Bionic Buffalo Tech Note #108

DSM-CC Data Carousel and Module Identification

last revised Tuesday 2003.09.22

Introduction

The DSM-CC specification (ISO/IEC 13818-6) describes several data items which can be used to identify data carousels and modules. This Tech Note discusses and summarizes their use.

The following discussion applies to the data carousel scenario only, and not to flow-controlled or non-flow-controlled download scenarios.

Available Identification Mechanisms

The carousels and modules offered by servers may be identified in various ways. These include:

Variable	***Identifies***	***Control Messages?***	***Data Msgs?***	***Remarks***
`serverId`	server address	`DownloadServerInitiate` only	no	associates a specific combination of `compatibilityDescriptor` and `privateData` with a server address
`transactionId`	session	yes	no	versioning mechanism for `DownloadInfoIndication` messages, changed whenever the message content changes
`downloadId`	scenario in progress	from server only	yes	identifies a single instance of a scenario in progress; unique within the network; not for multiplexing
`compatibilityDescriptor`	applicable hardware or software configuration	yes	no	may be used for implementation-specific purposes

Variable	*Identifies*	*Control Messages?*	*Data Msgs?*	*Remarks*
`privateData`	(any)	yes	no	may be used for implementation-specific purposes
`moduleId`	module	yes	yes	
`moduleVersion`	module	yes	yes	used to detect module coherency problems
`moduleInfo`	module	yes	no	arbitary data for each module, which may be examined by client to determine applicability of the module, or may carry ancillary data relating to the use of the module

Discussion of Variables

`serverId`

The `serverId` is found only in optional `DownloadServerInitiate` messages. These are used to redirect a client to a different connection, possibly depending on the message's `compatibilityDescriptor` and `privateData` contents. The `serverId` value is an OSI NSAP address. When there are multiple `DownloadServerInitiate` messages with different `compatibilityDescriptor`s and `privateData`, then the client may select the appropriate `serverId` to locate the applicable carousel.

`transactionId`

A server sends `DownloadInfoIndication` or `DownloadInfoResponse` messages to indicate the carousel's content. Each such message includes a `compatibilityDescriptor`, `privateData`, a `downloadId`, and information about the modules. The `transactionId` is used as a versioning mechanism for these messages: whenever any field in the messages changes (corresponding to an update of the carousel), then the `transactionId` is incremented. A client need only look for changed `transactionId` values; unchanged `transactionId` values represent duplicates of messages which have already been received.

`downloadId`

A `downloadId` identifies a single instance of a scenario in progress. It should be unique within the network, although the specification provides no guidance on how such `downloadId` values are assigned. (In a multi-vendor network, without some type of coordinated assignment of values, it is conceivable that two different carousels might improperly have the same `downloadId` value.)

The `downloadId` is not to be used for multiplexing: it is improper for two different carousels to operate on the same channel, with only the `downloadId` to distinguish between them. This implies that there must be only one carousel on any one channel.

The specification is silent regarding what constitutes a "single instance" of a scenario in progress. New versions of the modules may be sent without changing the `downloadId`, and the specification does not specifically require that adding or removing modules will require the `downloadId` to be changed. In other words, the contents of the carousel may change completely, yet the same `downloadId` may continue to be used. This does not preclude an implementation from adopting stricter conditions regarding `downloadId` uniqueness.

`compatibilityDescriptor`

The `compatibilityDescriptor` is used to indicate that the specified or requested modules are applicable to a specific hardware or software configuration. It has a defined format, described in the specification. A client may use the value of the `compatibilityDescriptor` to notify the server of the type of content the client needs, or to select from among several different content sets.

`privateData`

The `privateData` are used to convey any information required by the implementation. One use is the same as that of the `compatibilityDescriptor`, but without the constraints of that variable's defined syntax and semantics.

`moduleId`

A `moduleId` uniquely identifies a module for a given `downloadId`. The specification anticipates that a module may change (be updated), while its `moduleId` remains constant.

`moduleVersion`

The `moduleVersion` is used to enforce module coherency by detecting changes in module content.

A client may detect a change in `moduleVersion` when the content list in a `DownloadInfoIndication` message changes (which, in turn, triggers a change in `transactionId`.) However, the network may not guarantee that messages are delivered in their original order, and the client may alse detect a version change when the `moduleVersion` in a `DownloadDataBlock` doesn't match the expected value. (This would occur if the new `DownloadDataBlock` messages were received before the relevant `DownloadInfoIndication` message, or if the latter message were lost

entirely.) Receiving an unexpected `moduleVersion` is considered an error, and cancels the current operation.

These rules imply that, when a server updates a module with a new version, it should stop sending the old version. Transmission of the old and new versions ought not to overlap. If an implementation wants to send two versions simultaneously, it should use some other mechanism (such as `moduleId`) rather than `moduleVersion` to distinguish between the two versions.

`moduleInfo`

The `moduleInfo` may be used for any purpose needed by the implementation, including module identification.

If a module's `moduleInfo` changes, then the `transactionId` must change, but the `moduleVersion` may remain the same. A server may change the `moduleInfo` and leave the `moduleVersion` intact, without cancelling a download in progress. For instance, an implementation might use the `moduleInfo` to show the time and date for which a module is valid. If the date and time change, but the module contents do not, the `moduleInfo` could change without affecting the `moduleVersion`.

Content Subsets

A carousel server may send different `DownloadInfoIndication` messages, each referring to a different subset of the carousel's modules. The client may select the appropriate subset (or subsets) by inspection of the messages' `compatibilityDescriptor` and `privateData` values. All such subsets should have the same `downloadId` value. The client uses `moduleId` values, which are unique for a given `downloadId` value, to select applicable `DownloadDataBlock`s.

As the carousel cycles, it will send the `DownloadInfoIndication` messages repeatedly, just as it does the `DownloadDataBlock` messages. The specification requires that the `transactionId` change for changing `DownloadInfoIndication` messages. Since, with subsets, each `DownloadInfoIndication` message is different from the previous ones (even when the carousel content is unchanging), the client might be confronted with a new `transactionId` for each message, if the specification were to be interpreted strictly.

Bionic Buffalo's `guatemala` and `honduras` download software interpret the rule more liberally to allow reuse of `transactionId` values for each subset, when the contents of that subset have not changed. Each subset will have a unique `transactionId` value, which will change when the subset changes. For example, if the server is advertising three subsets, with `transactionId` values *A*, *B*, and *C*, then it will send `DownloadInfoIndication` messages with `transactionId` values

A, *B*, *C*, *A*, *B*, *C*, *A*, *B*, *C*, ...

A client listening to this sequence, and operating by comparing the current `transactionId` value with that from the previous message, will operate correctly, although it will do extra work by reviewing `DownloadInfoIndication` messages it has already seen. (Although, over time, the three subsets have not changed, each message's `transactionId` is different from that of the previous message.) To operate more efficiently, Bionic Buffalo's `honduras` download client caches recently received `downloadId` values, and (optionally) ignores ones it has already seen, even when they were sent several messages previously.

Composite Carousels

In Bionic Buffalo's `guatemala` download server software, the `Server`s created by one `ServerManager` share the same network address. For carousels, this means that they should have the same `downloadId`.

A client seeing such multiple `Server`s, with the same `downloadId`, is not normally able to distinguish them from one another. It sees them as a single carousel.

The primary use for these composite carousels is that of convenience for the server application, which may load or otherwise manage their contents separately. For example, one `Server` may be used for ordinary content, with another activated only for special content. The ordinary carousel will operate independently of the special carousel, subject to overall bandwidth limitations.
