents .....   | 17        |
| 4.6 Key Concepts .....                                       | 17        |
| 4.6.1 Registration and Domains.....                          | 17        |
| 4.6.2 Deregistration.....                                    | 17        |
| 4.7 Marlin Services.....                                     | 18        |
| 4.7.1 Services Initiated by Action Tokens.....               | 18        |
| 4.7.2 Security Data Services.....                            | 19        |
| <b>5 Sample Complete DRM System Interaction .....</b>        | <b>19</b> |
| <b>6 Client Application Usage of Sushi SDK API.....</b>      | <b>21</b> |
| 6.1 Initialization and Specifying a Callback Listener .....  | 21        |
| 6.2 Action Token Processing.....                             | 23        |
| 6.3 Checking a License and Playing Content.....              | 24        |
| <b>7 For Further Information.....</b>                        | <b>24</b> |
| 7.1 White Papers .....                                       | 24        |
| 7.2 Marlin Specifications.....                               | 25        |
| 7.3 NEMO Specifications.....                                 | 25        |
| 7.4 Octopus Specifications .....                             | 25        |

|          |                               |           |
|----------|-------------------------------|-----------|
| 7.5      | Sushi SDK Documentation ..... | 25        |
| 7.6      | BlueWhale Documentation ..... | 25        |
| <b>8</b> | <b>Glossary .....</b>         | <b>26</b> |

# 1 Introduction

The Marlin architecture specifies technologies for building copy protection and digital rights management (DRM) into consumer devices and services in a manner that is friendly to end users. Marlin technology allows users to acquire content through multiple distribution channels and to access it on any device that is part of their home domain. In order to accomplish this, Marlin defines both client capabilities and a service architecture so that the capabilities of consumer electronics devices can be powerfully enhanced by services provided over both local and wide-area networks.

This document provides an overview of the architecture for a complete system that includes utilization of Marlin protocols. It describes the key components, their responsibilities, and how they interact. It indicates where Marlin and other protocols are used. It also discusses example components that are already built, the Sushi Marlin Client SDK and the BlueWhale Marlin Broadband Server. Since it will be common for these components to be utilized in the near future, this document provides not only general information, but specific information relevant for using these components.

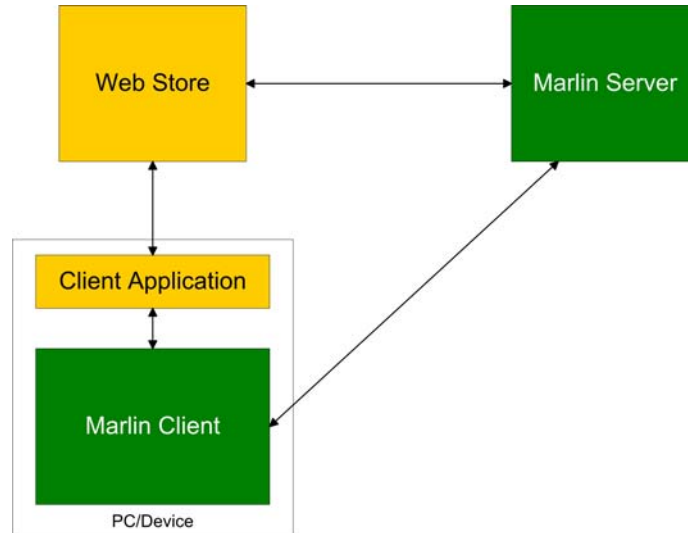
After Section 2 briefly describes the audience for this document, Section 3 provides a high-level “big picture” illustration of the Marlin DRM architecture and describes the components and their interactions. Section 4 then presents a more complete picture and documents the additional components, the technologies and communication protocols utilized in a Marlin DRM System, and key players and objects involved in DRM interactions. Next, Section 5 illustrates and describes all the steps that may be involved in a complete Marlin DRM System interaction for obtaining content and a license. Section 6 then provides an introduction to some of the main Sushi SDK API methods for common operations, including those referenced in Section 5. Finally, Section 7 lists additional documentation, and Section 8 provides a detailed glossary of terms utilized in this document.

## 2 Audience

This document is intended for Marlin adopters, that is, software architects and developers who will write applications for obtaining and rendering content.

## 3 Marlin DRM Architecture Big Picture

Figure 1 provides a high-level simplified illustration of the main components involved in a typical Marlin DRM system. The components and their interactions are described in the immediately following sections. More detailed illustrations and architecture descriptions are provided in §4.



**Figure 1 Basic Marlin DRM architecture**

## 3.1 Key Components

The components shown in Figure 1 are the following:

### 3.1.1 Web Store

The Web Store is the front end for all the operations that provide information and choices for the end user. The Web Store supplies content navigation and selection options. No particular format and protocol are required for these operations. Commonly, web pages will be utilized.

A basic example Web Store implementation is the Seahorse Web Store being developed by Intertrust.

### 3.1.2 Client Application

A user runs a custom Client Application on a PC or another device in order to connect to the Web Store, obtain digital media content (such as videos or music), and play content, if allowed by the content license. The Client Application provides the UI for interacting with the user, integrates with the Web Store, and interacts with the Marlin Client.

The Client Application and the Marlin Client are very closely integrated. In many cases, for example, the Marlin Client will be embedded within the application as a plug-in. For some devices (such as portable media players), the Marlin Client and the Client Application may actually be built into the device itself.

How the application interacts with the Web Store is outside the scope of Marlin specifications, except for one requirement: The Web Store must supply Action Tokens to the application, as described in §4.5.1.

One possibility for the Client Application/Web Store interaction is that an application on a PC may have a small HTML browser that displays web pages provided by the Web Store, and user selection of a link on a page triggers appropriate interaction between the application and the Web Store. A portable device may have another interface to the Web Store that can get a list of what is available and let users purchase content.

### 3.1.3 Marlin Client

A Marlin DRM Client (hereinafter referred to simply as a “Marlin Client”) is a Marlin-compliant application running on a PC or another device. It is the functional component responsible for requesting licenses and links (§4.5.3), and for controlling access to protected content.

A Marlin Client utilizes Marlin protocols to communicate with Marlin services on a Marlin Server, e.g., to acquire a license associated with an instance of content. After a license has been acquired, then when the user wants to access the content (for example, to play it), the Client Application asks the Marlin Client to check the terms of the license. If the license permits the access, the Marlin Client provides the Client Application the key required to decrypt the content. The Client Application decrypts the content and provides the requested access.

An example of Marlin Client software is the Sushi Marlin Client SDK (hereinafter referred to as the “Sushi SDK”), which implements all the functionality necessary to be a Marlin Broadband Client. The Sushi SDK was developed by Intertrust. Since it is expected that many Client Applications developed in the near future will utilize the Sushi SDK, this document provides some explicit details regarding interaction with that SDK.

### 3.1.4 Marlin Server

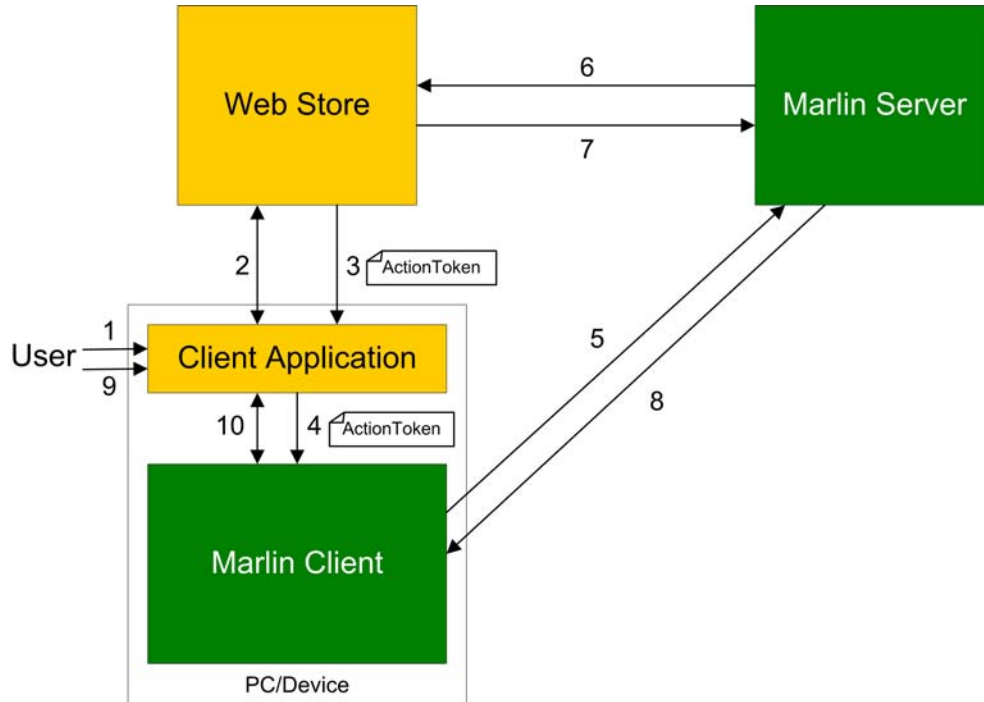
A Marlin DRM Server (hereinafter referred to simply as a “Marlin Server”) provides all the services that are needed by a Marlin Client. It is responsible for device registration, license acquisition, etc. In practice, often the same company will run both the Web Store and the Marlin Server.

A Marlin Server is a pass-through entity. It checks that requests it receives are properly formed and signed correctly, using keys that are not revoked. Then it simply utilizes the information in each request to create a corresponding request to the Web Store back-end, which applies the business logic required to fulfill the request. The Marlin Server then forms a response to the Marlin Client based on the information provided by the back-end service.

An example of a Marlin Server is the BlueWhale Marlin Broadband Server (hereinafter referred to as the “BlueWhale Server”) developed jointly by Intertrust and Sony. Since it is expected that many service providers will utilize the BlueWhale Server, this document provides some explicit details regarding interaction with that server.

## 3.2 Basic Interactions

Figure 2 illustrates the basics of a sequence of interactions for a common scenario, in which the user selects and purchases content, and the corresponding license governing the usage of the content is acquired.



**Figure 2 DRM System interactions**

The steps shown in Figure 2 are the following:

- Step 1. The user runs the Client Application.
- Step 2. The Client Application, directed by the user, interacts with the Web Store to present available content to the user. During this process, any user authorization (e.g., logging into an account or payment processing) must take place. When the user selects content for purchase, rental, or subscription, and completes payment, the remaining steps are triggered.
- Step 3. The Web Store supplies the Client Application information about how to obtain the purchased content. It also supplies a Marlin-defined XML document called an *Action Token*. The Action Token specifies a sequence of actions required to acquire the license corresponding to the content. It specifies which types of requests must be made to the Marlin Server, and any parameters that should be passed in those requests. For more information on Action Tokens, see §4.5.1.
- Step 4. The Client Application passes the Action Token to the Marlin Client. If the Marlin Client is the Sushi SDK, the Client Application passes the Action Token to the SDK API method `SHI_Engine_ProcessServiceToken`. (Not shown: The Client Application retrieves the content at some point.)
- Step 5. The Marlin Client parses the Action Token, and constructs and sends appropriate requests (SOAP service requests) to the Marlin Server. The Marlin Client and Marlin Server interact using Marlin protocols.
- Step 6. The Marlin Server, for each request sent by the client, sends a related request to the Web Store back-end.

- Step 7. The Web Store back-end applies the specific business logic required to check the validity of the requested operation, and sends the server a response containing all the information the server needs to form a response to the Marlin Client.
- Step 8. The Marlin Server forms and sends a response to the client. There may be several trips between the Marlin Client and the Marlin Server (and between the Marlin Server and the Web Store back-end). Ultimately, if all goes well, the Marlin Server returns the license to the Marlin Client, which provides it to the Client Application.
- Step 9. The user attempts to play the downloaded content.
- Step 10. The Client Application consults with the Marlin Client (e.g., by calling Sushi SDK API methods) to ensure that the license allows this, and to obtain the key used to encrypt the content. If playing the content is allowed, then the Client Application decrypts and plays it for the user.

Note: Delivery of content may actually occur at any time and via a variety of mechanisms. Content delivery may be a simple online request and download. However, since a Marlin content object is encrypted, it can be delivered via other methods, such as physical media, broadcast, peer-to-peer, or push.

## 4 Marlin DRM Architecture More Detailed Picture

Figure 3 provides a more complete picture than Figure 1 of the components that may be involved in a typical Marlin DRM System. Subsequent sections describe the additional optional components (§4.1), the technologies and communication protocols utilized in a Marlin DRM System (§4.2 and §4.3), and the key players and objects involved in DRM interactions (§4.4 and §4.5). Finally, §4.7 summarizes the different types of services offered by the Marlin Server.

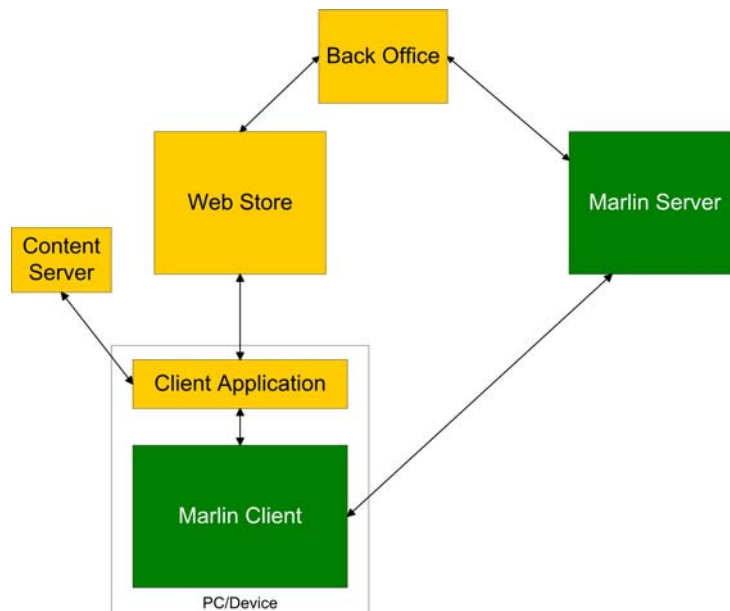


Figure 3 Marlin DRM architecture components

## **4.1 Optional Components**

In addition to the components illustrated in Figure 1, Figure 3 includes the following:

### **4.1.1 Back Office**

Frequently, the Web Store component shown in Figure 1 will be split into two parts: a component (that will be called the “Web Store” in the rest of this document) providing content navigation and access, and a Back Office component providing the back-end business logic. The Back Office is responsible for managing user accounts and license information, performing database transactions, etc. For each request sent by the Marlin Client to the Marlin Server, the server sends a related request to the service Back Office, which will apply the specific business logic required to fulfill it. Based on the response returned to the server from the Back Office, the server forms an appropriate response to the client.

Note: Intertrust is developing a basic example implementation of both a Web Store and a Back Office. The example implementation is called Seahorse.

### **4.1.2 Content Server**

When content is requested by the user, sometimes the Web Store provides it directly to the Client Application, but sometimes content is instead obtained from a separate Content Server.

How the application obtains content is up to it, and outside the scope of the Marlin specifications. Content does not have to be obtained from an online service. It could be obtained using other methods, such as from a memory card or via USB mass storage. It can be downloaded via HTTP, multicast, or other protocols.

## **4.2 Incorporated Technologies**

To facilitate adoption and implementation, the Marlin DRM system uses industry standards wherever practicable. As a result, the Marlin specifications incorporate many industry standard technologies, including security standards from IETF, W3C, ISO, OASIS, and others. To support flexible consumer usage scenarios and business models, Marlin also incorporates two important platform technologies known as the Octopus DRM architecture and the NEMO framework. Each of these platform technologies is briefly defined in the sections that follow.

### **4.2.1 Octopus DRM**

Octopus is a general-purpose DRM architecture that can be applied to a variety of applications ranging from enterprise document control to medical record privacy protection, consumer media copyright protection, or any other system requiring distributed governance and control of information. At the center of an Octopus system is an Octopus DRM Engine—a small, lightweight component responsible for determining whether access to content should be granted in a given set of conditions. By itself, Octopus is entirely agnostic regarding content formats, hardware/software platforms, and business semantics. That is, the Octopus DRM Engine responsible for governing access to content is entirely unaware of the type of content it protects, and is not bound by a particular set of semantics (e.g., not limited by predefined meanings or hard-coded

“rules languages”). The Marlin specifications are an application of this generic DRM architecture to consumer media.

One of the distinguishing features of an Octopus-based system, and an attribute that makes it particularly well suited for Marlin, is its ability to separate the *protection* of content from the *governance* of that content. This allows, among other benefits, the ability to issue rights to access content separately from information that governs where or when it can be used. This enables great end-user flexibility; for example, a service provider can issue rights to a customer to use content, but can allow that customer to independently manage which devices he or she would like to use that content on at any particular moment.

The Octopus specifications are listed in §7.4.

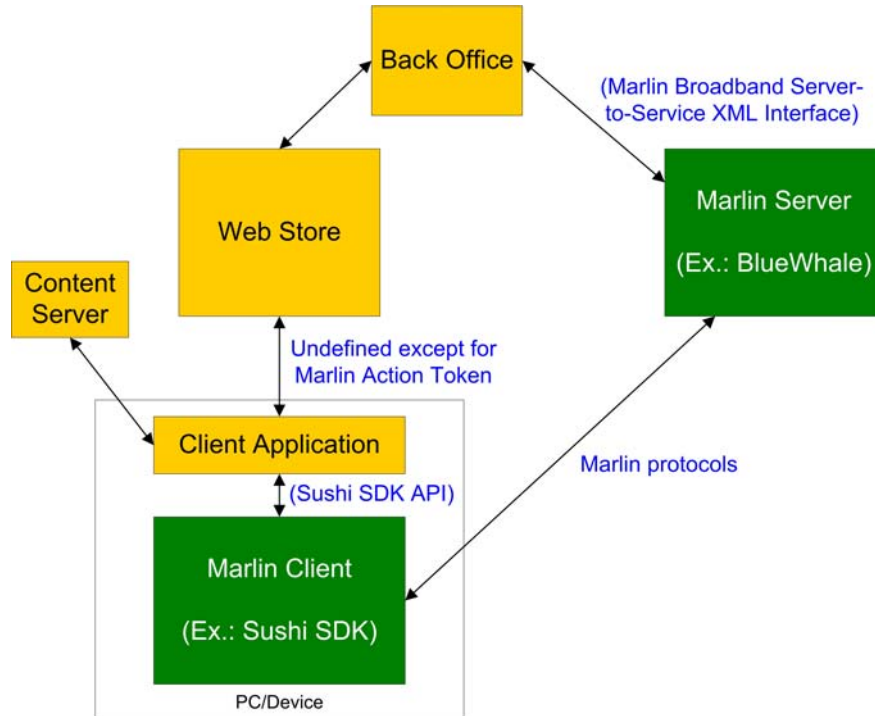
## 4.2.2 NEMO Framework

The NEMO (Networked Environment for Media Orchestration) framework provides the trusted “plumbing” between the various functional components in a Marlin system. NEMO combines SOAP web services with SAML authorizations to provide end-to-end message integrity and confidentiality protection, entity authentication, and role-based service authorization. Through the use of the NEMO framework, Marlin components can leverage a consistent mechanism to ensure that messages are delivered with appropriate protection and are exchanged between entities that are properly authenticated and authorized.

For the purposes of this overview document, we assume without mention that the “NEMO layer” performs all required message security operations between Marlin clients and services. This will allow the discussion to focus on higher-level processes and protocols instead of transmission details. To gain a better understanding of the application of NEMO to Marlin, refer to the “Role of NEMO in Marlin” white paper (§7.1) or the Marlin specifications (§7.2) .

## 4.3 Communication Protocols

Figure 4 shows which components’ interactions must be done using particular protocols. For every pair of components whose interaction is defined, the name of the protocol or specification that must be followed appears next to the line connecting the components. Where nothing appears next to such a line, that means that the interaction is up to the pair of components involved. The protocols that must be used when the Sushi SDK is the Marlin Client and when the BlueWhale Server is the Marlin Server are shown in parentheses.



**Figure 4 Communication protocols**

As you can see from Figure 4, the interactions are defined as follows:

### ***Web Store ↔ Back Office***

The form of this interaction is up to the Web Store and its associated Back Office.

### ***Web Store ↔ Client Application***

The form of the interaction between the Web Store and the Client Application is undefined, except for the requirement that the Web Store send Marlin-defined Action Tokens to the application.

### ***Content Server ↔ Client Application***

Communication between a Content Server and a Client Application is not specified. It could be via HTTP, multicast, etc.

### ***Client Application ↔ Marlin Client***

There are no specific requirements for communication between a Client Application and a Marlin Client in general. However, communication between a Client Application and the Sushi SDK is via the Sushi SDK API.

### ***Marlin Client ↔ Marlin Server***

Communication between a Marlin Client and a Marlin Server is always via Marlin protocols based on NEMO. The service requests and responses contain Octopus objects.

**Marlin Server ↔ Back Office**

There are no specific requirements for how the communication between a Marlin Server and a Back Office is done. However, communication between a BlueWhale Marlin Server and a Back Office is defined, in the *Marlin Broadband Server-to-Service XML Interface Definition* document.

**Web Store ↔ Marlin Server**

There are no direct interactions between the Web Store and the Marlin Server. However, the Marlin Server provider needs to supply the Web Store provider the information required in order for the Web Store to reference a Configuration Token (described in §4.5.1). That is, the server needs to tell the Web Store the Configuration Token ID, version, and URL, and needs to provide updated information whenever it changes.

## 4.4 Key Players

The individuals and entities that are responsible for performing the processes described in this document are primarily users, service providers, and in some instances, content providers.

### 4.4.1 Users

In this document, the term “user” refers to an individual (or a group of people, such as a family) who uses the Marlin DRM system to acquire and play media. A user establishes an account with a Web Store in order to be able to purchase or rent digital media content, or to sign up for a subscription service. The user interacts with the Client Application to obtain and play content. The downloaded content is typically considered to be owned by the user, and can be played by that user on any devices registered to him or her.

### 4.4.2 Service Providers

“Service provider” is the generic term used to describe the entity or organization that is responsible for selling or distributing digital media content and associated licenses. Service providers are not limited to any particular type of business model for licensing or distributing content. They can choose to support “a la carte” individual content licenses, rentals, subscription-based services, or a hybrid of these.

### 4.4.3 Content Providers

A content provider supplies digital media content. Frequently, the service provider also serves the role of a content provider and aggregates content from content owners and distributes it to users. However, Marlin is flexible and allows for a variety of mechanisms for content delivery.

## **4.5 Key Objects**

### **4.5.1 Tokens**

#### ***Action Token***

The one requirement specified for the interaction between a Web Store and a Client Application is that the Web Store send the application an Action Token whenever the application (or, more specifically, the Marlin Client) needs to do something to implement the DRM functionality required to satisfy a user request. An Action Token is an XML document defined in the Marlin Delivery System specifications, such as the Marlin Broadband Delivery System Specification.

The Client Application, after receiving an Action Token, passes it to the Marlin Client. For example, if the Marlin Client is the Sushi SDK, the Client Application passes the Action Token to the Sushi SDK API method SHI\_Engine\_ProcessServiceToken.

An Action Token directs the Marlin Client to send a sequence of requests to a Marlin Server. These could include, for example, a request to register or deregister a device (indicating that it is or is not associated with the user), or a request to acquire a license, preceded by requests that must be performed prior to license acquisition. The Action Token contains a list of the types of requests that need to be made to the Marlin Server, and any parameters that should be passed in those requests.

The Marlin Client parses the Action Token and, for each step it specifies, the client constructs an appropriate service request to be sent to a Marlin DRM Server. The server processes the request and returns a response. The client processes the response and then sends another request, if the Action Token indicates that another one needs to be made. Thus, a sequence of requests and responses are made between the Marlin Client and the Marlin Server, based on the Action Token.

#### ***Configuration Token***

An Action Token by itself does not have enough information to let the Marlin Client know where the specified service requests should be sent. It contains a reference to a Configuration Token that specifies to which service(s) the different types of requests should be sent. Specifically, the Configuration Token, another XML document, specifies the endpoint addresses to which the SOAP requests should be sent, and also provides keys and other information required for secure communication with a NEMO service.

An Action Token references a Configuration Token by ID, version, and URL. If the Marlin Client has not previously obtained the referenced Configuration Token, it resolves the specified URL to obtain the token. Most or all Marlin Clients, such as the Sushi SDK, will store each Configuration Token they download, so if the same Configuration Token (that is, one with the same ID and version) is specified in another Action Token, the Marlin Client does not need to download it again.

#### ***Business Token***

How does the Back Office processing a request from the Marlin Server know the context for the request? Via a Business Token. Each request specified by an Action Token includes a Business Token, which is an XML element containing opaque data whose content is up to the Web Store. It may contain things like transaction IDs and state information.

The Marlin Client simply passes the Business Token, unchanged, in its service request to the Marlin Server, and the Marlin Server passes the Business Token to the Back Office. The Back Office, like its associated Web Store, understands how the information in the Business Token can be used.

As an example, suppose the Action Token specifies that a license acquisition is required. The Marlin Client includes the Business Token in its license request to the Marlin Server. The Marlin Server does not know the usage rules for the license, or what the license should contain. It sends a request to the Back Office, essentially saying “I have a license request on behalf of this user, and here is the relevant Business Token, so please give me the information I need in order to construct the license.” The Business Token may point to a transaction record in the Back Office database so that the Back Office can look it up and confirm that the relevant content was sold to the user on whose behalf the license is requested. If that is the case, the Back Office returns the license information, such as an indication that the user may play the corresponding content for 30 days. The Back Office does not need to worry about the format of the license, or its cryptographic packaging. It just tells what the terms of the license should be. The Marlin Server creates the license, including the cryptographic objects, according to the Marlin specifications, and sends it back to the Marlin Client.

## 4.5.2 Content and Licenses

In response to a license acquisition request, the Marlin Server returns a license, encoded as an XML document containing a set of Octopus objects. The Octopus objects specify the governing usage rights for the content corresponding to the license, that is, the conditions under which the content can be used.

The license indicates which content it applies to by specifying the content ID(s). A piece of content (either the entire content or each of the streams in it) contains a globally unique ID. It also contains the content data (e.g., audio or video data streams), which is encrypted using a key stored in the license. That key is referred to as the *content key*. The content key is itself encrypted using a key possessed by the entity that is authorized to access the content (the entity that the content is considered to be “bound” to).

Since content is encrypted, it can be distributed separately from the license that governs its usage and enables its decryption. The Marlin specifications do not say anything about whether content and its corresponding license should travel together. They can either be a unit or be obtained separately. They can be in a single file or in separate files. If a Client Application obtains them separately, it is up to the application as to whether to combine the content and the license into a single file. This may be commonly done, for management convenience and efficiency. For example, if the content is transferred from one device to another, and the license is in the same file, then the second device does not have to do a license acquisition. If the content and the license are not in the same file, the second device has to obtain an Action Token and acquire the license.

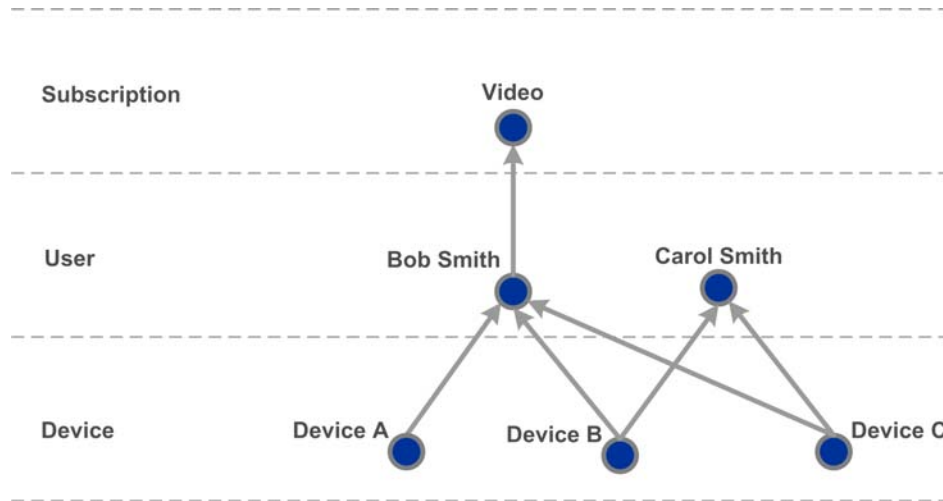
## 4.5.3 Nodes and Links

One of the unique elements of an Octopus-based DRM system such as Marlin is the use of Node and Link objects to express relationships between the principals within the system (e.g., users, devices, and subscriptions). In most traditional DRM systems, licenses are directly bound to the device that is used to obtain the rights. In Marlin, a License is typically bound to a user (more precisely, to an Octopus Node representing the user), and relationships between users and devices, or users and subscriptions, are

maintained separately, through a system of *Nodes* and *Links*. This ability to separate the governance rules (e.g., the License) from the frequently-changing environment of user devices and subscription status allows for a more flexible user experience.

In Marlin, Octopus Nodes are defined that represent logical entities or “principals” within the system. For example, Personality Nodes represent devices, User Nodes represent users, and Subscription Nodes represent subscriptions.

Links are self-protected objects that represent relationships between these principals, forming a directed graph. An example of such a graph is shown in Figure 5.



**Figure 5 Node and Link directed graph**

In Figure 5, Nodes are illustrated as circles, and Links between Nodes are shown as arrows. The actual semantic of a Link is a function of the business model the Link issuer is enabling. For example, for many issuers, the presence of a Link (or arrow) between two Nodes may indicate that there is a relationship between the two Nodes, and this relationship may be described as representing that a particular Node “belongs to” or “is a member of” the Node that the arrow is pointing to. Using this interpretation, the set of Nodes and Links in Figure 5 indicates that Device A “belongs to” user “Bob Smith”; and that Bob Smith “subscribes to” the “Video” subscription service. Similarly, Devices B and C “belong to” both “Bob Smith” and “Carol Smith”.

This system of Nodes and Links is integral to managing where, how, and when content can be used in a Marlin system. When a License is created, the control program within it will typically include a requirement that a certain Node, such as a User Node (e.g., one representing “Bob Smith”) be *reachable* by the device attempting to access the content. Being reachable means that the Marlin Client running on the device in question contains a valid set of Links from the Personality Node representing the device to the specified Node. This can be a direct Link from the Personality Node to the specified Node (such as from Device A to user “Bob Smith” in Figure 5) or a series of Links (such as from Device B to “Bob Smith” to “Video”). This process of indicating in the License’s control program that a particular Node must be reachable is referred to as *targeting* to that Node.

As another example of what the figure represents, note that Device A will be allowed to play content targeted to Bob Smith and/or the “Video” subscription, but it will not be able to play content targeted to Carol Smith, since there is no Link between Device A and Carol Smith.

#### **4.5.4 Agents**

An Agent is an object that carries a control program (code). An Agent is sometimes returned as a consequence of contacting a remote service. The Marlin Client receiving an Agent from a Marlin Server is expected to execute the control program and return the result (success or failure) to the Marlin Server.

The Agent control program performs actions required to be accomplished in the Marlin Client environment. Such actions could include initializing counters or storing certain information in the Seashell persistent-store database (defined by Octopus) that is managed by the Marlin Client. (Seashell is where the Marlin Client securely stores state variables.) Such database information could indicate, for example, that a particular Link is or is not currently valid.

### **4.6 Key Concepts**

#### **4.6.1 Registration and Domains**

*Registration* in Marlin involves grouping devices so that they can share access to content. Currently in Marlin Broadband, a device is registered with a user, and the collection of devices registered with a user is called a *domain*. More specifically, when a device is registered with a user, a Link is generated between the device’s Personality Node and the user’s User Node. (Note: Devices could be linked to other types of Nodes in the future.)

A device may be registered with multiple users, and multiple devices may be registered with any given user.

Many Licenses contain a requirement that a particular (target) Node, often a User Node, must be *reachable* by a device attempting to render content. This condition is satisfied if the device contains a Link or chain of Links to that Node. Thus, all devices in a user’s domain—each device containing a Link from its Personality Node to the User Node—will be able to render content targeted to that User Node, assuming all other License conditions are satisfied.

#### **4.6.2 Deregistration**

*Deregistration* removes a particular device from a domain. The purpose of deregistration is to support models where the number of registered devices or applications is limited. When the maximum number is reached, a user can deregister some devices (for example devices that are lost, no longer in use, etc.) in order to be able to register new ones. When a device is deregistered, the Link established during registration (from the Personality Node to a User Node) is invalidated. This prevents the device from using that Link to access content whose License specifies that that particular User Node must be reachable from the device.

## 4.7 Marlin Services

The Marlin Server provides two types of services: those initiated by Action Tokens (such as those for license and domain management), and those related to security data. Although both types of operations are typically done on the same server, sometimes the Marlin Server can actually be (or can be viewed as) two different servers providing the different types of services. In fact, any number of multiple servers could be utilized to provide the different Marlin services.

### 4.7.1 Services Initiated by Action Tokens

The services initiated by Action Tokens include the following:

#### **Registration Service**

A Registration Service is responsible for the following:

- Issuance of Nodes (e.g., User Nodes and Subscription Nodes)
- Issuance of Links, e.g., from a Personality Node to a User Node, or from a User Node to a Subscription Node
- Deregistration

In response to a Node Acquisition request from the Marlin Client, the Marlin Server returns the appropriate Node.

An Octopus Link is returned by the Marlin Server in response to a Link Acquisition request. In addition, for Personality Node-User Node Links (only), the server may optionally return an Agent (§4.5.4). The Agent carries a control program that performs actions required to indicate that the Link is valid. When an Agent is returned, the Marlin Client executes the Agent and returns the result (success or failure) to the Marlin Server.

When a Link is generated between a device's Personality Node and a user's User Node, the device is said to be *registered* with the user. That is, it has joined the user's domain, as described in §4.6.1.

A *Deregistration* (§4.6.2) request is a request to remove a device from a domain. In response to a Deregistration request, the server returns just an Agent. The Agent code performs actions necessary to deregister the device, including modifying the Seashell database to indicate that the Link previously obtained from the device Personality Node to the User Node is no longer valid. The Marlin Client executes the Agent and returns the result (success or failure) to the Marlin Server.

#### **License Service**

A License Service handles the creation of licenses.

As mentioned in §4.5.2, in response to a license acquisition request, the Marlin Server returns a license, encoded as an XML document containing a set of Octopus objects. The Octopus objects specify the governing usage rights for the content corresponding to the license, that is, the conditions under which the content can be used.

## 4.7.2 Security Data Services

Security data services provided by the Marlin Server are the Data Certification Service and the Data Update Service.

Marlin Clients are expected to have security-related data and keep it up-to-date. Security-related data includes

- a secure clock providing trusted time values
- a License Suspension List indicating which licenses are suspended, thereby preventing access to the content items the licenses apply to
- Certificate Revocation List information specifying which certificates have been revoked

Some of the services initiated by Action Tokens may require that the client security data be up-to-date. That is, an Action Token may specify that a particular request from the Marlin Client to the Marlin Server include a proof that that is the case. The Marlin Client can obtain such a proof by contacting the Data Certification Service (DCS) and passing information about its security data. If the data is sufficiently up-to-date, the DCS supplies the Marlin Client the required proof, in the form of a Data Certification Standard Assertion (DCSA).

If the data is not adequately up-to-date, the DCS tells the Marlin Client which data needs updating. Similarly, if the Marlin Client passes a previously-obtained DCSA in a service request to a Marlin Server, the service may respond that the DCSA is not current enough. In these cases, the Marlin Client interacts with the Data Update Service to obtain up-to-date security data, and contacts the DCS to obtain a current DCSA. It then tries the original request again, passing the new DCSA.

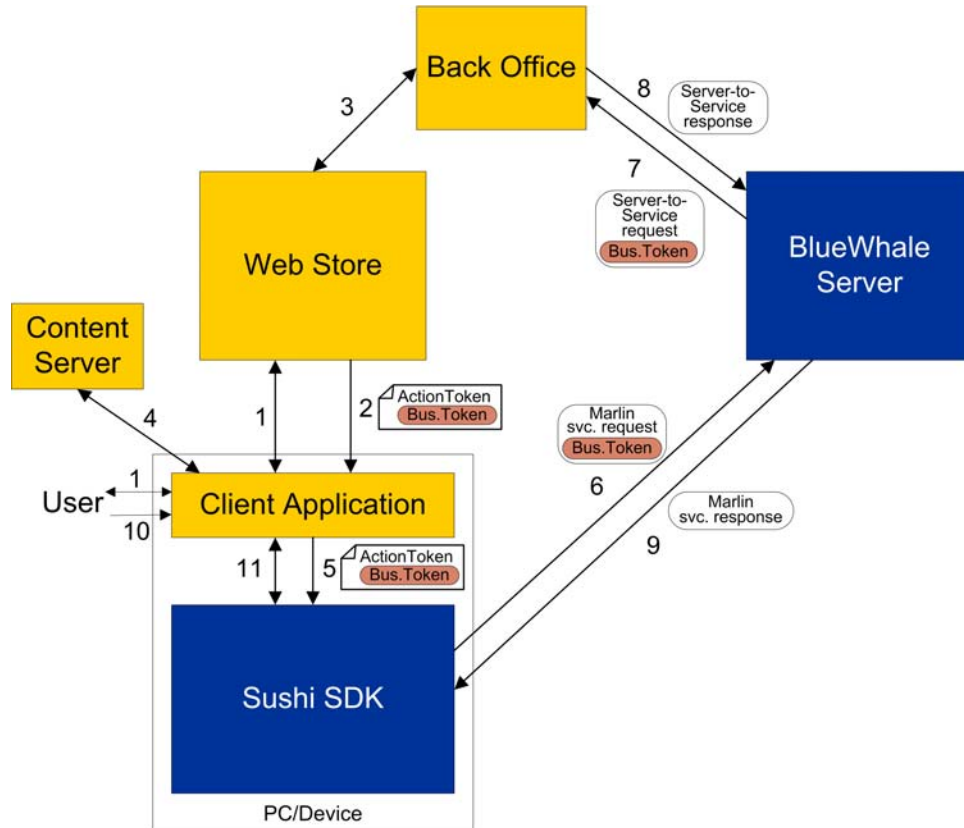
Note: When the Sushi SDK is used as the Marlin Client, the Client Application can call the `SHI_Engine_UpdateSecurityData` API method in order to force an update, rather than having an error condition cause the Sushi SDK to do an update.

## 5 Sample Complete DRM System Interaction

Figure 6 shows the steps that may be involved in a complete DRM System interaction.

This example assumes the following:

- The user has already created an account with the Web Store.
- The user has obtained and installed a Client Application (e.g., a PC Media Player) appropriate for interacting with that particular Web Store.
- The Client Application contains a small web browser for providing viewing and content navigation of web pages provided by the Web Store.
- The Marlin Client is the Sushi SDK.
- The Marlin Server is the BlueWhale Marlin Broadband Server.



**Figure 6 Complete DRM system interaction**

The steps are the following:

- Step 1. The user, using the Client Application, browses the Web Store web pages, finds some content (e.g., a song or video) he or she wants to download, and selects an appropriate link to do so.
- Step 2. The Web Store tells the Client Application where/how to obtain the content. The Web Store also supplies the Client Application an Action Token that specifies all the steps required in order to obtain the license for the content. For each step, the Action Token includes a Business Token, which is opaque data that indicates the context for that particular step.  
  
As an example, prior to obtaining a license, the Marlin Client must first have a User Node representing the user to which the content is targeted, and one or more Links from the Personality Node for the device on which the Marlin Client is running to the User Node (either directly or via a domain). The Action Token would include steps for obtaining the required Node(s) and Link(s), and then would include a step for the license acquisition.
- Step 3. The Web Store also communicates with its Back Office to supply it any information needed regarding the user purchase.
- Step 4. At some point, the Client Application obtains the content.

- 
- Step 5. The Client Application calls the Sushi SDK API method `SHI_Engine_ProcessServiceToken`, passing it the Action Token. This and the following step are described in greater detail in §6.2.
  - Step 6. `SHI_Engine_ProcessServiceToken` processes each step specified in the Action Token, in sequence. It first checks whether the step is needed. (It wouldn't be, for example, if it specified acquisition of a Node the Marlin Client already has.) For each step that is needed, `SHI_Engine_ProcessServiceToken` constructs an appropriate service request (using Marlin protocols), and uses HTTP to send the request to the BlueWhale Server. Each service request includes the Business Token supplied by the Action Token for that particular step.
  - Step 7. The BlueWhale Server constructs and sends a related request (including the Business Token) to the Back Office so that the Back Office can apply appropriate business logic.
  - Step 8. The Back Office applies the business logic and returns information required by the Marlin Server in order to construct a response for the Sushi SDK.
  - Step 9. The BlueWhale Server sends a response to the Sushi SDK. There may be several trips between the Sushi SDK and the BlueWhale Server (and between the BlueWhale Server and the Web Store Back Office). Ultimately, if all goes well, the BlueWhale Server returns the license to the Sushi SDK, which provides it to the Client Application.
  - Step 10. At some point after the Client Application has obtained the content and its associated license, the user requests playing of the content.
  - Step 11. The Client Application consults with the Sushi SDK to determine whether the license allows the content to be played, and, if so, decrypts and plays it. The specific substeps for this step, including the Sushi SDK API methods that must be called, are described in §6.3.

## 6 Client Application Usage of Sushi SDK API

The Sushi SDK API is a simple application interface providing access to the Sushi SDK implementation of all the functionality necessary to be a Marlin Broadband Client. The API is an ANSI C interface that adopts an object-oriented style. The following sections specify explicit Sushi SDK API calls that are made by Client Applications in common situations. It is not meant as a complete reference, but rather as an introduction to some of the methods, whose details you can read about in the Sushi documentation listed in §7.5.

### 6.1 Initialization and Specifying a Callback Listener

The lifecycle of a Client Application's interaction with the Sushi SDK starts with a call to **SHI\_Application\_Initialize** and ends with a call to **SHI\_Application\_Terminate**. After initializing the SDK, an application calls **SHI\_Engine\_Create** to create a `SHI_Engine` object that is the top-level interface to the DRM engine. All the major SDK functions are invoked by calling methods of a `SHI_Engine` object, or of an object returned by such a method.

The SHI\_Engine encapsulates the state of a DRM engine, provides access to the internal data structures, and supplies information about transactions it is performing on behalf of the application. Here is a sample SHI\_Engine\_Create call:

```
SHI_Engine* engine;
SHI_EngineConfig config ...; // set up config
SHI_Result result;
result = SHI_Engine_Create(&config, &engine);
```

The first argument to SHI\_Engine\_Create is a pointer to a SHI\_EngineConfig, defined as follows:

```
typedef struct {
    SHI_Flags          flags;
    SHI_EngineListener listener;
} SHI_EngineConfig;
```

In the current release of the SDK, no flags are defined, so the *flags* field should be 0. The SHI\_EngineListener is supplied so that it can be notified of the progress of ongoing transactions, such as service transactions that are performed asynchronously. The listener is a reference to an implementation of an event notification callback interface. Typically, the listener is the application itself. SHI\_EngineListener is defined as follows:

```
typedef struct {
    const SHI_EngineListenerInterface* iface;
    SHI_EngineListenerInstance* instance;
} SHI_EngineListener;
```

SHI\_EngineListenerInterface is an interface with a function table pointing to a single function that is called by the SDK to notify the listener that an event has occurred. It is defined as follows:

```
struct SHI_EngineListenerInterface {
    void (*OnEvent)(SHI_EngineListener self,
                   SHI_EngineEventType type,
                   const SHI_EngineEvent* event);
};
```

The SHI\_EngineListener is a reference to the listener, and the *event* parameter is a pointer to the event structure that contains information about the event. The possible event types are

```
typedef enum {
    SHI_ENGINE_EVENT_SERVICE_ADDED,
    SHI_ENGINE_EVENT_SERVICE_REMOVED,
    SHI_ENGINE_EVENT_SERVICE_CHANGED,
    SHI_ENGINE_EVENT_SERVICE_USER_ADDED,
    SHI_ENGINE_EVENT_SERVICE_USER_REMOVED,
    SHI_ENGINE_EVENT_SERVICE_USER_CHANGED,
    SHI_ENGINE_EVENT_TRANSACTION_BEGIN,
    SHI_ENGINE_EVENT_TRANSACTION_END,
    SHI_ENGINE_EVENT_TRANSACTION_PROGRESS,
    SHI_ENGINE_EVENT_LICENSE_DATA_RECEIVED
} SHI_EngineEventType;
```

## Event Structures

SHI\_EngineEvent is the base class for all event structures. When the listener is called, it is passed a pointer to a SHI\_EngineEvent. Based on the SHI\_EngineEventType also passed, the event pointer should be cast to a specific data structure type. For example, if the event type is SHI\_ENGINE\_EVENT\_LICENSE\_DATA\_RECEIVED, the event pointer should be cast to SHI\_LicenseDataReceivedEvent.

SHI\_EngineEvent is defined as follows:

```
typedef struct {
    const void* cookie;
    SHI_EngineEventInfoItems* info;
} SHI_EngineEvent;
```

The *cookie* is an application-supplied (§6.2) arbitrary value utilized to identify the particular interaction. SHI\_EngineEventInfoItems contains a pointer to a list of items containing details about the event.

SHI\_LicenseDataReceivedEvent is defined as follows:

```
typedef struct {
    SHI_EngineEvent base;
    const void* data;
    SHI_Size size;
} SHI_LicenseDataReceivedEvent;
```

The *data* pointer is a pointer to the license data, and the *size* field specifies the size, in bytes, of the data.

## 6.2 Action Token Processing

For each Action Token received from the Web Store, the Client Application should call the **SHI\_Engine\_ProcessServiceToken** method, passing it the token and an arbitrary value referred to as a *cookie*. SHI\_Engine\_ProcessServiceToken starts a transaction to process the token. The *cookie* is utilized to identify this particular interaction. It will be passed back to the engine listener in callbacks regarding any transactions related to the processing of the token. In order to receive the callbacks, your application must call **SHI\_Application\_ProcessCallbacks**.

The token processing may trigger a number of additional transactions, each of which is reported to the engine listener. The initial event reported will have SHI\_EngineEventType SHI\_ENGINE\_EVENT\_TRANSACTION\_BEGIN. Throughout the processing of the event, the different steps will be reported by events with type SHI\_ENGINE\_EVENT\_TRANSACTION\_PROGRESS. Such events report the current step and the total number of steps that will be performed. When the processing of the transaction is completed, the listener receives an event with type SHI\_ENGINE\_EVENT\_TRANSACTION\_END.

If the Action Token included a step for acquiring a license, then if that is successful, an event of type SHI\_ENGINE\_EVENT\_LICENSE\_DATA\_RECEIVED will be sent to the callback. As described in §6.1, the event will be a SHI\_LicenseDataReceivedEvent containing a pointer to the license data and a field specifying the data size, in bytes.

## 6.3 Checking a License and Playing Content

When the Marlin Client obtains a license from a Marlin Server, it simply passes the license to the Client Application. It is up to the application as to whether to add that license to the content file or store the license separately, as described in §4.5.2.

In order to play the content that the license applies to, the application must first ensure that the conditions specified in the license are satisfied, and must obtain the key required in order to decrypt the content.

When the Marlin Client is the Sushi SDK, the Client Application calls SDK API methods to create necessary data structures, determine whether the access conditions are satisfied, and obtain the key. Here are the steps the Client Application must perform:

1. Obtain the license.
2. Call **SHI\_Engine\_CreateLicense** to obtain a SHI\_License object.
3. Call **SHI\_License\_ProcessLicenseData** to register the license data. Pass it a pointer to the license data, and specify the size of the data, in bytes.
4. Obtain from the content file the content IDs for the parts of the content to be accessed. For example, in an MP4 file, there is one content ID for each media track in the file.
5. Call **SHI\_License\_SetContentIds** to register the content IDs. Pass it an array of the IDs and the number of elements in the array.
6. Call **SHI\_License\_CreateAction** to create a SHI\_Action for the appropriate type of action. Pass it the name of the action (e.g., “Play”), any parameters required, and the address of a SHI\_Action pointer that will be set to refer to the SHI\_Action created by this method.
7. Call **SHI\_Action\_Perform** to ensure the license allows the specified access.
8. If the action is allowed, call **SHI\_License\_GetContentKey** for each piece of content that will be accessed. Pass it a pointer to the content ID and the address of a SHI\_Data pointer. SHI\_License\_GetContentKey sets the latter pointer to refer to a SHI\_Data object containing the content key.
9. Use the content key(s) to decrypt the content.
10. Perform the action, such as playing the content.

## 7 For Further Information

### 7.1 White Papers

- *Marlin Architecture Overview* (An introductory-level overview to the Marlin DRM architecture and its application to common implementation use cases. Much of its material is presented at a higher level than the document you are reading. For example, its use cases combine under the term “service provider” all the functionality of what this document refers to as the Web Store, the Back Office, and the Marlin Server.)
- *The Role of Octopus in Marlin*

- *The Role of NEMO in Marlin*
- *Refusal, Remediation and Renewability in Marlin*

## **7.2 Marlin Specifications**

- *Marlin Core System Specification*
- *Marlin Broadband Delivery System Specification*
- *OMArlin Specification*
- *Marlin IPTV-ES Specification*

## **7.3 NEMO Specifications**

- *NEMO Message Bindings*
- *NEMO Security Bindings*
- *NEMO Trust Management Bindings*
- *NEMO Discovery Bindings*
- *NEMO Policy Bindings*

## **7.4 Octopus Specifications**

- *Octopus Controls*
- *Octopus Objects*
- *Plankton Virtual Machine*
- *Scuba Key Distribution*
- *Seashell Object Store*
- *Octopus Object Serialization*

## **7.5 Sushi SDK Documentation**

- *Sushi (Marlin Client) SDK API*
- *Sushi (Marlin Client) SDK API Example Programs*
- Sushi SDK API reference documentation automatically generated by an open-source documentation utility called doxygen.

## **7.6 BlueWhale Documentation**

- *Marlin Broadband Server Operations and Installation Manual*
- *Marlin Broadband Server-to-Service XML Interfaces Document*

## 8 Glossary

The following table lists a number of frequently used terms and their definitions, along with cross references to the sections in which the terms are introduced. Within each definition, terms defined elsewhere in the table are capitalized.

| Term                              | Definition  | Section |
|-----------------------------------|---|---------|
| Action Token                      | An XML-encoded document that directs a Marlin Client to perform a sequence of actions, such as obtaining a User Node from a Registration Service or acquiring a License from a License Service. An Action Token specifies the ID, version, and location of a Configuration Token telling the service locations and access information. It also contains information necessary to make Marlin protocol requests for communicating with the specified services. An Action Token is issued by a Web Store or an e-commerce system, for example after a user utilizing a browser embedded in a Client Application has selected a hyperlink requesting purchase of a particular piece of content from a Web Store. | §4.5.1  |
| Authentication                    | The process of validating the identity of an individual, device, entity, or system.   | §4.2.2  |
| Back Office                       | A component providing the back-end business logic for a Web Store.  | §4.1.1  |
| Binding a License to a Node       | Encrypting the Content Key (for the Content the License applies to) with either the public key or the symmetric secret key associated with the specified Node (such as a User Node). The License for the content is said to be “bound” to that Node. Only Devices that have access to the private key (or secret key, as appropriate) of that Node have the necessary key to decrypt the Content Key, which can then be used to decrypt the Content. Access to the private or secret key is possible when there is a path of Links from a Device Personality Node to the Node that the Content is bound to.   | §4.5.2  |
| BlueWhale Marlin Broadband Server | An implementation of a Marlin Server.   | §3.1.4  |

| <b>Term</b>              | <b>Definition</b>   | <b>Section</b>    |
|--------------------------|---|-------------------|
| Business Token           | An XML element containing opaque data whose content is supplied by the Web Store. The Business Token is relayed (via a Marlin Client and a Marlin Server) to the Back Office application to provide context for a given service request. The Web Store includes a Business Token for each request specified in each Action Token it creates.  | §4.5.1            |
| Client Application       | An application that interacts with the user, communicates with the Web Store, and interacts with a Marlin Client.   | §3.1.2            |
| Configuration Token      | An XML document that includes relatively static information for one or more Marlin services. It includes the locations of the services and information required to access them.   | §4.5.1            |
| Content (Marlin Content) | Digital media (such as a music or video file) encrypted with a Content Key and packaged into a Marlin file format. A content file is usually a collection of several media tracks (for example audio and video).  | §4.5.2            |
| Content Key              | A symmetric cryptographic key used to encrypt and decrypt an instance of Content. The encrypted Content Key is stored in the License corresponding to the Content.  | §4.5.2            |
| Content Provider         | A supplier of Content.  | §4.4.3            |
| Content Server           | A server that may supply Content to a Client Application.   | §4.1.2            |
| Deregistration           | A protocol used to invalidate a Link. For example, Deregistration may undo a membership relationship between a Personality Node (representing a Device) and a User Node by invalidating the Link connecting the two.  | §4.6.2,<br>§4.7.1 |
| Device (Marlin Device)   | A self-contained hardware component or software application capable of hosting a Marlin Node with specific Marlin roles.  | §3.1              |
| Domain                   | A collection of Devices (more precisely, a collection of Personality Nodes associated with Marlin DRM Clients). A Device is registered with a Domain (thereby becoming a Domain member) by creating a Link from the Device (Personality Node) to the Domain Node, which, in Marlin Broadband, is a User Node. Typically, a User can play Content on any of the Devices in the User's Domain, if allowed by the License for the Content. | §4.6.1            |

| Term   | Definition   | Section |
|--|--|---------|
| License  | An XML document containing a set of Octopus objects that govern the use of Content and convey the conditions necessary for allowing access to the Content Key used to encrypt the Content.   | §4.5.2  |
| License Service                                      | A Marlin service handling the creation of Licenses.  | §4.7.1  |
| License Suspension List                              | A list of IDs that may be referenced in Licenses to indicate that Licenses have been suspended. Suspension lists (lists of such IDs) are distributed to clients via the DUS (Data Update Service). A License Service may specify in a subscription License that a certain ID must not be present in the current License Suspension List, if for example it expects at some point in time to invalidate a License by updating the suspension list with the ID referenced in that License. This is primarily used in subscriptions where one subscription Link enables access to a large set of content, yet certain elements of that set need to be removed from the set over time. | §4.7.2  |
| Link   | An Octopus object, encoded in XML, that expresses a relationship between two Octopus Nodes.  | §4.5.3  |
| Marlin Client (Marlin DRM Client)                    | A Marlin-compliant Device or application that is able to communicate with Marlin services, e.g., to acquire and evaluate a License which governs access to an instance of Content.   | §3.1.3  |
| Marlin Delivery System Specifications                | Specifications that define how Marlin Content and Licenses are created and delivered via various content distribution systems (such as broadband, broadcast, and mobile), and how content may be imported from a non-Marlin environment into a Marlin environment.   | §4.5.1  |
| Marlin Server (Marlin DRM Server)                    | A server providing the services needed by a Marlin Client. It is responsible for Device Registration and Deregistration, License acquisition, etc.   | §3.1.4  |
| NEMO (Networked Environment for Media Orchestration) | A framework for trusted connections between the various components of a Marlin System. NEMO combines SOAP web services with SAML authorizations to provide end-to-end message integrity and confidentiality protection, entity authentication, and role-based service authorization.   | §4.2.2  |
| Node   | Either an Octopus Node or a NEMO Node, depending on context. In this document, the term “Node” always refers to an Octopus Node.   | §4.5.3  |

| Term                 | Definition  | Section           |
|----------------------|---|-------------------|
| Octopus              | A general-purpose DRM architecture. The Marlin specifications are an application of this generic DRM architecture to consumer media distribution. In particular, Marlin uses Octopus for governance and key management.   | §4.2.1            |
| Octopus Node         | An object, encoded in XML, representing a Marlin DRM System entity. For example, a Personality Node represents a Device, a User Node represents a User, and a Subscription Node represents a Subscription. An Octopus Node includes a public/private key pair or a symmetric key that can be used to bind Licenses to the Node.   | §4.5.3            |
| Personality Node     | An Octopus Node representing a Device.  | §4.5.3            |
| Registration         | A protocol by which a Device establishes a membership relationship with a Domain. Membership in a Domain is represented by a Link from the Personality Node representing the Device to the Domain Node, which, in Marlin Broadband, is a User Node.   | §4.6.1,<br>§4.7.1 |
| Registration Service | A Marlin service responsible for issuing Octopus Nodes and Links, and for disassociating the relationship between two Nodes.  | §4.7.1            |
| Security Metadata    | Metadata necessary for managing the security and trustworthiness of the Marlin DRM System. An example is a License Suspension List.   | §4.7.2            |
| Service Provider     | Entity that provides services, such as a License Service and a Registration Service. Also a generic term used to describe an entity or organization responsible for selling or distributing digital media Content and associated Licenses.  | §4.4.2            |
| Subscription         | An arrangement represented by a License and a subscription Link granting a User the right to access a large collection of Content for a limited period of time. During the validity period of the Subscription (typically encoded as a validity period on the subscription Link), the User is permitted to play or use any Content that is part of the Subscription as many times as they wish. At the end of the validity period, the Subscription may be renewed (via the renewal of the subscription Link). If it is not, any Licenses referencing that subscription and used to access the Content expire and become invalid. | §4.5.3            |

| Term                                     | Definition   | Section |
|--|--|---------|
| Subscription Node                        | An Octopus Node representing a subscription. A Link from a User Node to a Subscription Node indicates that the user has obtained the Subscription and is therefore permitted to play any Content that is part of the Subscription.   | §4.5.3  |
| Sushi Marlin Client SDK                  | An example of Marlin Client software that implements all the functionality necessary to be a Marlin Broadband Client.  | §3.1.3  |
| Targeting a License to one or more Nodes | A process by which a Client Application running on a Device is allowed to access Content only if the associated Marlin Client contains a valid set of Links from the Personality Node (the Node representing the accessing Device) to the Node(s) to which the License is targeted. A License may be targeted, for example, to a User Node, a Subscription Node, or a Personality Node.                          | §4.5.3  |
| Token                                    | An XML fragment with a specific purpose. See, for example, Action Token and Configuration Token.   | §4.5.1  |
| User                                     | An individual or a group of people (such as a family) represented by a User Node. A User uses the Marlin DRM System to acquire and play Content whose usage is governed by a Marlin License.   | §4.4.1  |
| User Node                                | An Octopus Node representing a User.   | §4.5.3  |
| Web Store or e-commerce system           | <p>An entity that is the front end for all the operations that interact with the User. As a result of such an interaction, a Marlin Client can be provisioned with an Action Token that triggers the Marlin Client to contact a Marlin service.</p> <p>Note that this entity is only used for illustration, as the same tokens could be delivered via another mechanism without affecting the specification.</p> | §3.1.1  |