STORAGETEK®

APPLICATION NOTE

Lifecycle Fixed Content Manager 100 Series solution

Content certificates: integrating verifiability with enterprise business logic

DECEMBER 2004

1. Executive summary

Many enterprises are building business processes to manage the lifecycle of the tremendous amounts of information crucial to their day-to-day operations. This data arrives from a multitude of sources, and the agility derived from process integration across the enterprise provides a competitive advantage. For this reason, many organizations use application platforms to automate their business logic.

IT managers are struggling with an explosive growth of information at the same time that government regulations have imposed strict records retention requirements on companies in many industries. Businesses desire to store this incoming data through conventional file system interfaces for ease of integration with existing solutions, but also need to track and verify data at an object level throughout their business logic.

The Lifecycle Fixed Content Manager 100 Series solution uniquely suits these business needs. Lifecycle Fixed Content Manager 100 Series solution's standards-based architecture provides access to a scalable, reliable and secure storage system through conventional file-based interfaces. The Lifecycle Fixed Content Manager 100 Series solution extends this with easily accessible WORM functionality to meet both government regulations and industry best practices for records retention. The content certificates provided by the Lifecycle Fixed Content Manager 100 Series solution allow additional extrinsic validation of its non-erasable, non-rewritable nature, and allow access to object storage internals for business logic integration.

2. Retention management issues and concerns

To meet business needs and regulatory compliance, many customers need to implement processes and systems that provide verification that the business is being managed properly. Part of this verification is the ability to retain critical business documents over many years and to be able to access these when required.

But how do businesses provide verification that historical documents have not changed? This is relatively easy to prove with paper documents as tampering is usually easy to detect (although paper documents are easy to destroy). With electronic records there is a challenge to demonstrate that tampering or destruction has not taken place.

The Lifecycle Fixed Content Manager 100 Series solution helps by providing tamper-proof storage of fixed content data. This is provided primarily in three ways:

- By providing the ability to mark data as read-only
- By providing each record with a unique content certificate ("fingerprint") that cannot be altered
- By continually running automated verification processes to confirm that data has not been altered

The purpose of this application note is to outline how the implementation of content certificates within the Lifecycle Fixed Content Manager 100 Series solution contributes to the content verification process.

3. Lifecycle Fixed Content Manager 100 Series solution

The Lifecycle Fixed Content Manager 100 Series solution solves enterprise requirements for WORM storage, as described in the StorageTek® technical paper, "The Lifecycle Fixed Content Manager 100 Series solution: archival WORM storage on magnetic disks." The Lifecycle Fixed Content Manager 100 Series solution is ideally suited to compliant storage due to its object storage architecture, which provides inherent data verifiability down to the block level.

Each block of data, and the collection of blocks that then make up a file, are identified by unforgable 256-bit fingerprints. This allows for greater verifiability during write, instant detection of tampering attempts and efficient validation of stored data. Similarly, the Lifecycle Fixed Content Manager 100 Series solution is ideally suited for WORM storage, as existing applications can access Lifecycle Fixed Content Manager 100 Series solution storage through its conventional file system interface, and a simple and standard WORM protocol is used to protect files against modification. Besides the intrinsic verification provided by object storage that data has been correctly recorded and has not been since modified, Lifecycle Fixed Content Manager 100 Series solution allows extrinsic verification of data by use of content certificates.

4. Content certificates

For any given file, the corresponding content certificate is retrieved directly through the file system interface. This is done through an extensible interface for accessing a file's object metadata. Within every directory in a Lifecycle Fixed Content Manager 100 Series solution WORM file system there exists a synthetic subdirectory named .object. While this subdirectory does not appear in directory listings, it can be accessed by all standard means. Within this subdirectory is a subdirectory named cert, and within this subdirectory exist files of the same names as all files in the top directory. Each of these files in the subdirectory is the content certificate for the correspondingly named file in the top directory. (Content certificates are not presented for directories.) For example, consider the file **`/Finances/2004Taxes.xls`**. The content certificate for this file can be found by reading **`/Finances/.object/cert/2004Taxes.xls`**. Listing the directory **`/Finances/.object/cert`** would present all of the content certificates for files in the parent directory **`/Finances`**. Content certificates are a representation of the internal structure of data, conveniently presented through the file system interface. Although these content certificates appear as files, they are not documents that are stored separately within the system. When a content certificate is read, the system instantly validates the object and presents the certificate for the object as stored. The integrity of the file can then be easily verified by a simple comparison against applicable sections of a previously stored certificate. For business applications that desire to be object-aware, Lifecycle Fixed Content Manager 100 Series solution content certificates provide yet more power without requiring them to write to proprietary software libraries. Certificates in the file system are presented as structured, XML documents. Use of XML allows content certificate manipulation to be easily integrated with leading enterprise business logic tools. The provided XML schemas and descriptions in this document allow a business to unlock the full power of Lifecycle Fixed Content Manager 100 Series solution content certificates.

4.1 Extensible Markup Language

The Extensible Markup Language, XML [1], provides a generalized way for programs to exchange data. It is a language for describing object-oriented information (called *documents*) in a simple, formal and concise manner. Developed under the auspices of the World Wide Web Consortium (W3C), it aims to define a syntax for information interchange without defining the semantics of that information. In this sense it is a *meta-language*; it provides a framework in which information languages can be developed. In the past six years, XML has become widespread in enterprise software and Web services. Instead of inventing a new data encoding for each new type of document, XML allows the application writer to simply export their hierarchically structured internal objects to a flat file in a standard manner. XML does little to define the meaning of a document, however, so while one application can read in a document written by another application and verify that it is syntactically well-formed, it can not necessarily perform any operations with that data.

4.2 XML schema

XML provides a rudimentary mechanism for defining the structure of documents, the *document type definition*. XML Schema [2] is a more powerful language in which the structure and meaning of XML documents can be described. The XML Schema standard defines a language in which document structure is defined. It allows constraints to be described, which documents can then be tested against for validity. These constraints are with regards to structure (e.g. what attributes objects are allowed, and what objects they can contain) and type (e.g. what data type and range attributes and content may have). An XML Schema is a way of describing application business logic in a way that can be enforced when manipulating the data. By defining and using a schema for XML documents, data can thus be shared more effectively between applications. An application can read a document and instantly know if it is a valid piece of data to operate upon. Furthermore, since schemas are extensible, documents can be easily made backwards compatible.

5. Instance document format

```
<?xml version="1.0"?>
<ContentCert xmlns="http://www.example.com/schema/ContentCert"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.example.com/schema/ContentCert
http://www.example.com/schema/ContentCert.xsd">

<volume>C34242B7AF4E6B9A</volume>
<namespace>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB=<
/namespace>
<handle>XG5ld2NvbW1hbmR7XFBCc2VjdGlvbn1bMV17XHJlZnN=</handle>
<version>XBjb3VudGVy=</version>
<Stream number="0">
<immediate>
<content>e3NlY31cb2xkc2VjdGlvbltcdGhlc2VjIH4gXG==</content>
</immediate>
</Stream>

<Stream number="1">
<direct>
<blockName>ZG93Ym94e1xjb2xvcmJveHtzZWNibHVlfXtcbWFrZWJ=
</blockName>
</direct>
</Stream>

<WORMCompatible enterprise="true">
<protected expiry="2004-12-31T08:00:00Z"/>
</WORMCompatible>

</ContentCert>
```

Figure 1. *Content certificate. Each file in a WORM file system has a content certificate. This provides information that allows an application to verify that the file is on a particular volume (independent of the file system name), what kind of volume (e.g., compliance) and contains information that, if recorded, can be used later to prove to a third party that the contents of the file have not changed.*

Figure 1 presents an example of a Lifecycle Fixed Content Manager 100 Series solution content certificate. Let us review this line by line.

5.1 Preamble

```
<?xml version="1.0"?>
```

This identifies the content certificate as an XML document. It is a standard component of any XML document.

```
<ContentCert xmlns="http://www.example.com/schema/ContentCert"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.example.com/schema/ContentCert
http://www.example.com/schema/ContentCert.xsd">
```

The root element in a Lifecycle Fixed Content Manager 100 Series solution content certificate is the **`ContentCert`** element. The **`ContentCert`** element identifies that this document conforms to an XML Schema, and identifies the schema against which it can be validated. The **`ContentCert`** element contains, in order, a volume, a namespace, a handle, a version, one or more Streams, and an optional **`WORMCompatible`** element. Future product versions may add additional attributes in order to expose additional functionality.

5.2 Object locators

```
<volume>C34242B7AF4E6B9A</volume>
```

Although the lower levels of the Lifecycle Fixed Content Manager 100 Series storage system present a flat-named object space in which objects are located, much like i-nodes in a UNIX file system or blocks on a disk, at a higher level these objects are grouped into hierarchical file systems. In Lifecycle Fixed Content Manager 100 Series solution, these file systems are called *volumes*. While the same data may appear in multiple volumes, and data coalescence (savings in space by storing redundant data only once) will be seen, a given file instance appears in only one volume. This allows verifiability that the file being inspected is in the intended WORM volume.

```
<namespace>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAB=<
/namespace>
```

```
<handle>XG5ld2NvbW1hbmR7XFBCc2VjdGlvbn1bMV17XHJlZnN=</handle>
```

These two elements, namespace and handle, uniquely identify this file within the Fixed Content Manager 100 Series storage system. Together, they represent the logical location within the Fixed Content Manager 100 Series solution that the file is stored, much like a conventional disk i-node. During its life, a file will retain the same name space and handle even as its content changes. A object-aware application can consider these two together as the *key* into a database of mutable objects.

```
<version>XBjb3VudGVy=</version>
```

Unlike a conventional disk, all objects within the Fixed Content Manager 100 Series storage system are versioned. As a file is modified, its name space and handle remain the same, but the version identifier for the most recent version changes. Multiple versions of a file are retained based on snapshot and thinning policies within the system. Inspecting the content certificate for the same file at different points in time, or in different snapshots, might yield the same name space and handle but different versions, provided that the file was not yet WORM protected and thus was allowed to be modified.

5.3 Stream descriptors

```
<Stream number="0">
<immediate>
<content>e3NlY31cb2xkc2VjdGlvbltcdGhlc2VjIH4gXG==</content>
</immediate>
</Stream>
<Stream number="1">
<direct>
<blockName>ZG93Ym94e1xjb2xvcmJveHtzZWNibHVlfXtcbWFrZWJ=
</blockName>
</direct>
</Stream>
```

Here we reach the object core of the Lifecycle Fixed Content Manager 100 Series storage system. Each version of a file consists of several linear streams of data. These are stored in a way such that any modification of the data stream will instantly be identifiable in the stream descriptor. For verification of the integrity of data over time, these streams are of the utmost concern.

In this example, we see a file which consists of two streams, numbered 0 and 1. Some files, particularly those on which Lifecycle Fixed Content Manager 100 Series solution achieves increased coalescence by specific knowledge of the data format, may have additional streams.

The **Stream** element has one attribute, number. For file storage, stream number 0 is reserved for file metadata. Information such as file owner, group, mode, modification time, and access time is stored in this data stream. If this stream element is the same as one previously recorded, one can be assured that no file metadata has changed.

Do note that because the mechanism for extending the retention period of a protected file, as described in the "Lifecycle Fixed Content Manager 100 Series solution" white paper, involves modifying the last access time of a file, extending the retention period will cause the file metadata to change. Application writers using this stream to verify content integrity should keep this in mind.

Additional streams present in an object contain the actual file data. This may be stored in a single stream, numbered 1, or split among multiple streams in order to more efficiently store data for coalescence. If these stream descriptor elements match those in a stored content certificate, an application can be assured that no modification to the file data has occurred. Due to the cryptographic security of the Lifecycle Fixed Content Manager 100 Series solution block naming, the modification of even a single bit in any data stream would unavoidably change the stream descriptor.

The **Stream** element contains one of six different elements, each describing a specific data stream structure. Each type offers the same integrity guarantees, and when storing data, the Lifecycle Fixed Content Manager 100 Series solution system chooses the stream type that will most efficiently store the application data with the desired level of security. These stream types are described in more detail below.

Because the same data may be stored in different ways depending on application requirements for compression, encryption, and coalescence, two stream descriptors being different is not a guarantee that the stored data is different. If two stream descriptors are the same, however, this is an unforgable guarantee that the data is the same.

5.4 WORM status

```
<WORMCompatible enterprise="true">
<protected expiry="2004-12-31T08:00:00Z"/>
</WORMCompatible>
```

The final element that appears in a content certificate indicates the WORM status of the file in question. The presence of a **WORMCompatible** element indicates that the file is stored on a Lifecycle Fixed Content Manager 100 Series solution WORM volume, presenting a standard and compatible WORM protocol through the file-based interface. This element has one attribute, enterprise. If enterprise is present and has the value true this indicates that the WORM file system is an enterprise WORM file system. If this attribute is not present, or false, this indicates that the WORM file system is a compliance WORM file system.

The **WORMCompatible** element may contain one of three elements, indicating the current protection status of the file. The element **unprotected** indicates that the file is not currently under WORM protection, and may be modified at any time. The element **protected**, as seen above, indicates that the file is under WORM protection, and that this protection is scheduled to **expire** at the time indicated in the expiry attribute. Finally, the element **expired** indicates that the WORM protection for this file has expired and the file can now be deleted, but still not modified.

This element should be used with care when verifying data integrity against a stored certificate, as the protection status of a file will change when its protection expires, even though the data has not been modified.

6. Integrating content certificates

By the nature of being XML documents, Lifecycle Fixed Content Manager 100 Series solution content certificate management can be easily integrated into virtually any application. No tokenization or parsing routines need be written by the implementor, speeding development and ensuring compatibility. Most standard application frameworks include XML document tools, and additional powerful tools (such as the free Apache Xerces parsers, available at **http://xml.apache.org/**) are also available.

An XML parser will provide the application with an object hierarchy describing the Lifecycle Fixed Content Manager 100 Series solution content certificate. An overview of this object structure is provided in the previous section, and more detail can be found in the XML Schema descriptions found in the Appendices. **Machine-readable versions of these schemas are available from StorageTek at the URLs indicated in the attached schemas.** It is not necessary for the application to use an XML parser which also provides an XML Schema processor, however doing so will ease integration by providing strict data typing and additional validation of processed documents.

For most uses, an application concerned with verifying the integrity of files stored in the Lifecycle Fixed Content Manager 100 Series system will desire to retrieve content certificates for files immediately after they are stored and keep this information in an outside database for later comparison against the storage system.

6.1 Verifying file location (serial number)

One common operation an application may wish is to perform is to verify that a file being examined is the correct file; such an operation is often referred to as a serialization requirement on the storage system.

In Lifecycle Fixed Content Manager 100 Series solution, this can be done at the file system level by verifying the file path name. Because a protected file cannot be moved, modified or replaced, its path name represents a unique, unmodifiable identifier for the file.

With Lifecycle Fixed Content Manager 100 Series solution content certificates, absolute file identification can also be performed at the object level by an application that wishes to do so. The **`namespace`** and **`handle`** values found in the content certificate uniquely locate a file within the Lifecycle Fixed Content Manager 100 Series storage system. Together, they form a 512-bit identifier that is guaranteed to correspond to only one file.

Further, the version value uniquely identifies a specific revision of the file in question. Application implementors should note that operations that extend the retention period of a file modify the metadata indicating the retention period, which may cause the version identifier to change. Thus, namespace and handle should be considered the object-level serial number for a specific protected file.

6.2 Verifying file data has not changed

Most applications interacting with Lifecycle Fixed Content Manager 100 Series solution content certificates will wish to use them to validate that, beyond the inherent protection guarantees the Lifecycle Fixed Content Manager 100 Series system offers, their data has not been modified.

This can be done by examining the stored **`Stream`** objects, for all **`Streams`** with the attribute number set to greater than 0, and comparing these against the stored objects. If these are equal, the application can be assured that the file data has not changed. Due to the cryptographic fingerprinting operations described in Section 8, a modification of even a single bit in any of the data would cause these stream descriptors to be entirely different.

The stream descriptor numbered 0 contains metadata information only.

6.3 Verifying file metadata has not changed

Some applications may wish to verified that file metadata has not changed. This can be done by comparing the **`Stream`** object with a number attribute of 0 against the stored object.

This can be done, for example, to verify that the retention period on this file has not been extended since it was originally protected. Extending the retention period of a file modifies metadata indicating the protection period of the file, as this is presented through the file system interface as the “last access time” of the file. The retention period for a file can never be shortened in a Lifecycle Fixed Content Manager 100 Series storage system.

6.4 Confirming file protection status

Due to the nature of the standard, compatible file system protocol used to protect files in Lifecycle Fixed Content Manager 100 Series solution against modification or erasure, it is not always possible to tell by passive observation that a file has been protected. A file that has been initially created with read-only permissions will appear with read-only permissions, but is not protected against modification or erasure. In competitive products also implementing the Lifecycle Fixed Content Manager 100 Series solution file system protocol for data protection, the only way to verify that a file is read-only is by attempting to modify it. This is not a desirable mechanism.

Lifecycle Fixed Content Manager 100 Series solution content certificates allow verification of file protection without the need to attempt a modifying operation. To do so, an application need only inspect the **`WORMCompatible`** object in the document. If this contains the **`protected`** object, the application is assured that the file in question is under the highest level of protection against modification or erasure. The **`protected`** object also indicates, by its expiry attribute, the time until which the file is protected.

If the file has not yet been protected by the Lifecycle Fixed Content Manager 100 Series storage system, the **`WORMCompatible`** object will contain an **`unprotected`** object. Regardless of the permissions viewed through the file system interface, this indisputably indicates the disposition of the file in the storage system. If the file is in this state, it can easily be protected as described in the "Lifecycle Fixed Content Manager 100 Series solution" document, by setting the permissions to writable and then back to read-only.

If the file was protected by the Lifecycle Fixed Content Manager 100 Series storage system but the protection period has since lapsed, the **`WORMCompatible`** object will contain an **`expired`** object. This indicates that the file in question is no longer protected against erasure, although it still cannot be modified.

6.5 Identifying linked files

There are more advanced analyses an application can perform by inspecting the content certificate data. One example of this is the identification of linked files in the file system. Conventional file systems allow the concept of two file names that refer to the same data, where modifying one file necessarily modifies the other. On UNIX-like systems, these are often called "hard links" in order to differentiate them from "symbolic links" which occur at a higher level in the system. ("Symbolic links" are more equivalent to Microsoft Windows "shortcuts" or Apple Macintosh "aliases.")

Lifecycle Fixed Content Manager 100 Series solution content certificates allow an application to easily identify multiple-linked files. As described above, the **`namespace`** and **`handle`** objects uniquely identify a file in the Lifecycle Fixed Content Manager 100 Series storage system. If two content certificates indicate the same **`namespace`** and **`handle`**, this is proof that they identify the same file.

6.6 Identifying identical data

An application can use Lifecycle Fixed Content Manager 100 Series solution content certificates to usually, though not always, identify multiple files containing identical data. Given a database of content certificates, identical files can often be located much more easily than reading the full data for every file.

If all of the **`Streams`**, numbered greater than 0, of a file are identical, this is unrefutable proof that the two files contain identical data. The information contained in a stream descriptor only identifies one byte stream of data within the storage system, and any modification of that data would unavoidably change the stream descriptor.

Note that the stream descriptors for two files being different does not provide that the files are, in fact, different. This is due to the underlying nature of the Lifecycle Fixed Content Manager 100 Series storage system. The content of a stream descriptor depends on the data in that stream and also the manner in which that stream was segmented into blocks. The same data segmented differently would have a different stream descriptor.

A Lifecycle Fixed Content Manager 100 Series solution access node has a default strategy for segmenting files; they are separated into 64-kilobyte blocks for ease of management and sub-file coalescence (a savings in space by storing redundant data only once) on unknown data. For certain data types, such as those produced by certain backup and archiving products, the Lifecycle Fixed Content Manager 100 Series solution portal will segment the file differently in order to achieve increased coalescence on the files included in the archive.

If an identical archive was written twice, one with and one without the coalescence option enabled, the stream descriptors would be different even though the data is the same. The first instance would be stored by the Lifecycle Fixed Content Manager 100 Series solution access node with the default segmentation behavior, and the second instance would be written with the intelligent segmentation. In cases such as these, comparing stream descriptors to locate multiple identical files in the storage system would not identify all such files.

Two stream descriptors being the same is an unquestionable guarantee that the files are the same. The Lifecycle Fixed Content Manager 100 Series system will never resegment a file once it has been written. Two stream descriptors being different, however, is not an assurance that the files are different.

7. Opportunities

Lifecycle Fixed Content Manager 100 Series solution uniquely suits modern business needs for the management and storage of a quickly growing volume of compliant records. Lifecycle Fixed Content Manager 100 Series solution allows for simple integration with existing and new applications by providing both record storage and retention management through a conventional file system interface. This easy-to-use interface is backed by Lifecycle Fixed Content Manager 100 Series solution’s scalable, reliable and secure object storage infrastructure. The content certificates presented by the Lifecycle Fixed Content Manager 100 Series storage system allow for extrinsic verification of compliant record retention, and provide a mechanism by which records stored in Lifecycle Fixed Content Manager 100 Series solution can be easily integrated into growing enterprise business logic.

Appendix A: How data is stored in Lifecycle Fixed Content Manager 100 Series solution

Data blocks

The data block is the atomic unit of information stored in the Lifecycle Fixed Content Manager 100 Series solution repository. Data blocks contain small linear chunks of data that may represent a data stream on their own, or be assembled into larger linear byte streams as described below. The data within a data block may be encoded in one of several ways, and not all formats are usable in all stream types.

Every data block has an associated 256-bit data name that uniquely identifies that data block, and only that data block. Lifecycle Fixed Content Manager 100 Series solution data blocks have the self-naming property, which is to say that the data name is generated deterministically from the data block. A SHA-256 hash is performed over the complete data block, and the result is taken as the data name, or block name. All data names must be generated in this fashion. This name generation guarantees that a given data block has only a single name, and that within Lifecycle Fixed Content Manager 100 Series solution a given data name uniquely identifies a single data block.

Two conflicting motivations influence the recommended data block size. On one hand, the Lifecycle Fixed Content Manager 100 Series system introduces overhead in its use of data names and stream structures, so it is desirable to have data blocks that are as large as feasible. On the other hand, we wish to be able to provide acceptable performance for live modifications to files, such as through an NFS portal, so data blocks cannot be so large as to introduce unacceptable latency into file system operations. To balance these concerns, the maximum data size before encoding is 64 kilobytes (65,536 bytes).

Data objects

While the data block is the atomic unit of information stored in the Lifecycle Fixed Content Manager 100 Series solution repository, data blocks cannot be retrieved directly by their data name. Instead, data blocks must be retrieved by reference to their location within a *data object*.

A data object represents some entity, such as a file, for the duration of its existence. It provides not just the current view of that entity, but also an historical record of previous versions of the entity. As such, a data object consists of multiple immutable *object versions*.

An object version represents a snapshot of some entity at a given point in time. Object versions are immutable; once written, they cannot be modified. To change the current view of an entity in the repository, a new object version is deposited and added to the data object's list of versions.

An object version consists of multiple parallel byte streams of data, called *streams* and described in detail below. This allows an object version to represent multiple linear byte streams that must be synchronized in time for a coherent snapshot, such as a multi-forked file. Apple Macintosh files have multiple data forks, and Microsoft supports multiple data streams in NTFS. Up to 2^{15} independent streams may appear in an object version, identified uniquely by any of the integers from 0 to 2^{15} -1, inclusive. Stream numbers need not be consecutive.

Streams

All application data that is stored within a Lifecycle Fixed Content Manager 100 Series solution repository is addressable within some linear byte stream, a stream. Streams consist of short data or references to data blocks, and a number of stream types exist to suit different needs.

Streams always appear within the context of a data object version and have a stream number uniquely identifying them within that object version. Stream numbers may range from 0 to 2^{15} - 1, inclusive. By convention, stream 0 is reserved for format metadata (if present), and stream 1 is reserved for the primary data stream.

Streams may be encrypted or unencrypted. For streams that contain references to data blocks, all referenced blocks must have the same encryption state as the stream.

Immediate

Immediate streams are used to store extremely small amounts of data for efficiency of storage and access. The largest amount of data that may be stored in an immediate stream is 256 bytes. This is an unencrypted stream type, so data stored in this manner is visible to the server.

Immediate-encrypted

Immediate-encrypted streams are used to securely store extremely small amounts of data for efficiency of storage and access. The recommended largest amount of unencrypted data that may be stored in an immediate stream is 256 bytes.

Direct

Direct streams are used to store data that is short enough to fit within a single data block. A direct stream references a single data block in the repository by data name. This data name is stored in the object metadata as part of the stream descriptor. This is an unencrypted stream type. As such, the referenced data block must be of an encoding that does not require a key.

Direct-encrypted

Direct-encrypted streams are used to securely store data that is short enough to fit within a single data block. A direct-encrypted stream references a single data block in the repository by data name. This is an encrypted stream type. As such, the referenced data block must be of an encoding that requires a key.

Indirect

Indirect streams are used to store larger amounts of data, and to allow for coalescence based on portal knowledge of where to place block boundaries. An indirect stream references many data blocks in the repository by data name. This is done by constructing one or more index blocks to describe the stream structure. Additionally, key blocks are constructed that provide additional metadata, such as block lengths, to the access node.

An index block for an indirect stream contains references by name for one or more data blocks in the order they appear in the stream, a reference by name for a key block providing additional information such as data block lengths, and a reference by name to another index block if this is not the first block of the stream. Index blocks are reverse chained; the final index block (referencing the final data blocks of the stream) contains a reference to the preceding index block, and so on until the initial (zero-indexed) index block is reached. This allows one to easily deposit a large stream of data, or append to a long indirect stream by depositing a new final index block that references the previous one.

This hierarchical data structure allows the entire data in the stream, of arbitrary length, to be referenced by only one piece of data; the index block name. Any modification to any data in the entire stream will cause this index block name to change, but will not require recomputing the hashes for all data in the stream.

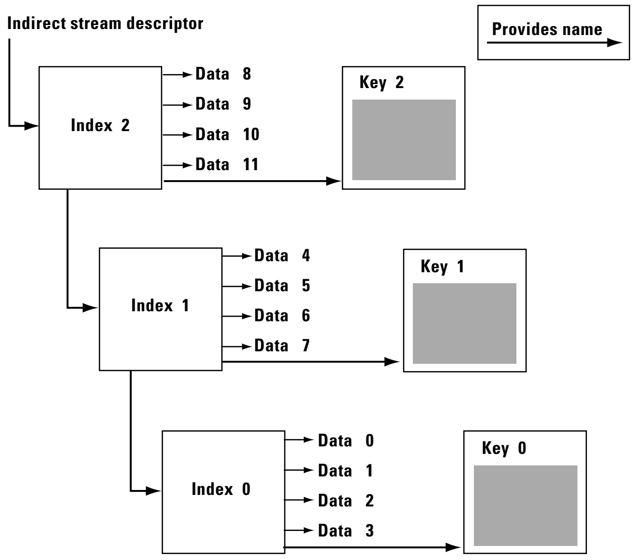

Figure 2. *shows the structure of an indirect stream. Solid arrows denote that some entity provides the name for some data block.*

Indirect-encrypted

Indirect-encrypted streams are used to securely store larger amounts of data. An indirect-encrypted stream references many data blocks in the repository by data name, and easily manages all of the associated data keys. Constructing one or more index blocks and key blocks to describe the stream structure does this. Keying information is kept separate from the index information for data safety. Even in an indirect-encrypted stream, the server must have access to the unencrypted index block contents so that it may retrieve data blocks by reference. However, the server must never have access to the unencrypted data keys for the associated blocks.

Appendix B: Content certificate schema

```
<?xml version="1.0"?>
<xsd:schema xmlns="http://www.example.com/schema/ContentCert"
         targetNamespace="http://www.example.com/schema/ContentCert"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:cc="http://www.example.com/schema/ContentCert"
         elementFormDefault="qualified"
         version="1.0">
<xsd:include
schemaLocation="http://www.example.com/schema/Stream.xsd"/>

<xsd:annotation>
         <xsd:documentation xml:lang="en">
         Definition of LFCM 100 content certificates
         </xsd:documentation>
         </xsd:annotation>

<xsd:complexType name="EmptyType">
         <xsd:annotation>
                  <xsd:documentation xml:lang="en">
                  WORM records that are not protected or whose
                  protection has expired are represented by the
                  elements "unprotected" and "expired," which have no
                  attributes and no content. This EmptyType provides
                  that type.
                  </xsd:documentation>
         </xsd:annotation>
</xsd:complexType>

<xsd:simpleType name="dateTimeZulu">
         <xsd:restriction base="xsd:dateTime">
                  <xsd:pattern value="\d{4}-\d{2}-
                           \d{2}T\d{2}:\d{2}:\d{2}Z">
                           <xsd:annotation>
                                    <xsd:documentation xml:lang="en">
                                    This type provides a restriction upon the
                                    XML Schema dateTime type, bringing it
                                    closer to the ISO8601 base. Years greater
                                    than 9999 are disallowed, negative
                                    years are disallowed, hours, minutes and
                                    seconds must be specified, and a time
                                    zone of UTC is required. (The pattern is
                                    simple because it's a restriction on
                                    a type that will already limit us to
                                    valid values for months, hours, etc.)
                                    </xsd:documentation>
                           </xsd:annotation>
                  </xsd:pattern>
         </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:complexType name="ProtectedType">
        <xsd:annotation>
                <xsd:documentation xml:lang="en">
                A WORM record currently being protected is
                represented by a "protected" element, which has an
                attribute indicating the period of protection.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:attribute name="expiry" type="cc:dateTimeZulu"
        use="required"/>
</xsd:complexType>

<xsd:element name="ContentCert">
        <xsd:complexType>
                <xsd:sequence>

                        <xsd:element name="volume">
                                <xsd:simpleType>
                                <xsd:annotation>
                                        <xsd:documentation
                                                xml:lang="en">
                                        Volume identifiers are
                                        constrained to 64 bits, and
                                        are commonly expressed as
                                        hexadecimal values.
                                        </xsd:documentation>
                                </xsd:annotation>
                                <xsd:restriction
                                        base="xsd:hexBinary">
                                        <xsd:length value="8"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:element>

                <xsd:element name="namespace"
                        type="ThirtyTwo"/>

                <xsd:element name="handle" type="ThirtyTwo"/>

                <xsd:element name="version">
                        <xsd:simpleType>
                                <xsd:annotation>
                                        <xsd:documentation
                                                xml:lang="en">
                                        Version numbers may range
                                        from 0 to 255 bytes in
                                        length, and are expressed as
                                        base64-encoded values.
                                        </xsd:documentation>
                                </xsd:annotation>
                                <xsd:restriction
                                        base="xsd:base64Binary">
                                        <xsd:minLength value="0"/>
                                        <xsd:maxLength value="255"/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:element>

                <xsd:element ref="Stream" minOccurs="1"
                        maxOccurs="32768"/>
                <xsd:element name="WORMCompatible"
                        minOccurs="0">
                        <xsd:complexType>
                                <xsd:choice>
                                        <xsd:element
                                                name="unprotected"
                                                type="EmptyType"/>
                                        <xsd:element name="protected"
                                                type="ProtectedType"/>
                                        <xsd:element name="expired"
                                                type="EmptyType"/>
                                </xsd:choice>
                                <xsd:attribute name="enterprise"
                                        type="xsd:boolean"
                                        default="false"/>
                                        </xsd:complexType>
                                </xsd:element>

                        </xsd:sequence>
                </xsd:complexType>

                <xsd:key name="StreamID">
                        <xsd:annotation>
                                <xsd:documentation xml:lang="en">
                                Two streams within a single content certificate
                                may not have the same stream number.
                                </xsd:documentation>
                        </xsd:annotation>
                        <xsd:selector xpath="cc:Stream"/>
                        <xsd:field xpath="@number"/>

                </xsd:key>

        </xsd:element>

</xsd:schema>
```

Appendix C: Stream descriptor schema

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
version="1.0">

                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Stream types for LFCM 100 Content certificates.
                        </xsd:documentation>
                </xsd:annotation>

        <xsd:simpleType name="ThirtyTwo">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        Many values in content certificates are constrained to
                        be 256 bit values, commonly represented in base64
                        encoding.
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:restriction base="xsd:base64Binary">
                        <xsd:length value="32"/>
                </xsd:restriction>
                </xsd:simpleType>

        <xsd:simpleType name="StreamNum">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        The range of stream numbers in SSNAP is that of the
                        non-negative short integers, i.e. from 0 to 2**15-1.
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:restriction base="xsd:short">
                        <xsd:minInclusive value="0"/>
                </xsd:restriction>
        </xsd:simpleType>

        <xsd:element name="Stream">
                <xsd:annotation>
                        <xsd:documentation xml:lang="en">
                        The Stream element has a required attribute "number"
                        indicating the stream number within the object of this
                        stream. The element contains a single stream
                        descriptor, which may be of any of the six allowed
                        SSNAP stream types.
                        </xsd:documentation>
                </xsd:annotation>
                <xsd:complexType>
                        <xsd:choice>
                                <xsd:element ref="immediate"/>
                                <xsd:element ref="immediateEncrypted"/>
                                <xsd:element ref="direct"/>
                                <xsd:element ref="directEncrypted"/>
                                <xsd:element ref="indirect"/>
                                <xsd:element ref="indirectEncrypted"/>
                        </xsd:choice>
                        <xsd:attribute name="number" type="StreamNum"
                                use="required"/>
                </xsd:complexType>
</xsd:element>
```

```
<xsd:element name=”immediate”>
        <xsd:annotation>
                <xsd:documentation xml:lang=”en”>
                Immediate streams are used to store extremely small
                amounts of data directly in the object metadata
                records, for efficiency of storage and access. The
                largest amount of data that may be stored in an
                immediate stream is 256 bytes. By convention, immediate
                data is represented in a base64 encoding.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name=”content”>
                                <xsd:simpleType>
                                        <xsd:restriction
                                                base=”xsd:base64Binary”>
                                                <xsd:minLength value=”0”/>
                                                <xsd:maxLength value=”256”/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:element>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>

<xsd:element name=”immediateEncrypted”>
        <xsd:annotation>
                <xsd:documentation xml:lang=”en”>
                Immediate-encrypted streams are used to securely store
                extremely small amounts of data directly in the object
                metadata records, for efficiency of storage and access.
                The largest amount of post-encryption data that may be
                stored in an immediate-encrypted stream is 272 bytes; the
                additional 16 bytes account for the padding incurred by
                an exactly 256 byte unencrypted stream.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
                <xsd:sequence>
                <xsd:element name=”content”>
                        <xsd:simpleType>
                                <xsd:restriction
                                        base=”xsd:base64Binary”>
                                        <xsd:minLength value=”0”/>
                                        <xsd:maxLength value=”272”/>
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:element>
                <xsd:element name=”secretID”
                        type=”xsd:unsignedInt”/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="direct">
        <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Direct streams are used to store data that is short
                enough to fit within a single data block. A direct
                stream references a single data block in the repository
                by data name. This data name is stored in the metadata
                as part of the stream descriptor.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="blockName" type="ThirtyTwo"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>

<xsd:element name="directEncrypted">
        <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Direct-encrypted streams are used to securely store
                data that is short enough to fit within a single data
                block. A direct-encrypted stream references a single
                data block in the repository by data name. This is an
                encrypted stream type.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="blockName" type="ThirtyTwo"/>
                        <xsd:element name="blockKey" type="ThirtyTwo"/>
                        <xsd:element name="secretID"
                                type="xsd:unsignedInt"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>

<xsd:element name="indirect">
        <xsd:annotation>
                <xsd:documentation xml:lang="en">
                Indirect streams are used to store larger amounts of
                data, and to allow for coalescence based on client
                knowledge of where to place block boundaries. An
                indirect stream references many data blocks in the
                repository by data name. This is done by constructing
                one or more index blocks to describe the stream
                structure. Additionally, key blocks are constructed
                that provide additional metadata, such as block
                lengths, to the client.
                </xsd:documentation>
        </xsd:annotation>
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="indexName" type="ThirtyTwo"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="indirectEncrypted">
         <xsd:annotation>
                  <xsd:documentation xml:lang="en">
                  Indirect-encrypted streams are used to
securely store
                  larger amounts of data. An indirect-
encrypted stream
                  references many data blocks in the
repository by data
                  name, and easily manages all of the
associated data
                  keys. This is done by constructing one or
more index
                  blocks and key blocks to describe the
stream structure.
                  </xsd:documentation>
         </xsd:annotation>
         <xsd:complexType>
                  <xsd:sequence>
                           <xsd:element name="indexName"
type="ThirtyTwo"/>
                           <xsd:element name="indexKey"
type="ThirtyTwo"/>
                           <xsd:element name="keyKey"
type="ThirtyTwo"/>
                           <xsd:element name="secretID"
                           type="xsd:unsignedInt"/>
                  </xsd:sequence>
         </xsd:complexType>
</xsd:element>

</xsd:schema>
```

End notes

[1] W3C, "Extensible Markup Language (XML) 1.0 (Third Edition)," available at http://www.w3.org/TR/2004/REC-xml-20040204/

[2] W3C, "XML Schema Part 0: Primer," available at http://www.w3c.org/TR/xmlschema-0/

STORAGETEK®

ABOUT STORAGETEK®

Storage Technology Corporation (NYSE: STK) is a $2 billion global company that enables businesses, through its information lifecycle management strategy, to align the cost of storage with the value of information. The company's innovative storage solutions manage the complexity and growth of information, lower costs, improve efficiency and protect investments. For more information, visit www.storagetek.com, or call 1.800.275.4785 or 01.303.673.2800.

WORLD HEADQUARTERS

Storage Technology Corporation
One StorageTek Drive
Louisville, Colorado 80028 USA
1.800.877.9220 or 01.303.673.5151

© 2004 Storage Technology Corporation, Louisville, CO. All rights reserved. Printed in USA. StorageTek and the StorageTek logo are registered trademarks of Storage Technology Corporation. Other names mentioned may be trademarks of Storage Technology Corporation or other vendors/manufacturers. StorageTek equipment is manufactured from new parts, or new and used parts. In some cases, StorageTek equipment may not be new and may have been previously installed. Regardless, StorageTek's standard warranty terms apply, unless the equipment is specifically identified by StorageTek as "used" or "refurbished." Replacement parts provided under warranty or any service offering may be either new or equivalent-to-new, at StorageTek's option. Specifications/features may change without notice.

FT 0004 A 12/05