MarkLogic Server

Temporal Developer's Guide

MarkLogic 8
February, 2015

Last Revised: 8.0-7, August, 2017

Table of Contents

Temporal Developer's Guide

1.0 Understanding Temporal Documents

You can configure MarkLogic Server to manage and query bitemporal data. Bitemporal documents are associated with both a *valid* time that marks when a thing is known in the real world and a *system* time that marks when the thing is available for discovery in MarkLogic Server.

Bitemporal data is necessary whenever there is a requirement to maintain snapshots of a transaction across various time dimensions. For example, financial and insurance industries use bitermporal data to track changes to contracts, policies, and events in a manner that adheres to strict regulation and compliance requirements.

In this guide, the term *temporal* refers to bitemporal documents and collections.

This chapter describes the basic components used to manage temporal documents, and includes the following sections:

1.1 Terms

To understand the temporal functions, you need to understand the meaning of the following terms:

- *Instant*: an instant of time (such as "now", "12/31/2012, 01:00:00 am").
- *Period*: an anchored duration of time (e.g. December 01, 1999 through December 31, 2000, the fall semester).
- *Axis*: a named pair of range indexes that is the container for periods. Temporal documents have both a valid and system axis.
- *User-defined Time*: a time value that user provides in replacement of system start time.
- *LSQT* (Last Stable Query Time): a document with a system start time before this point can be queried and a document with a system start time after this point can be updated and ingested.
- A *split* refers to the creation of a new document that contains the same content as a previous document, but with different valid timestamps and system end time.

  Note: Document quality and permissions travel with the split. Document properties do not travel with the split.

1.2 Overview of Temporal Documents

Bitemporal data tracks information along two different time lines:

- *Valid Time*: when the information was true in the real world. Valid time may also be called application time. Valid time is provided by the user or application. The valid end time is updated by the system when the document is split.
- *System Time*: when the information was stored in the database. System time may also be called transaction time. System time is managed by the system, except in cases when the system start time is set by the application as described in "Last Stable Query Time (LSQT) and Application-controlled System Time" on page 20.

In MarkLogic, a bitemporal document is managed as a series of versioned documents in a protected collection. The 'original' document inserted into the database is kept and never changes. Updates to the document are inserted as new documents with different valid and system times. A delete of the document is also inserted as a new document. In this way, a bitemporal document can be "rolled back" to review, at any point in time, when the information was known in the real world and when it was recorded in the database.

1.3 Roles and Permissions

The `temporal-admin` or `admin` role is required to create axes, temporal collections, and otherwise configure the temporal environment.

> **Note:** Changing permissions on a temporal document only affects the latest version of the document that is created as a result of the change. All previous versions of the document maintain their original permissions.

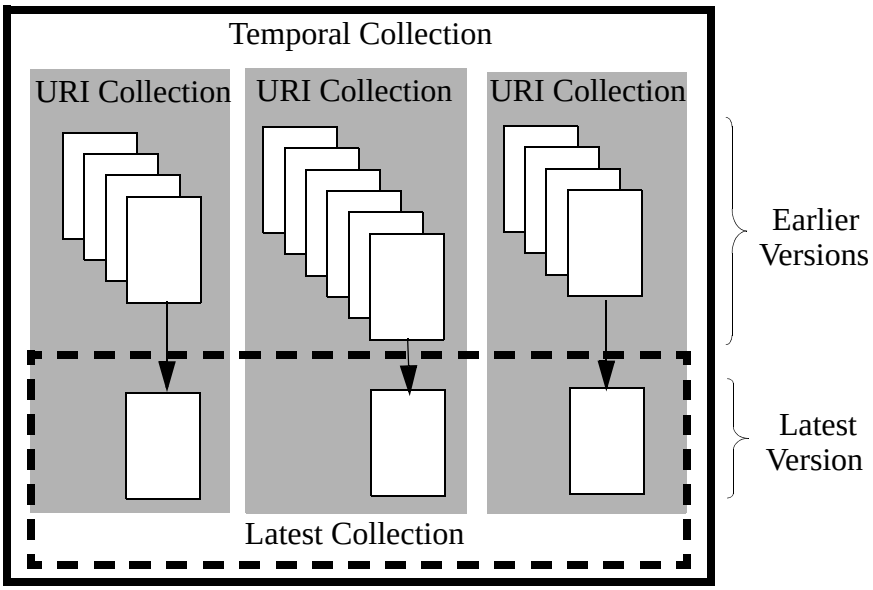
1.4 Temporal, URI, and Latest Collections

Bitemporality is defined on a protected collection, known as a *temporal collection*. A temporal collection is a logical grouping of temporal documents that share the same axes with timestamps defined by the same range indices. You can create additional temporal collections if you have documents that require a different schema for the timestamps.

When a document is inserted into a temporal collection, a *URI collection* is created for that document. When the document is updated, a new document representing the update is inserted into the document's URI collection. Any new document inserted into the temporal collection will have its own unique URI collection that will hold all of the versions of that document.

Additionally, the latest version of each document will reside in a *latest collection*.

The following illustrates how three temporal documents would be organized into the temporal, URI, and latest collections:



Note: When a temporal document is deleted, it is removed from the `latest` collection.

2.0 Quick Start

You can configure MarkLogic Server to manage and query temporal data.

This chapter walks you through the procedures for configuring the `Documents` database to store temporal documents and for inserting and querying temporal documents. The following are the main sections:

- Create the Range Indexes for the Valid and System Axes
- Create System and Valid Axes
- Create a Temporal Collection
- Insert Some Temporal Documents
- Run some Search Queries on the Temporal Documents

2.1 Create the Range Indexes for the Valid and System Axes

The valid and system axis each make use of `dateTime` range indexes that define the start and end times. For example, the following query creates the element range indexes to be used to create the valid and system axes.

JavaScript Example:

```
var admin = require("/MarkLogic/admin.xqy");
var config = admin.getConfiguration();
var dbid = xdmp.database("Documents");

var validStart = admin.databaseRangeElementIndex(
    "dateTime", "", "validStart", "", fn.false() );

var validEnd   = admin.databaseRangeElementIndex(
    "dateTime", "", "validEnd", "", fn.false() );

var systemStart = admin.databaseRangeElementIndex(
    "dateTime", "", "systemStart", "", fn.false() );

var systemEnd   = admin.databaseRangeElementIndex(
    "dateTime", "", "systemEnd", "", fn.false() );

config = admin.databaseAddRangeElementIndex(config, dbid, validStart);
config = admin.databaseAddRangeElementIndex(config, dbid, validEnd);
config = admin.databaseAddRangeElementIndex(config, dbid,
systemStart);
config = admin.databaseAddRangeElementIndex(config, dbid, systemEnd);

admin.saveConfiguration(config);
```

XQuery Example:

```
xquery version "1.0-ml";
import module namespace admin = "http://marklogic.com/xdmp/admin"
    at "/MarkLogic/admin.xqy";

let $config := admin:get-configuration()
let $dbid := xdmp:database("Documents")

let $validStart := admin:database-range-element-index("dateTime",
"", "validStart", "", fn:false() )
let $validEnd := admin:database-range-element-index("dateTime",
"", "validEnd", "", fn:false() )

let $systemStart := admin:database-range-element-index("dateTime",
"", "systemStart", "", fn:false() )
let $systemEnd := admin:database-range-element-index("dateTime",
"", "systemEnd", "", fn:false() )

let $config := admin:database-add-range-element-index($config, $dbid,
$validStart)
let $config := admin:database-add-range-element-index($config, $dbid,
$validEnd)
let $config := admin:database-add-range-element-index($config, $dbid,
$systemStart)
let $config := admin:database-add-range-element-index($config, $dbid,
$systemEnd)

return

    admin:save-configuration($config)
```

2.2 Create System and Valid Axes

Temporal documents have both a valid and system axis. Create two axes, named “valid” and “system,” each to serve as a container for a named pair of range indexes.

JavaScript Example:

```
var temporal = require("/MarkLogic/temporal.xqy");

var validResult = temporal.axisCreate(
    "valid",
    cts.elementReference(fn.QName("", "validStart")),
    cts.elementReference(fn.QName("", "validEnd")));

var systemResult = temporal.axisCreate(
    "system",
    cts.elementReference(fn.QName("", "systemStart")),
    cts.elementReference(fn.QName("", "systemEnd")));
```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

temporal:axis-create(
    "valid",
    cts:element-reference(xs:QName("validStart")),
    cts:element-reference(xs:QName("validEnd")));

xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

temporal:axis-create(
    "system",
    cts:element-reference(xs:QName("systemStart")),
    cts:element-reference(xs:QName("systemEnd")));
```

2.3 Create a Temporal Collection

Create a temporal collection, named “kool,” that uses the previously created “system” and “valid” axes.

JavaScript Example:

```
var temporal = require("/MarkLogic/temporal.xqy");

var collectionResult = temporal.collectionCreate(
"kool", "system", "valid");
```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
     at "/MarkLogic/temporal.xqy";

temporal:collection-create("kool", "system", "valid")
```

Note: Axis and temporal collection names are case-sensitive.

2.4 Insert Some Temporal Documents

Insert some documents into the temporal collection. In this example, a stock trader, John, places an order to buy some stock. The record of the trade is stored as a bitemporal document, as follows:

1. The stock of KoolCo is trading around $12.65. John places a limit order to buy 100 shares of the stock for $12 at 11:00:00 on 3-Apr-2014 (this is the valid time). The document for the transaction is recorded in the broker’s database at 11:00:01 on 3-Apr-2014 (this is the system time).

   JavaScript Example:

   ```
   var temporal = require("/MarkLogic/temporal.xqy");

   var root =
       { "tempdoc": {
         "systemStart": null,
         "systemEnd": null,
         "validStart": "2014-04-03T11:00:00",
         "validEnd": "9999-12-31T11:59:59Z",
         "trader": "John",
         "price": 12
         }
       };

   declareUpdate();
   temporal.documentInsert("kool", "koolorder.json", root);
   ```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-04-03T11:00:00</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <trader>John</trader>
    <content>12</content>
</tempdoc>

return
temporal:document-insert("kool", "koolorder.xml", $root)
```

2. At 11:30:00, John changes his order to buy the stock at $13. The change is recorded as another version in the broker's database at 11:30:01.

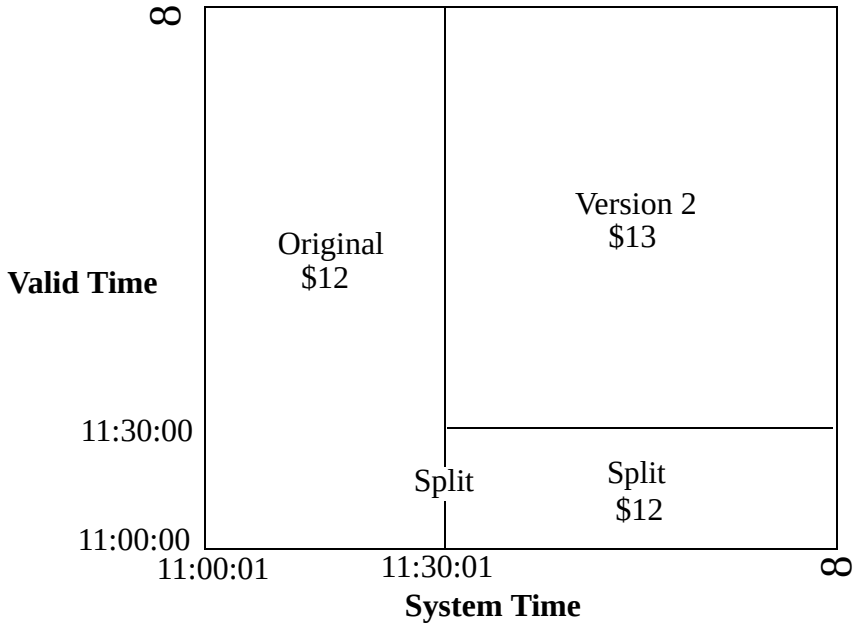
JavaScript Example:

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
      "systemStart": null,
      "systemEnd": null,
      "validStart": "2014-04-03T11:30:00",
      "validEnd": "9999-12-31T11:59:59Z",
      "trader": "John",
      "price": "13"
      }
    };

declareUpdate();
temporal.documentInsert("kool", "koolorder.json", root);
```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-04-03T11:30:00</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <trader>John</trader>
    <content>13</content>
</tempdoc>

return
temporal:document-insert("kool", "koolorder.xml", $root)
```

The result should be three documents with valid and system times as shown in the graphic below. Note that the second query resulted in a split on the Original document that resulted in a “split” document, as well as Version 2 that contains the new content.



2.5 Run some Search Queries on the Temporal Documents

The following query searches the temporal documents, using the `cts:period-range-query` function to locate the documents that were in the database between 11:10 and 11:15. `ISO_CONTAINS` is one of the comparison operators described in "ISO SQL 2011 Operators" on page 37.

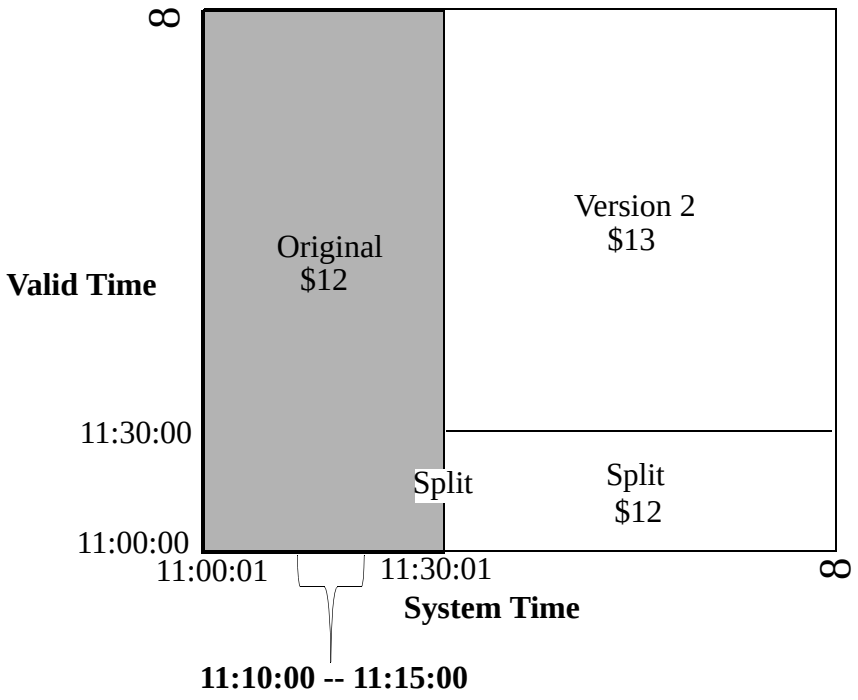
In this example, only Original Document.meets the search criteria.

JavaScript Example:

```
cts.search(cts.periodRangeQuery(
    "system",
    "ISO_CONTAINS",
    cts.period(xs.dateTime("2014-04-03T11:10:00"),
               xs.dateTime("2014-04-03T11:15:00")) ));
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:period-range-query(
    "system",
    "ISO_CONTAINS",
    cts:period(xs:dateTime("2014-04-03T11:10:00"),
               xs:dateTime("2014-04-03T11:15:00")) ))
```



The following query searches the temporal documents, using the `cts:period-range-query` function to locate the documents that have a valid time period that starts after 10:30 and ends at 11:30. `ALN_FINISHES` is one of the comparison operators described in “Allen Operators” on page 34.

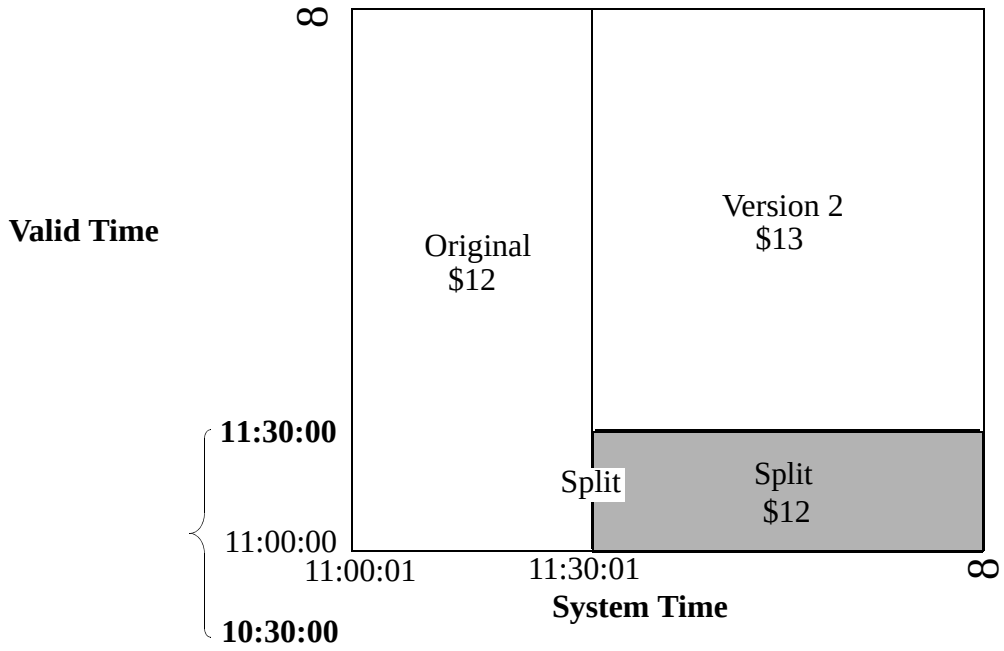
In this example, only the Split document meets the search criteria.

JavaScript Example:

```
cts.search(cts.periodRangeQuery(
    "valid",
    "ALN_FINISHES",
    cts.period(xs.dateTime("2014-04-03T10:30:00"),
               xs.dateTime("2014-04-03T11:30:00")) ));
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:period-range-query(
   "valid",
   "ALN_FINISHES",
   cts:period(xs:dateTime("2014-04-03T10:30:00"),
              xs:dateTime("2014-04-03T11:30:00")) ))
```



The following query searches the temporal documents, using the `cts:period-range-query` function to locate the documents that were in the database after 11:20. `ALN_AFTER` is one of the comparison operators described in “Allen Operators” on page 34.

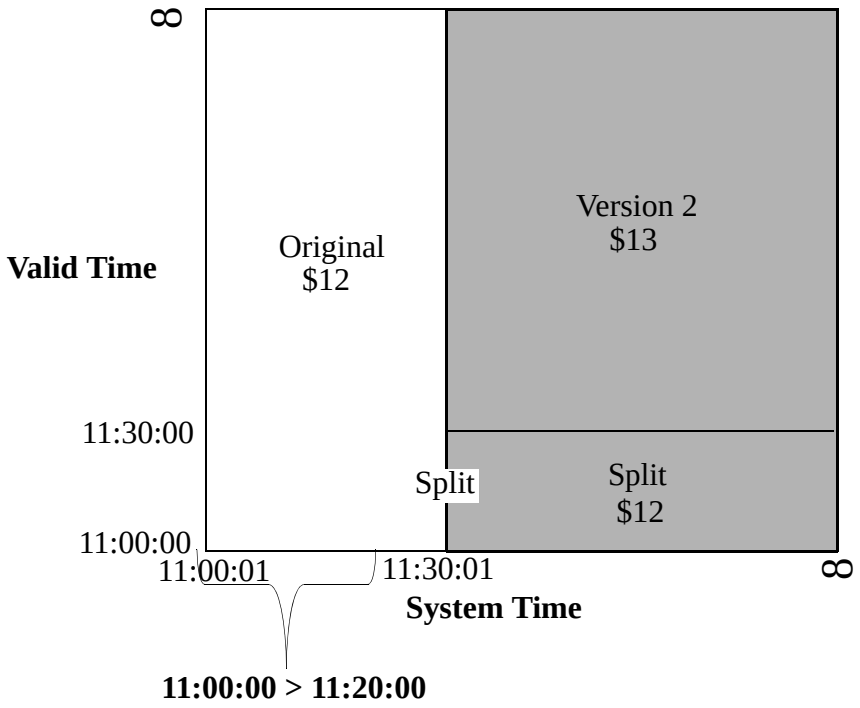
In this example, both the Split and Version 2 documents meet the search criteria.

JavaScript Example:

```
cts.search(cts.periodRangeQuery(
    "system",
    "ALN_AFTER",
    cts.period(xs.dateTime("2014-04-03T11:00:00"),
               xs.dateTime("2014-04-03T11:20:00")) ));
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:period-range-query(
   "system",
   "ALN_AFTER",
   cts:period(xs:dateTime("2014-04-03T11:00:00"),
              xs:dateTime("2014-04-03T11:20:00")) ))
```



The following query searches the temporal documents, using the `cts:period-compare-query` function to locate the documents that were in the database when the valid time period is within the system time period. `ISO_CONTAINS` is one of the comparison operators described in "ISO SQL 2011 Operators" on page 37.

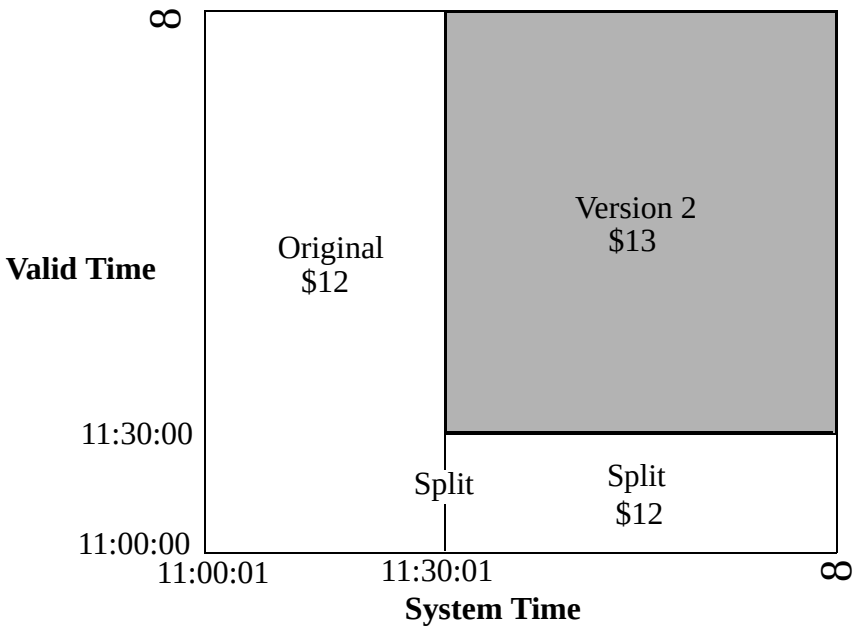
In this example, only Version 2 of the document meets the search criteria.

JavaScript Example:

```
cts.search(cts.periodCompareQuery(
    "system",
    "ISO_CONTAINS",
     "valid" ))
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:period-compare-query(
   "system",
   "ISO_CONTAINS",
   "valid" ))
```



The following query uses the `cts:and-query` to AND two `cts:collection-query` functions to return the temporal document that is in the URI collection, `koolorder.xml`, and the `latest` collection.

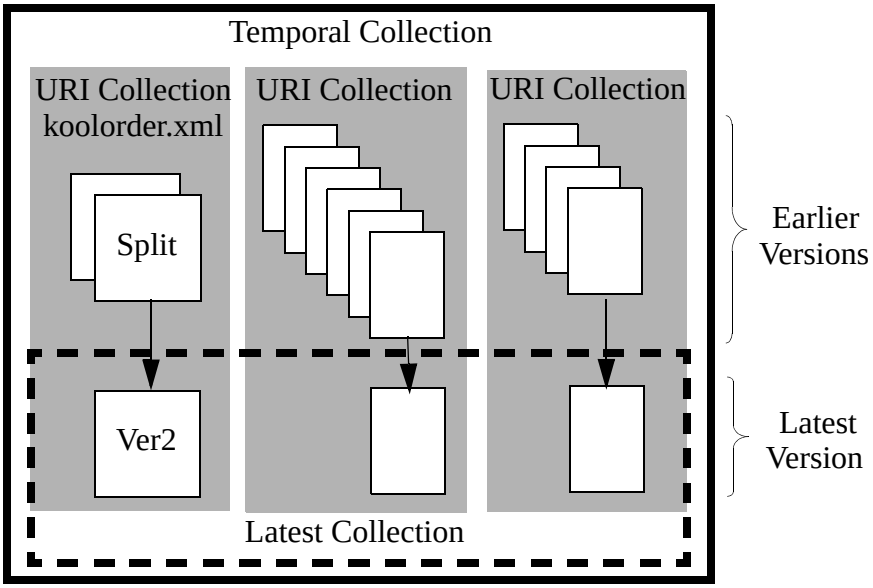
In this example, Ver 2 meets the search criteria.

JavaScript Example:

```
cts.search(cts.andQuery([
    cts.collectionQuery("koolorder.json"),
    cts.collectionQuery("latest")]))
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:and-query((
    cts:collection-query(("koolorder.xml")),
    cts:collection-query(("latest")))))
```



3.0 Managing Temporal Documents

This chapter describes how to insert and update temporal documents, and includes the following sections:

3.1 Inserting and Loading Temporal Documents

There are a number of ways to insert and update temporal documents in MarkLogic Server. These include:

- The XQuery functions, `temporal:document-insert` and `temporal:document-load`.
- The REST Client API resource addresses, `POST:/v1/documents` and `PUT:/v1/documents` with the `temporal-collection` parameter.
- The Node.js Client API, by including a temporal collection URI in operations such as `documents.write`. For details, see the *Node.js Client API Reference*.
- The MarkLogic Content Pump (mlcp), as described in “Using MarkLogic Content Pump (MLCP) to Load Temporal Documents” on page 22.
- XCC `ContentCreateOptions` class functions: `getTemporalCollection` and `setTemporalCollection` to get and set the temporal collection object associated with the inserted documents.
- Hadoop Connector `mapreduce.marklogic.output.temporalcollection` property to specify the temporal collection that is used to ingest the documents.

Though MarkLogic manages temporal documents in the same manner regardless of the tool you use, this section describes the use of the XQuery functions, `temporal:document-insert` and `temporal:document-load`, to insert and update temporal documents into MarkLogic Server.

Calling either `temporal:document-insert` or `temporal:document-load` on an existing URI “updates” the existing temporal document. An update on a temporal document results in a new document, rather than an overwrite of the original document.

You can use `xdmp:document-set-properties` to set properties on temporal documents, but you cannot use the `xdmp:document-set-quality`, `xdmp:document-set-permissions`, or `xdmp:document-set-collections` functions to set their respective attributes on temporal documents.

The document being inserted or updated must include a valid start and end times. These times must be `dateTime` values identified by elements that map to the range indexes that represent the valid start and end time period. On insert, MarkLogic sets the system start time to the current time system time and the end time to the farthest possible time (infinity).

JavaScript Example:

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-06-03T14:13:05",
          "validEnd": "9999-12-31T11:59:59Z",
          "content": "v1-content here"
          }
    };

declareUpdate();
temporal.documentInsert("temporalCollection", "doc.json", root);
```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-06-03T14:13:05</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <content>v1-content here</content>
</tempdoc>

return
    temporal:document-insert("temporalCollection", "doc.xml", $root)
```

If you don't specify a system time, MarkLogic Server will fill in the system start and end times. If you have enabled LSQT, as described in "Last Stable Query Time (LSQT) and Application-controlled System Time" on page 20, you can alternately have your application call the `temporal:statement-set-system-time` function to specify a system time along with your insert. The `dateTime` given must be later than the LSQT (Last Stable Query Time) returned by the `temporal:get-lsqt` function.

JavaScript Example:

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-06-03T14:13:05",
          "validEnd": "9999-12-31T11:59:59Z",
          "content": "v1-content here"
          }
    };

declareUpdate();
temporal.documentInsert("temporalCollection", "doc.json", root),
temporal.statementSetSystemTime(xs.dateTime("2014-06-03T14:15:00"));
```

XQuery Example:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-06-03T14:13:05</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <content>v1-content here</content>
</tempdoc>

return
    temporal:document-insert("temporalCollection","doc.xml",$root),
    temporal:statement-set-system-time(
        xs:dateTime("2014-06-03T14:15:00"))
```

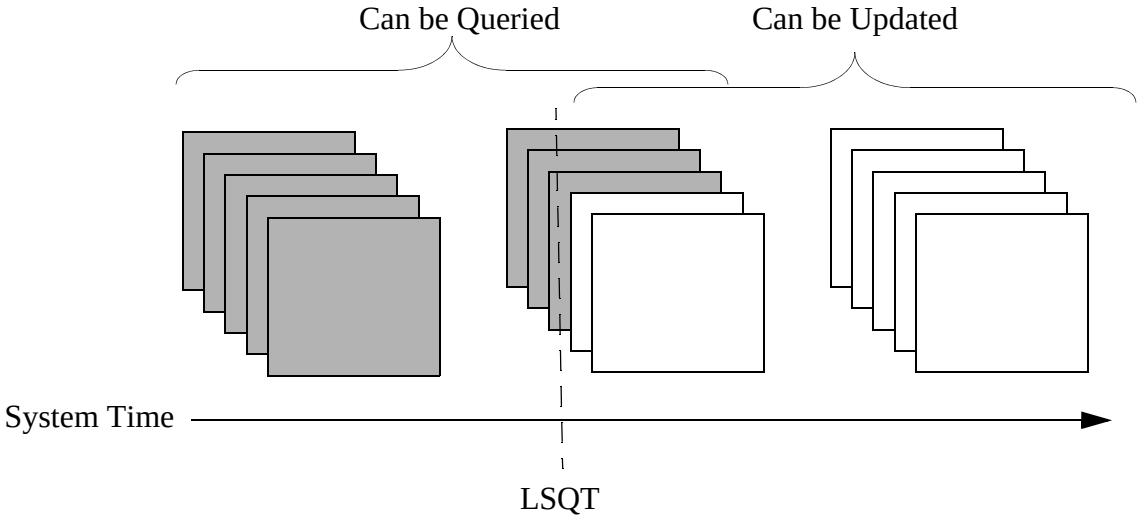
Note: Document properties are not updated on temporal documents, so do not use Content Processing Framework (CPF) or Library Services (DLS) on temporal data.

3.2 Last Stable Query Time (LSQT) and Application-controlled System Time

You can manually set the system start time when inserting or updating a document in a collection. This feature is useful when you need to maintain a "master" system time across multiple clients that are concurrently inserting and updating bitemporal documents, without the need for the clients to communicate with one another in order to coordinate their system times.

The system start times for document versions with the same URI must progress along the system time axis, so that an update to a document cannot have a system start time that is earlier than that of the document that chronicles its last update. However, when managing documents with different URIs in a temporal collection, it is necessary to ensure that the system time progresses at the same rate for every document insert and update.

A special timestamp, called the LSQT (Last Stable Query Time), can be enabled on a temporal collection to manage system start times across documents with different URIs. A temporal document with a system start time before the LSQT can only be queried and a document with a system start time after the LSQT can be updated / ingested, but not queried. You can advance the LSQT, either manually or automatically, to manage which documents are available to be queried and which documents can be updated.



You can use the `temporal:set-use-lsqt` function to enable or disable LSQT on a temporal collection. When LSQT is enabled, the LSQT is stored in a document in the database, with a name of the form *`collection-name`*`.lsqt`. You can call the `temporal:advance-lsqt` function to manually advance the LSQT or use the `temporal:set-lsqt-automation` function to direct MarkLogic to automatically advance the LSQT at set periods.

When LSQT is enabled on a temporal collection, the LSQT value starts at 0 (lowest timestamp) When advanced, document reads and writes are quiesced until the LSQT is reset to the maximum system start time in the database. You must have LSQT enabled in order to use the `temporal:statement-set-system-time` function to set the system start time.

> **Note:** You can only call the `temporal:statement-set-system-time` function once per statement

For example, the following query first checks to make sure the application time (simulated by the current time) is greater than the LSQT:

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
    at "/MarkLogic/temporal.xqy";

let $appTime := fn:current-dateTime()
let $LSQT := temporal:get-lsqt("temporalcollection")

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-06-03T14:13:05</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <content>v1-content here</content>
</tempdoc>

let $systemTime :=
      if ($appTime > $LSQT)
      then (temporal:statement-set-system-time(xs:dateTime($appTime)))
      else ()

return (
   temporal:document-insert(
     "temporalcollection",
     "doc.xml",
     $root),
     $systemTime )
```

3.3 Using MarkLogic Content Pump (MLCP) to Load Temporal Documents

You can use the MarkLogic Content Pump (MLCP) with the `-temporal_collection` option to load temporal documents into a specific temporal collection in the MarkLogic Server database. The two MLCP commands that are supported for loading temporal documents are `import` and `copy`.

For example, to import the temporal documents in the `/etc/orders` directory on the filesystem into the temporal collection, named "kool," into the temporal database on the host, `desthost`, you would use the following MLCP command:

```
mlcp.sh import -temporal_collection kool -input_file_path /etc/orders \
-host desthost -port 8006 -username user1
```

Note: If you omit `-port`, MLCP will use port 8000.

You can use the `copy` command to copy non-temporal documents from a database and ingest them as temporal documents into the temporal collection in another database. For example to migrate the temporal documents from the database used by the host, `srchost`, to the temporal collection named "kool" in the database used by the host, `desthost`, you would use a command like the following:

```
mlcp.sh copy -mode local -input_host srchost -input_port 8006 \
-input_username user1 -input_password password1 \
-output_host desthost -output_port 8010 -temporal_collection kool \
-output_username user2 -output_password password2
```

Note: You cannot import or copy binary files as temporal documents or specify an `-input_file_type` of `rdf` or `forest` for temporal documents.

For details on using MLCP to load documents, see Loading Content Using MarkLogic Content Pump in the *Loading Content Into MarkLogic Server Guide*.

3.4 Deleting Temporal Documents

You can use the `temporal:document-delete` function to delete temporal documents. Deleting a temporal document maintains the document and all of its versions in the URI collection and updates the deleted document and all of its versions that have a system end time of infinity to the time of the delete.

Deleting a temporal document removes the document from the `latest` collection. So the `latest` collection is the source of all of the documents that are currrently valid and the URI collections are the source of the history of each document.

Should you insert a document using the same URI as a deleted document, the deleted document, and all of its previous versions remain in the same URI collection as the "newly" inserted document. The newly inserted document is then added to the `latest` collection.

3.5 Example: The Lifecycle of a Temporal Document

The example in this section builds on the example described in "Quick Start" on page 6. The purpose of this example is to show how new versions of a temporal document are generated and updated from a series of changes to a stock purchase order.

1. The stock of KoolCo is trading around $12.65. John places a limit order to buy 100 shares for $12 at 11:00:00 on 3-Apr-2014 (this is the valid start time). A temporal document for the order is recorded in the broker's database at 11:00:01 on 3-Apr-2014 (this is the system start time).

2. At 11:30:00, John changes his order to buy the stock at $13. The change is recorded as another version in the broker's database at 11:30:01.

3. At 12:10:00, John changes his mind again and decides to change his order to a limit order to buy at $12.50. This transaction is recorded as another version with a valid time of 12:10:00, but due to heavy trading, the change is not recorded in the broker's database until 12:10:12.

4. At 13:00:00, the purchase order has not been filled and John decides he no longer wants to buy the stock, so he cancels his order. This cancellation is recorded as another version with a valid time of 13:00:00 and recorded in the broker's database at 13:00:02.

5. However, at 13:00:01, the stock hits $12.50 and John's order is filled.

6. The broker's policy is to honor the valid times for all orders. At 13:00:03, the order fulfillment application reviews the valid and system times recorded at the time of the cancellation, determines that John in fact cancelled his order before it was filled, and does not debit his account for the stock purchase.

The valid and system times are each `dateTime` ranges that define a start and end time. The start time represents the time at which the information is known (as both valid and system times) and the end time represents the time at which the information is no longer true.

The above stock purchase example was kept simple for clarity. The following shows the insert query and the resulting documents with the actual valid and system times for the example, along with their respective start and end times. The graphic at the end displays the relationships between the documents in terms of valid and system times.

11:00:01 -- Initial order:

Insert Query (JavaScript):

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-04-03T11:00:00",
          "validEnd": "9999-12-31T11:59:59Z",
          "order 1": "12"
          }
    };

declareUpdate();
temporal.documentInsert("kool", "koolorder.json", root);
```

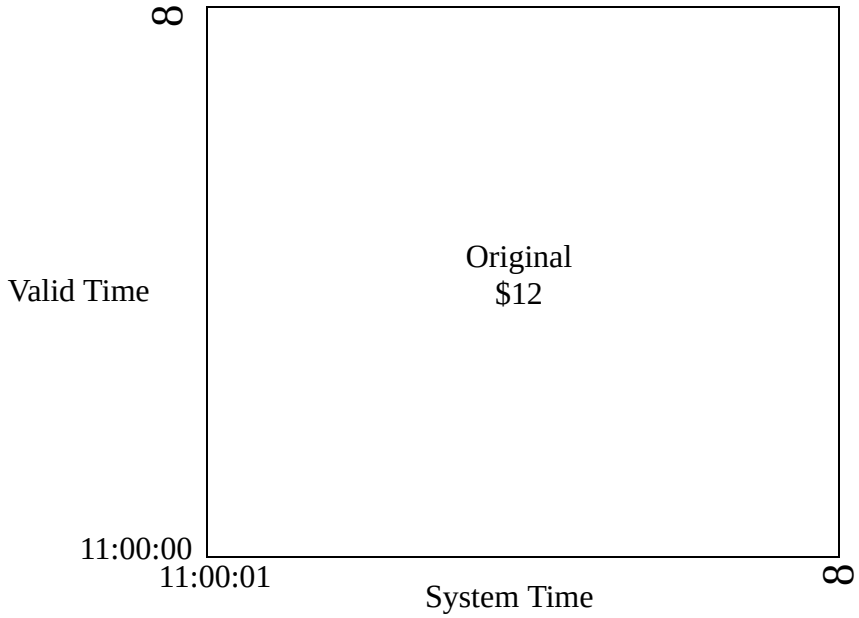
Insert Query (XQuery):

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
      at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-04-03T11:00:00</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <order1>12</order1>
</tempdoc>

return temporal:document-insert("kool", "koolorder.xml", root)
```

Results:

Original Document:

```
Order Price: $12

System Start: 2014-04-03T11:00:01
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T11:00:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```



Note: The date-time `9999-12-31T11:59:59Z` represents infinity and is used when there is no valid date-time yet for that start or end time.

11:30:00 -- Changed order from $12 to $13.

Update Query (JavaScript):

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-04-03T11:30:00",
          "validEnd": "9999-12-31T11:59:59Z",
          "order 2": "13"
          }
    };

declareUpdate();
temporal.documentInsert("kool", "koolorder.json", root);
```

Update Query (XQuery):

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
      at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
    <systemStart/>
    <systemEnd/>
    <validStart>2014-04-03T11:30:00</validStart>
    <validEnd>9999-12-31T11:59:59Z</validEnd>
    <order2>13</order2>
</tempdoc>

return temporal:document-insert("kool", "koolorder.xml", $root)
```

Results:

Original Document (updated):

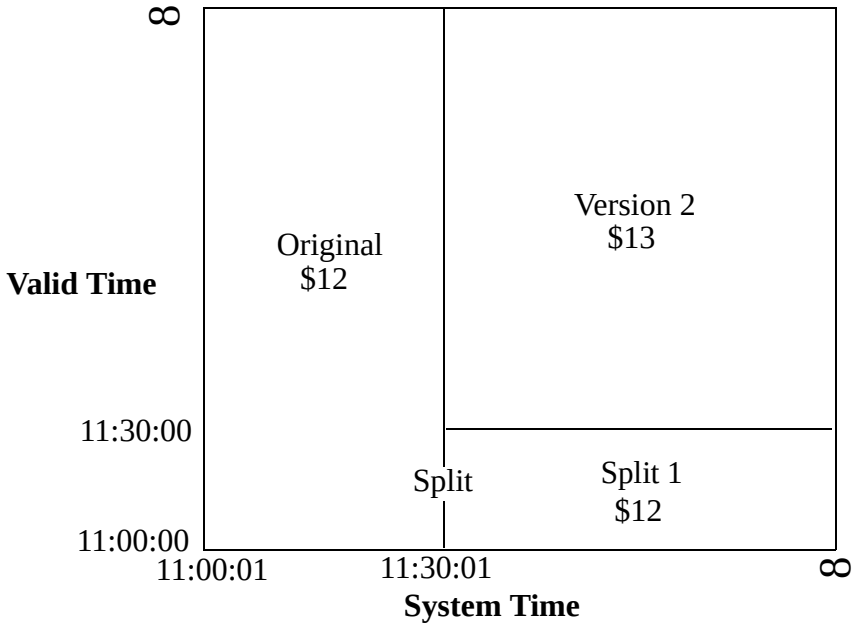
```
Order Price: $12

System Start: 2014-04-03T11:00:01
System End: 2014-04-03T11:30:01 <-- changed

Valid Start: 2014-04-03T11:00:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```

Split 1 from Original Document:

```
Order Price: $12

System Start: 2014-04-03T11:30:01
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T11:00:00
Valid End: 2014-04-03T11:30:00
```

Version 2:

```
Order Price: $13

System Start: 2014-04-03T11:30:01
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T11:30:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```



12:10:00 -- Changed order from $13 to $12:50:

Update Query (JavaScript):

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-04-03T12:10:00",
          "validEnd": "9999-12-31T11:59:59Z",
          "order 3": "12.50"
          }
    };

declareUpdate();
temporal.documentInsert("kool", "koolorder.json", root);
```

Update Query (XQuery):

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
      at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
   <systemStart/>
   <systemEnd/>
   <validStart>2014-04-03T12:10:00</validStart>
   <validEnd>9999-12-31T11:59:59Z</validEnd>
   <order3>12.50</order3>
</tempdoc>

return temporal:document-insert("kool", "koolorder.xml", $root)
```

Results:

Original Document (no change)

Split 1 (no change)

Version 2 (update):

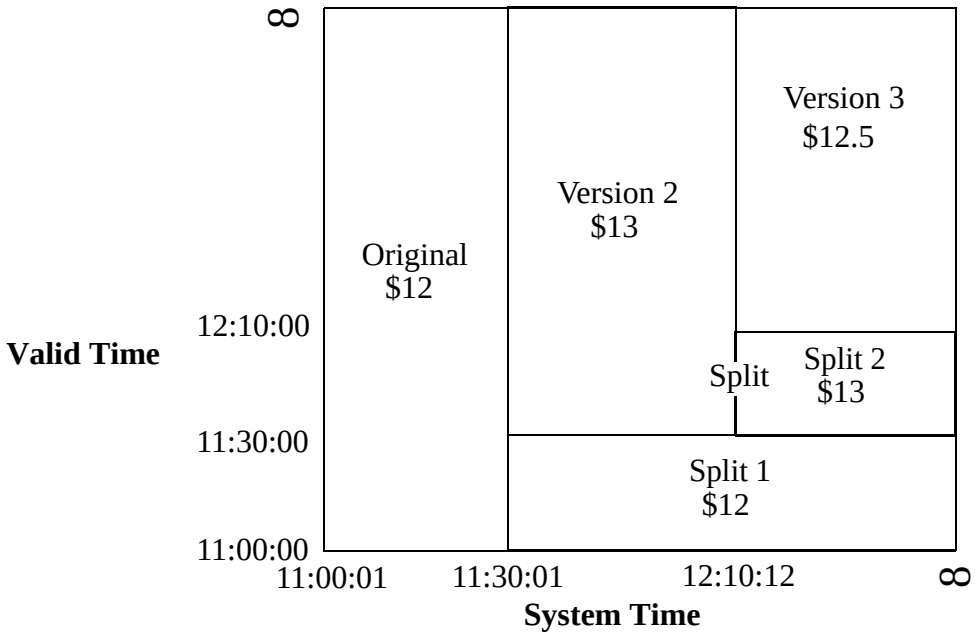
```
Order Price: Closing price

System Start: 2014-04-03T11:30:01
System End: 2014-04-03T12:10:12 <-- changed

Valid Start: 2014-04-03T11:30:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```

Split 2 (new)

```
Order Price: $12

System Start: 2014-04-03T12:10:12
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T11:00:00
Valid End: 2014-04-03T12:10:00
```

Version 3 (new):

```
Order Price: $12.5

System Start: 2014-04-03T12:10:12
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T12:10:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```



13:00:00 -- Cancelled Order. 6 Documents:

Update Query (JavaScript):

```
var temporal = require("/MarkLogic/temporal.xqy");

var root =
    { "tempdoc": {
          "systemStart": null,
          "systemEnd": null,
          "validStart": "2014-04-03T13:00:00",
          "validEnd": "9999-12-31T11:59:59Z",
          "cancel": "0"
          }
    };

declareUpdate();
temporal.documentInsert("kool", "koolorder.json", root);
```

Update Query (XQuery):

```
xquery version "1.0-ml";

import module namespace temporal = "http://marklogic.com/xdmp/temporal"
      at "/MarkLogic/temporal.xqy";

let $root :=
<tempdoc>
   <systemStart/>
   <systemEnd/>
   <validStart>2014-04-03T13:00:00</validStart>
   <validEnd>9999-12-31T11:59:59Z</validEnd>
   <cancel>0</cancel>
</tempdoc>

return temporal:document-insert("kool", "koolorder.xml", $root)
```

Results:

Original Document (no change)

Split 1 (no change)

Version 2 (no change)

Split 2 (no change)

Version 3 (updated)

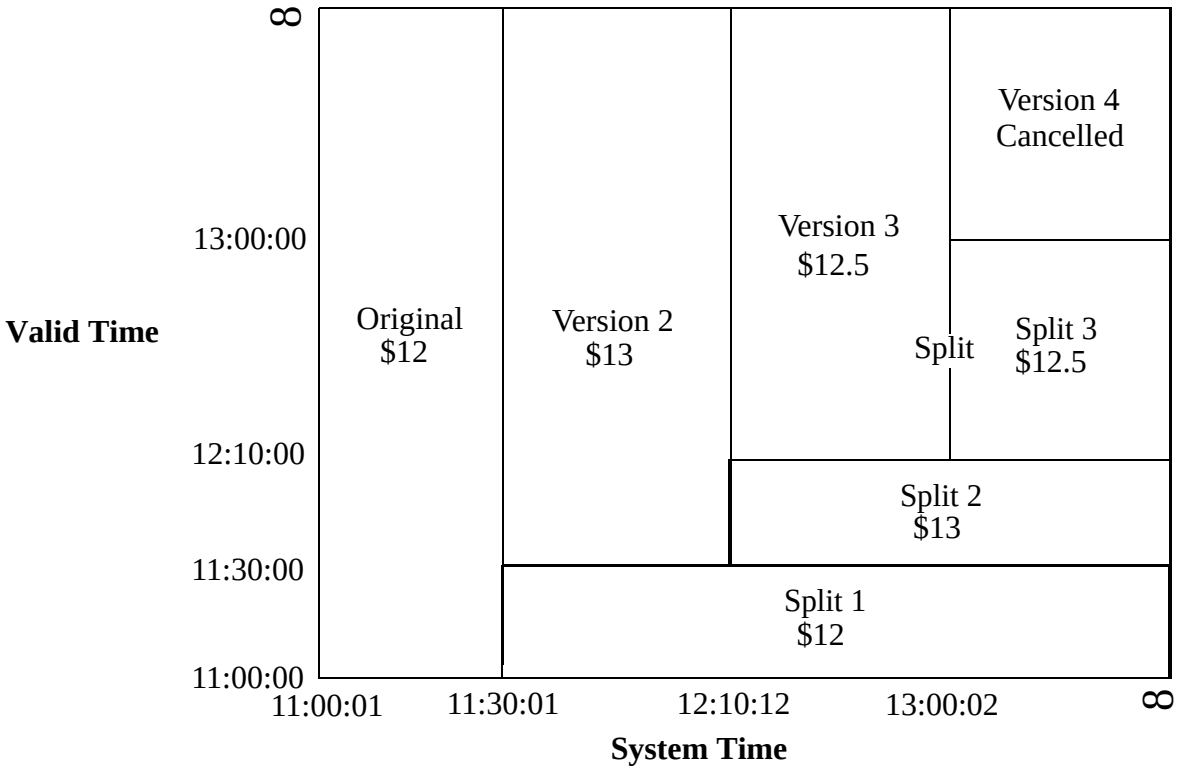
```
Order Price: $12.5

System Start: 2014-04-03T12:10:12
System End: 2014-04-03T13:00:02 <-- changed

Valid Start: 2014-04-03T12:10:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```

Split 3 (new)

```
Order Price: $12.5

System Start: 2014-04-03T13:00:02
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T11:00:00
Valid End: 2014-04-03T13:00:00
```

Version 4 (new):

```
Order Price: $0

System Start: 2014-04-03T13:00:02
System End: 9999-12-31T11:59:59Z <-- infinity

Valid Start: 2014-04-03T13:00:00
Valid End: 9999-12-31T11:59:59Z <-- infinity
```


∞
13:00:00
12:10:00
11:30:00
11:00:00
Valid Time
Original
$12
Version 2
$13
Version 3
$12.5
Version 4
Cancelled
Split
Split 3
$12.5
Split 2
$13
Split 1
$12
11:00:01
11:30:01
12:10:12
13:00:02
∞
System Time

4.0 Searching Temporal Documents

This chapter describes the temporal search features, and includes the following sections:

4.1 Temporal Search Query Constructors

The following table summarizes the available functions used to construct `cts:query` expressions for searching temporal documents. For general details on constructing `cts:query` expressions, see Composing cts:query Expressions in the *Search Developer's Guide*.

Function (XQuery and JavaScript)	Description
`cts:period-compare-query` `cts.periodCompareQuery`	Returns a `cts:query` matching documents that have relevant pair of period values.
`cts:period-compare-query-operator` `cts.periodRangeQueryOperator`	Returns the operator used to construct the specified query.
`cts:period-compare-query-options` `cts.periodCompareQueryOptions`	Returns the options for the specified query.
`cts:period-compare-query-axis-1` `cts.periodCompareQueryAxis1`	Returns the name of the first axis used to construct the specified query.
`cts:period-compare-query-axis-2` `cts.periodCompareQueryAxis2`	Returns the name of the second axis used to construct the specified query.
`cts:period-range-query` `cts.periodRangeQuery`	Returns a `cts:query` matching period by name with a period value with an operator.
`cts:period-range-query-operator` `cts.periodRangeQueryOperator`	Returns the operator used to construct the specified query.
`cts:period-range-query-options` `cts.periodRangeQueryOptions`	Returns the options for the specified query.

Function (XQuery and JavaScript)	Description
`cts:lsqt-query` `cts.lsqtQuery`	Returns only documents before LSQT or a timestamp before LSQT for stable query results.
`cts:lsqt-query-options` `cts.lsqtQueryOptions`	Returns the options for the specified query.
`cts:lsqt-query-temporal-collection` `cts.lsqtQueryTemporalCollection`	Returns the name of the temporal collection used to construct specified query.
`cts:lsqt-query-timestamp` `cts.lsqtQueryTimestamp`	Returns timestamp used to construct the specified query.
`cts:lsqt-query-weight` `cts.lsqtQueryWeight`	Returns the weight with which the specified query was constructed.

4.2 Period Comparison Operators

This section describes the Allen and ISO SQL algebra operators that can be used in search queries. Temporal queries are basically some interval operations on time period such as, period equalities, containment and overlaps. MarkLogic Server supports both Allen and SQL operators when comparing time periods. Allen's interval algebra provides the most comprehensive set of these operations. SQL 2011 also provides similar operators. However all the SQL operators can be expressed using Allen's Algebra.

4.2.1 Allen Operators

In general, Allen operators, which are identified with an ALN_ prefix, are more restrictive than ISO SQL operators, which are identified with an ISO_ prefix. The illustration below shows the relationships between the X and Y periods for each operator as used in the following period queries:

```
cts:period-range-query(X, operator, Y)
```

where X is an axis and Y is a period.

```
cts:period-compare-query(X, operator, Y)
```

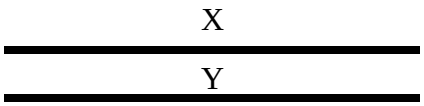
where X and Y are both axes.

```
cts:period-compare(X, operator, Y)
```

where X and Y are both periods.

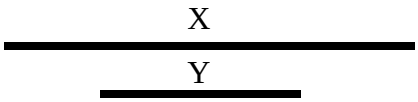
ALN_EQUALS

X-start = Y-start and X-end = Y-end



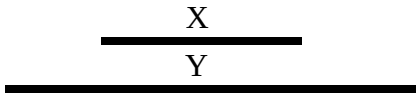
ALN_CONTAINS

X-start < Y-start and X-end > Y-end



ALN_CONTAINED_BY

X-start > Y-start and X-end < Y-end

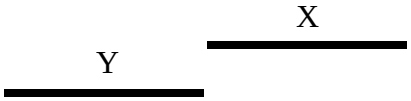


ALN_MEETS

X-end = Y-start

ALN_MET_BY

X-start = Y-end

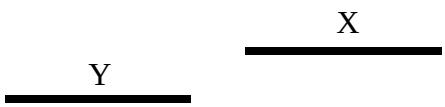


ALN_BEFORE

X-end < Y-start

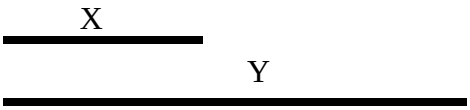
ALN_AFTER

X-start > Y-end



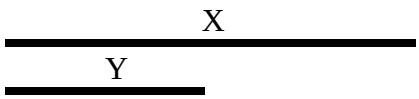
ALN_STARTS

X-start = Y-start and X-end > Y-end



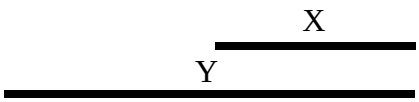
ALN_STARTED_BY

X-start = Y-start and X-end > Y-end



ALN_FINISHES

X-end = Y-end and X-start > Y-start



ALN_FINISHED_BY

X-end = Y-end and X-start < Y-start

ALN_OVERLAPS

X-start < Y-start and
X-end > Y-start and
X-end < Y-end

ALN_OVERLAPPED_BY

X-start > Y-start and
X-end > Y-end and
X-start < Y-end

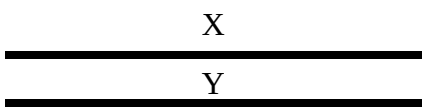
4.2.2 ISO SQL 2011 Operators

Similar to Allen operators, SQL 2011 operators can also be mapped MarkLogic range index operations. The following illustrations show the mapping from ISO SQL operators to a composition of Allen Operators.

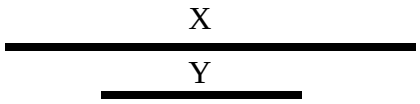
Note that the ALN operators described in “Allen Operators” on page 34 each have only one X/Y period relationship, whereas some of the less restrictive ISO SQL operators have multiple X/Y period relationships. Such ISO SQL operators have the effect of multiple Allen operators, as shown below.

ISO_OVERLAPS

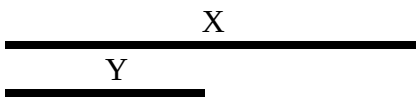
X ALN_EQUALS Y



X ALN_CONTAINS Y



X ALN_STARTED_BY Y

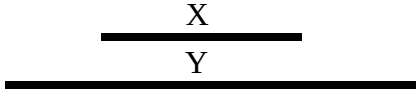


X ALN_FINISHED_BY Y

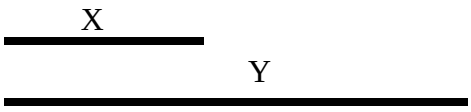
X ALN_OVERLAPPED_BY Y

X ALN_OVERLAPS Y

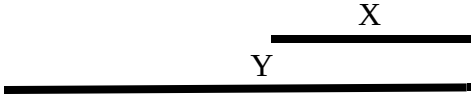
X ALN_CONTAINED_BY Y



X ALN_STARTS Y

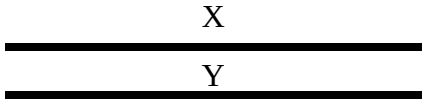


X ALN_FINISHES Y

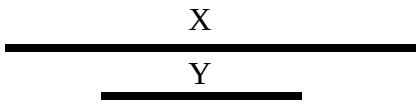


ISO_CONTAINS

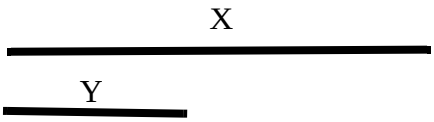
X ALN_EQUALS Y



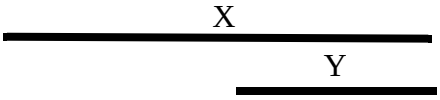
X ALN_CONTAINS Y



X ALN_STARTED_BY Y



X ALN_FINISHED_BY Y

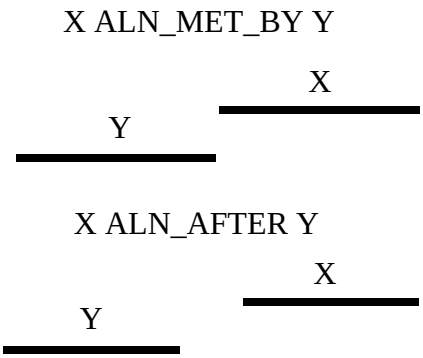


ISO_PRECEDES

X ALN_MEETS Y

X ALN_BEFORE Y

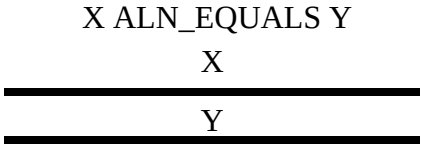
ISO_SUCCEEDS



ISO_IMM_PRECEDES

ISO_IMM_SUCCEEDS

ISO_EQUALS



4.3 Comparing Two Periods

The range query on periods can be used only when there are range indexes for them on the database. However you may need to query based either on some external period values or those from a document. The following example shows how you can use the `cts:period-compare` function to determine whether two period values satisfy the conditions imposed by the comparison operator, `ALN_MEETS`. If the `ALN_MEETS` conditions are satisfied, then `true` is returned; otherwise `false` is returned.

JavaScript Example:

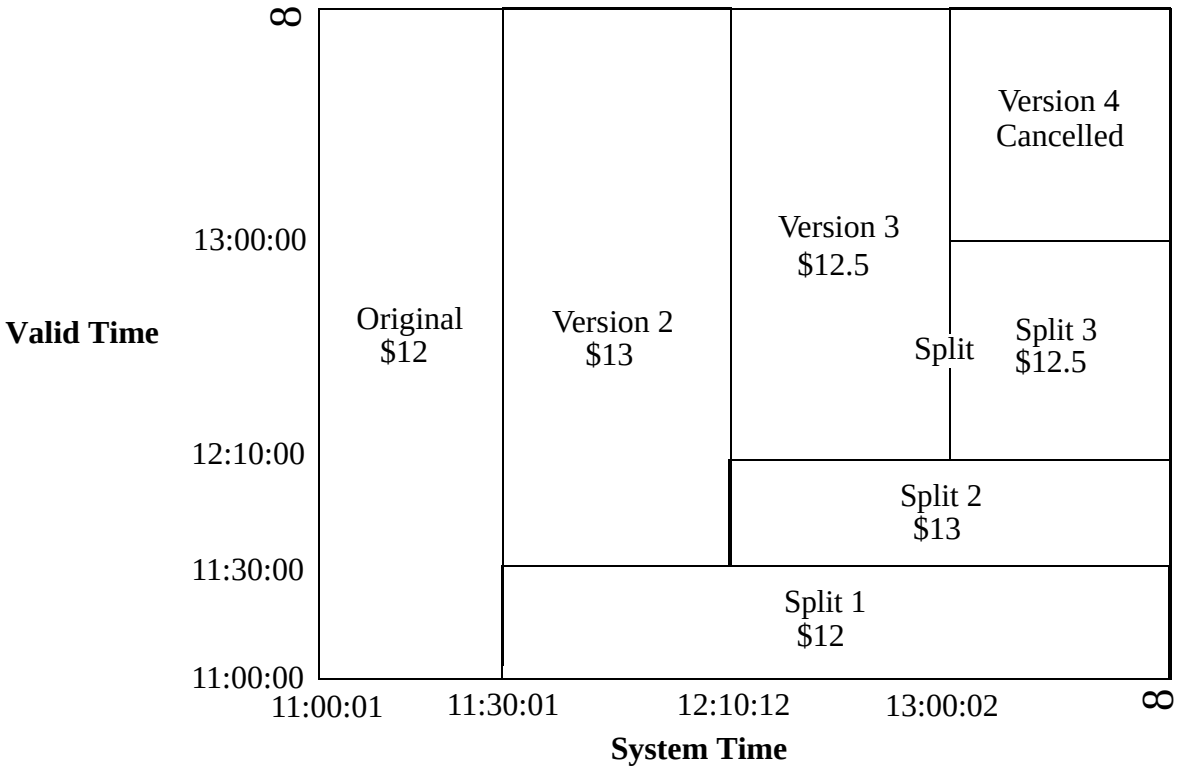
```
var period1 = cts.period(xs.dateTime("2000-05-31T09:30:10"),
                         xs.dateTime("2003-05-31T12:30:00"));

var period2 = cts.period(xs.dateTime("2003-05-31T12:30:00"),
                         xs.dateTime("2004-05-31T14:30:00"));

cts.periodCompare(period1, "ALN_MEETS", period2);
```

XQuery Example:

```
xquery version "1.0-ml";

let $period1 := cts:period(xs:dateTime("2000-05-31T09:30:10"),
                           xs:dateTime("2003-05-31T12:30:00"))

let $period2 := cts:period(xs:dateTime("2003-05-31T12:30:00"),
                           xs:dateTime("2004-05-31T14:30:00"))

return cts:period-compare($period1,"ALN_MEETS",$period2)
```

4.4 Example Search Queries

This section describes some sample search queries. The searches described in this section are done on the documents described in “Example: The Lifecycle of a Temporal Document” on page 23.



The following query searches for the temporal documents that have a valid end time before `14:00`:

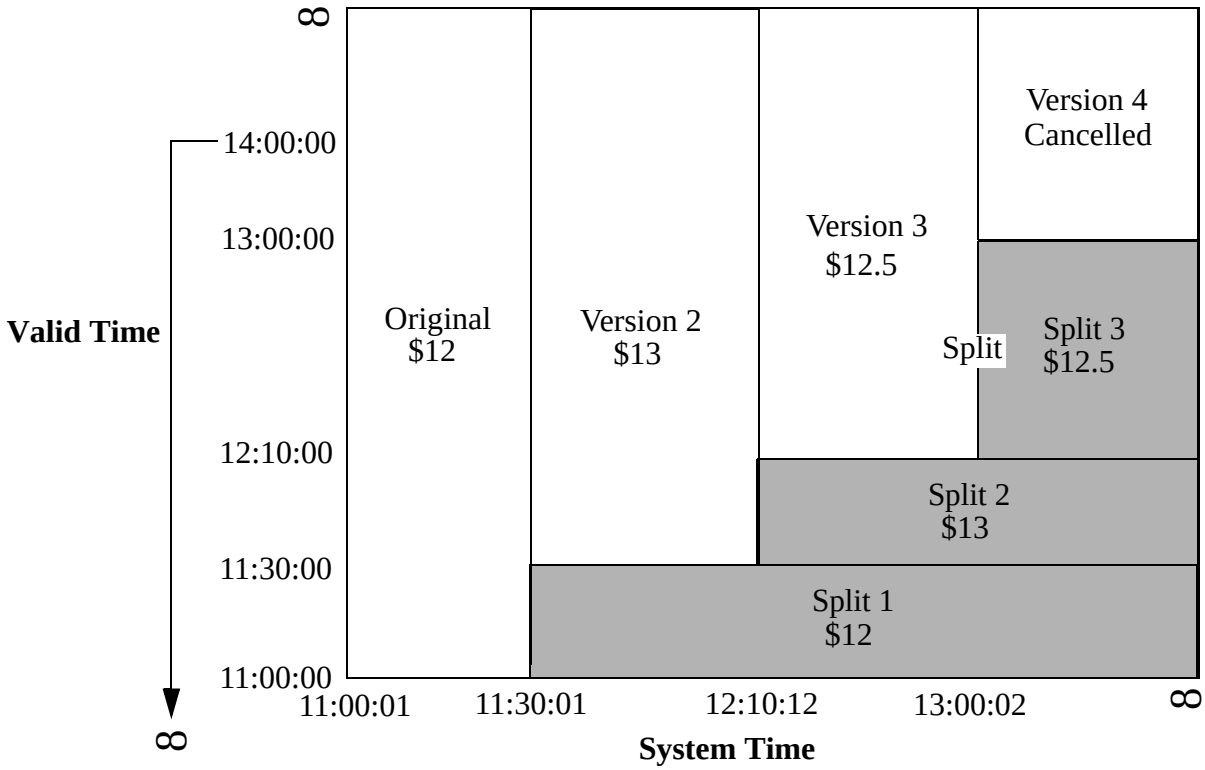
JavaScript Example:

```
cts.search(cts.periodRangeQuery(
    "valid",
    "ALN_BEFORE",
    cts.period(xs.dateTime("2014-04-03T14:00:00"),
               xs.dateTime("9999-12-31T11:59:59Z")) ))
```

XQuery Example:

```
cts:search(fn:doc(), cts:period-range-query(
   "valid",
   "ALN_BEFORE",
   cts:period(xs:dateTime("2014-04-03T14:00:00"),
              xs:dateTime("9999-12-31T11:59:59Z")) ))
```

This query returns Splits 1, 2 and 3 of the document.



The following query searches the temporal documents, using the `cts:and-query` to AND two `cts:period-range-query` functions, to locate the documents that represented the order at 11:30 when queried at 11:51.

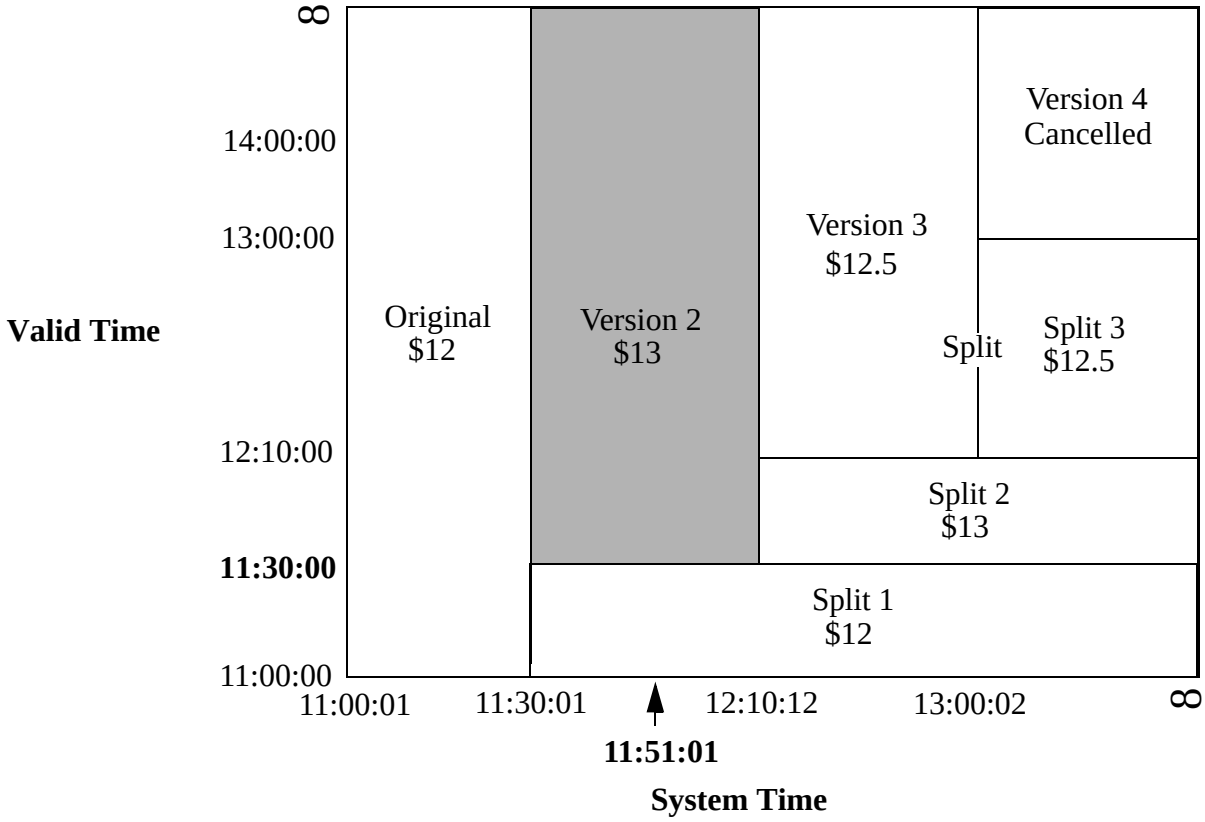
JavaScript Example:

```
cts.search(cts.andQuery(
      [cts.periodRangeQuery(
          "system",
          "ISO_CONTAINS",
          cts.period(xs.dateTime("2014-04-03T11:51:00"),
                     xs.dateTime("2014-04-03T11:51:01"))),
      cts.periodRangeQuery(
          "valid",
          "ISO_CONTAINS",
          cts.period(xs.dateTime("2014-04-03T11:30:00"),
                     xs.dateTime("2014-04-03T11:30:01")))]))
```

XQuery Example:

```
xquery version "1.0-ml";

cts:search(fn:doc(), cts:and-query((
   cts:period-range-query(
      "system",
      "ISO_CONTAINS",
      cts:period(xs:dateTime("2014-04-03T11:51:00"),
                 xs:dateTime("2014-04-03T11:51:01"))),
   cts:period-range-query(
      "valid",
      "ISO_CONTAINS",
      cts:period(xs:dateTime("2014-04-03T11:30:00"),
                 xs:dateTime("2014-04-03T11:30:01"))))))
```

This query returns Version 2 of the document.



The following query searches for the temporal documents that have a valid end time of `12:10`:

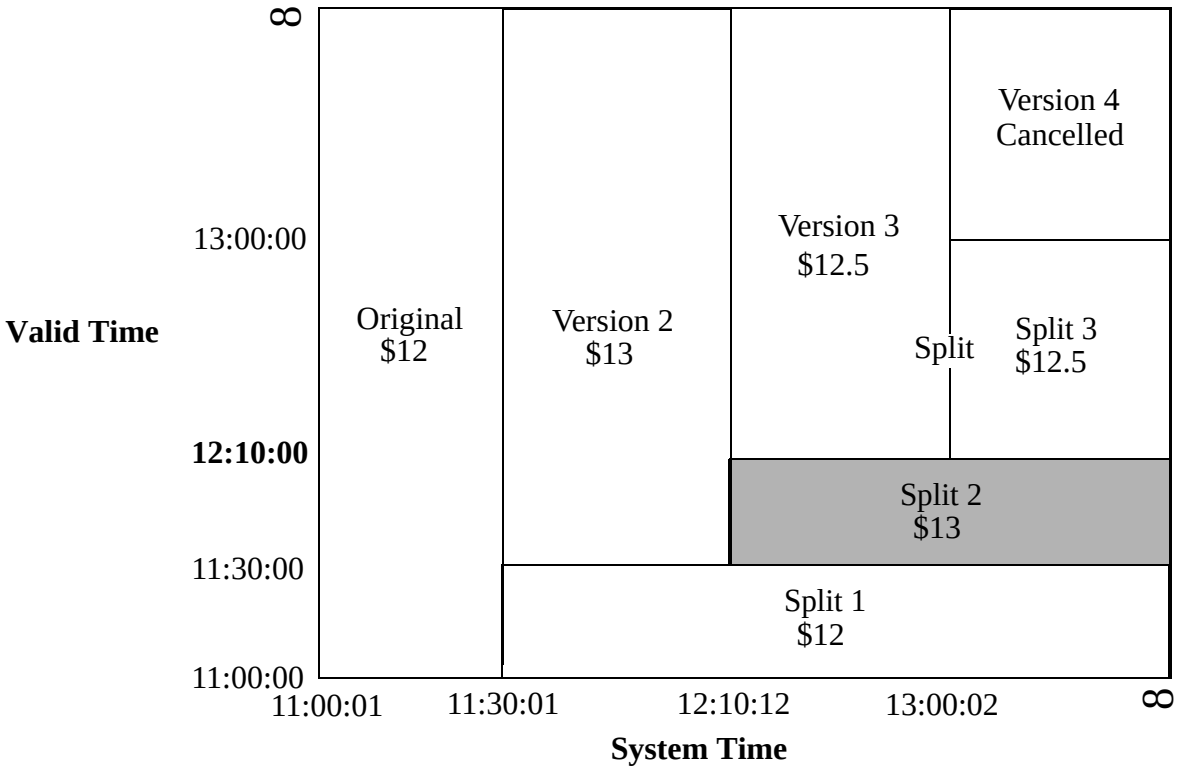
JavaScript Example:

```
var period = cts.period(xs.dateTime("2014-04-03T12:10:00"),
                        xs.dateTime("2014-04-03T13:10:00"));

cts.search(cts.periodRangeQuery("valid", "ALN_MEETS", period))
```

XQuery Example:

```
xquery version "1.0-ml";

let $period := cts:period(xs:dateTime("2014-04-03T12:10:00"),
                          xs:dateTime("2014-04-03T13:10:00"))

return cts:search(fn:doc(), cts:period-range-query(
     "valid", "ALN_MEETS", $period))
```

This query returns Split 2 of the document.



5.0 Copyright

MarkLogic Server 8.0 and supporting products.

NOTICE

6.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at http://help.marklogic.com to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the Support Handbook for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at http://developer.marklogic.com. For general questions, join the general discussion mailing list, open to all MarkLogic developers.