guatemala
DSM-CC Download Server Introduction

Bionic Buffalo Corporation
2003.09.23

Contents

Document Information and Copyright

Document Information and Copyright

Document Context

This document is a portion of the documentation for Bionic Buffalo's guatemala product. Information about the product and other, related documentation can be found at *http://www.tatanka.com/prod/guatemala.html*

Bionic Buffalo offers a family of products implementing DSM-CC and related protocols. For more information, see
http://www.tatanka.com/prod/dsmcc.html

Copyright, License, and Trademarks

Author and Publisher

Bionic Buffalo Corporation
502 North Division Street
Carson City, Nevada 89703
USA

telephone +1 775 882 1842
fax +1 775 882 6047
e-mail *query@tatanka.com*
web *http://www.tatanka.com*

PGP/GnuPG key fingerprint:

```
a836 e7b0 24ad 3259 7c38 b384 8804 5520 2c74 1e5a
```

Document History

Project name: *guatemala*
File name: *gtm_introduction*
Formal revision date: Tuesday 2003.09.23
Last modified: 2003-09-24-14:32

For the latest version of this document, or for other forms of this document, see
http://www.tatanka.com/doc/man/guatemala/index.html

Updates and revisions of this document are announced on Bionic Buffalo's DSM-CC e-mail announcement list. For more information, see
http://www.tatanka.com/bbc/lists.html

Overview

Overview

Bionic Buffalo's guatemala product implements DSM-CC download servers as specified in *Extensions for DSM-CC* (ISO/IEC 13818-6), one of the MPEG-2 family of specifications.

The guatemala product includes a library of software which can be used to implement download server applications. It also includes a server application (dnlserver) which can be used without additional programming.

This document is an overview of the guatemalar product and its use. For a description of the library API, refer to *DSM-CC Download Server API Reference Manual*. For information on dnlserver, refer to *DSM-CC Download* dnlerver *User's Guide*.

Before reading the guatemala manuals, it is suggested that you first read the *DSM-CC Download Common Introduction*, which is part of the angola product documentation. That document provides an overview of the download protocol and how it is used in applications.

Interfaces and Operations

Interfaces and Operations

guatemala defines three interfaces (object classes): Guatemala::Application, Guatemala::ServerManager, and Guatemala::Server. This chapter describes each interface and its operations.

Guatemala::Application

A Guatemala::Application object is created by the application using the guatemala library. It is registered with the ServerManager, and its operations provide a way for ServerManager and Server objects to call the application asynchronously. In traditional terms, it is a call-back mechanism.

The following operations are defined on the Application interface:

> **complete ()** -- informs the application that a scenario has completed, normally or otherwise, giving the reason for the termination; may also signal automatic destruction of an automatically-created Server object
>
> **request ()** -- informs the application that a client has sent a DownloadInfoRequest message, allowing the application to accept or reject the request, load the Server, or take other appropriate action
>
> **server_created ()** -- informs the application that, as a result of connection establishment, a Server object was automatically created

---

Guatemala::ServerManager

A Guatemala::ServerManager object creates Server objects, and handles their network connections. (All Server objects created by a single ServerManager must have the same network addresses.) The ServerManager also deals with some attributes and parameters which are common to the Servers under its control.

The following operations and attributes are defined on the `ServerManager` interface:

`application` -- this attribute notifies the `ServerManager` regarding the identity of the `Application` object

`create_server ()` -- instantiates a `Server` object

`Guatemala::Server`

A `Guatemala::Server` object is create by a `ServerManager`. Each `Server` represents an instance of a "Download Server" as defined by the specification, and includes a state machine distinct from that of other `Server` objects.

Depending on configuration parameters, in reliable environments, a Server listening on a port may replicate itself when a connection is established. This allows automatic support for multiple, simultaneously active clients, so that multiple download operations can procede in parallel.

The following operations and attributes are defined on the `Server` interface:

`activate ()` -- enables operation of the `Server`

`compatibility_descriptor` -- the default compatibility descriptor, as sent in the `DownloadInfoResponse` and `DownloadInfoIndication` messages

`deactivate ()` -- disables operation of the `Server`

`destroy ()` -- destroys a `Server`

`load ()` -- loads contents into the `Server`, and specifies the default module information table to be sent in the `DownloadInfoResponse` and `DownloadInfoIndication` messages

`private_data` -- the value of the private data portion of the `DownloadInfoResponse` and `DownloadInfoIndication` messages

`session_parameters` -- defines operating parameters for the session

`subset_add ()` -- creates and defines a specialized `DownloadInfoResponse` message

`subset_delete ()` -- deletes a specialized `DownloadInfoResponse` message

`update ()` -- modifies the contents of a server already loaded

Example Procedure: Simple Download

Example Procedure: Simple Download

This procedure makes a single image or set of modules available for download. It has limitations, but it requires a minimal amount of coding to implement. It may be useful to test connections or other aspects of an application or system.

Limitations

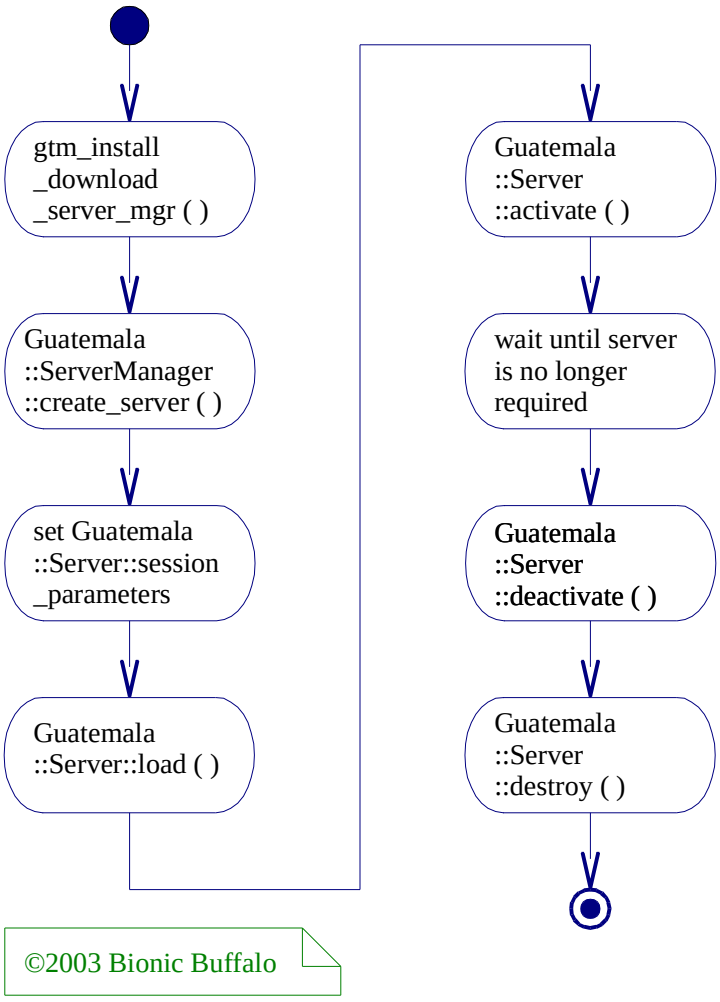
This procedure has limitations. The most significant are:

- Neither compatibility descriptors, nor private data, are supported. There is no option to offer varying content depending on client requirements.
- Configurability is limited. However, additional operations can be added to the basic sequence of operations to adjust the configuration.

Description

1. Install `guatemala` using `gtm_install_download_server_mgr()`. This returns a reference to the `ServerManager` object.
2. Use the `ServerManager` to create a `Server`.
3. Set the `Server::session_parameters` and load content.
4. Activate the server. The server will continue to run until deactivated. In the flow-controlled download scenario, it will accept clients as they arrive. In the non-flow-controlled download and carousel scenarios, it will repeat operation indefinitely.
5. Deactivate the `Server`.
6. Destroy the `Server`.

Sequence of Operations



Example Procedure: Dynamically-Loaded Flow-Controlled Download

Example Procedure: Dynamically-Loaded Flow-Controlled Download

This procedure demonstrates the capability of the server to adjust dynamically the downloaded content in response to the client's compatibility descriptor and private data.

Description

1. Install `guatemala` using `gtm_install_download_server_mgr()`. This returns a reference to the `ServerManager` object.

2. Use the `ServerManager` to create a `Server`. Create an `Application` object using standard CORBA mechanisms. Register the `Application` with the `ServerManager`.

3. Activate the `Server`. The `Server` will listen for a `DownloadInfoRequest` from a `Client`.

4. When a `Client` sends a `DownloadInfoRequest`, then the `Server` will invoke the `Application`'s request operation, letting the `Application` know the contents of the request's private data and compatibility descriptor.

5. Based on the details of the `DownloadInfoRequest`, the `Application` loads appropriate content into the `Server`, then returns from the request operation invoked by the `Server`.

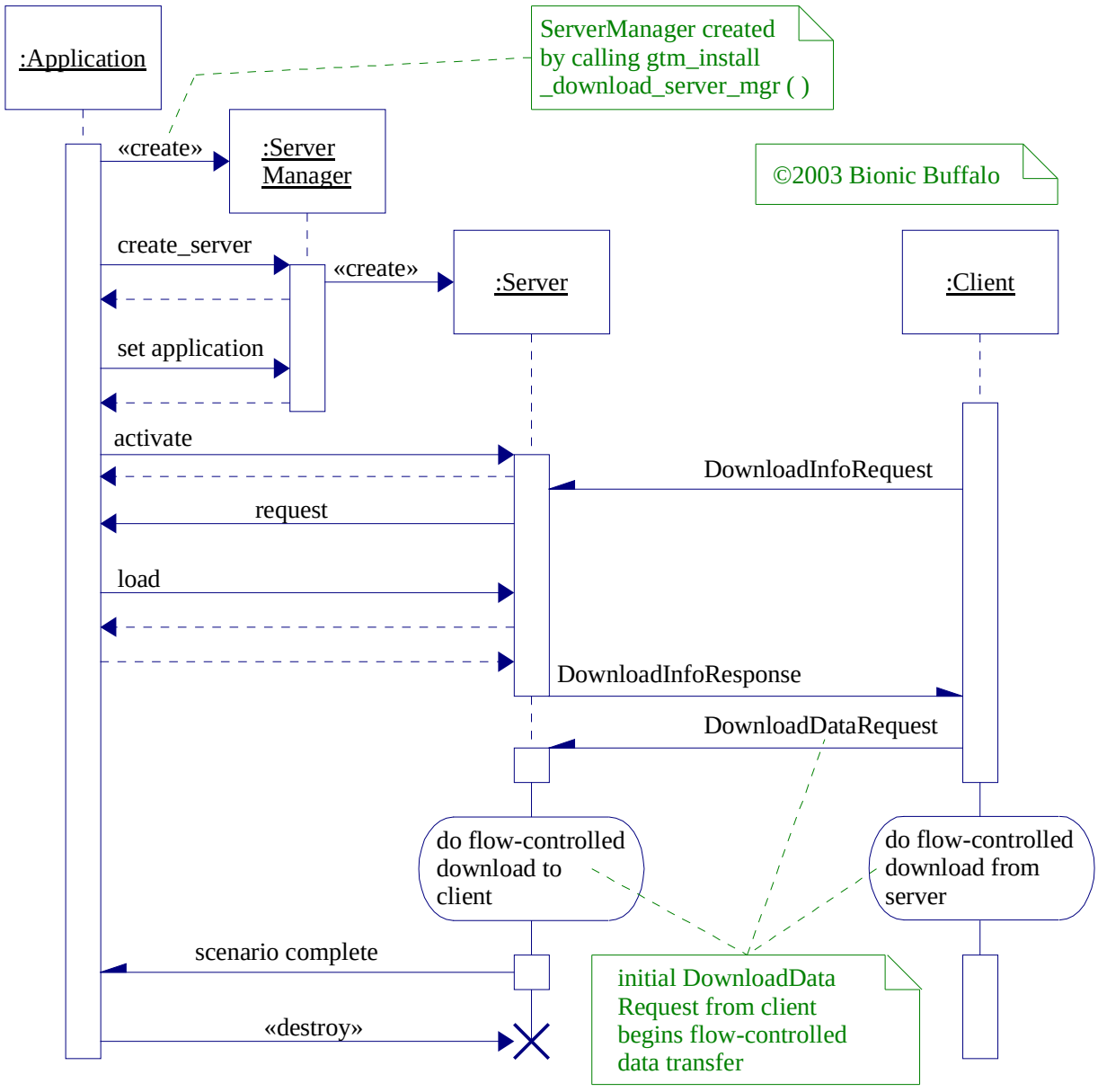
6. When the `Application` returns from the request operation, the `Server` replies to the client with a `DownloadInfoResponse`, which describes the content loaded onto the `Server`.

7. The `Client`, having received the `DownloadInfoResponse`, will allocate buffers or otherwise prepare to receive the content. Then it will begin the transfer by sending an initial `DownloadDataRequest` message.

8. The `Server` downloads the content to the client, then notifies the `Application` that the

scenario is complete. The `Application` destroys the `Server`.

Sequence of Operations



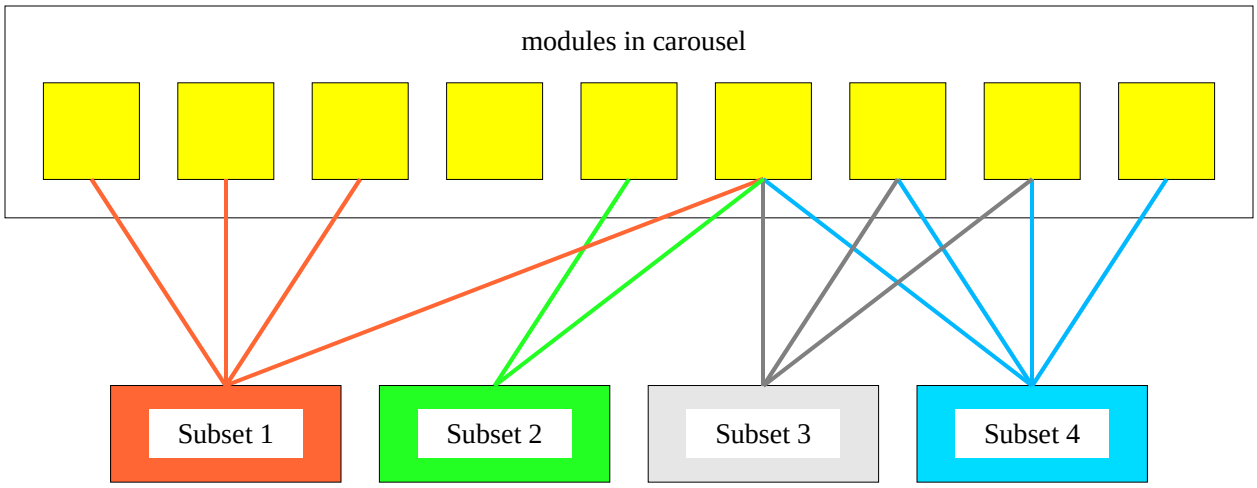
Example Procedure: Complex Carousel

Example Procedure: Complex Carousel

This procedure explains the steps needed to set up a carousel with multiple, differing subsets of modules tailored for different types of clients.

Background and Discussion

The modules in a carousel may be grouped into possibly overlapping subsets, with each subset intended for a different client application, configuration, or version.



A client learns about the modules in a carousel from the `DownloadInfoIndication` messages sent by the server. Among other things, each `DownloadInfoIndication` message contains the following information:

- a list of modules, with their sizes and versions, and up to 255 octets of `moduleInfo` data for each module in the list

- a capability descriptor, with a predefined structure, intended to contain an inventory of hardware and software, along with ancillary information
- up to 65535 octets of “private data”, which can be structured any way the implementor wants

In the simple download example given earlier, the application merely loads the carousel contents and activates the `Server` object. This results in the following default behaviour:

- the Server transmits only a single kind of DownloadInfoIndication message
- there is only one subset, which is the improper subset consisting of all modules loaded into the carousel
- the capability descriptor and private data fields are empty

In order to use carousel subsets, along with multiple kinds of `DownloadInfoIndication` messages, the simple procedure cannot be employed. Instead, the following procedure can be used.

Description

(This description assumes that the `ServerManager` and `Server` objects have already been created. It also omits `Server` deactivation and destruction.)

1. The application loads the `Server` contents with all modules in the carousel, including any which will be in any subset.
2. If the application does *not* want there to be a `DownloadInfoIndication` message describing all of the modules, then `inhibit_complete_set` must be set to `TRUE` when setting the `session_parameters`. Otherwise, the `Server` will advertise all of the modules in one of the `DownloadInfoIndication` messages.
3. The application uses the `subset_add` operation to describe each subset to be advertised.
4. When the `Server` is activated, each subset will result in a different `DownloadInfoIndication` message being sent.