

## Xenomorph White Paper

# TimeScape and MATLAB

---

This paper describes how Xenomorph's TimeScape analytics and data management solution can be accessed and utilised within MathWorks' MATLAB computational and visualization environment. It is intended for MATLAB users who wish to combine data and analytics hosted in TimeScape with MATLAB tools in order to produce best-of-breed analytical solutions based on cleansed and validated data.

Version 1.1, November 2011



## Contents

---

<b>1 Introduction</b> .....	3
<b>2 Objectives</b> .....	3
<b>3 TimeScape and MATLAB Integration</b> .....	4
<b>4 Basket Optimisation Example</b> .....	5
4.1 Accessing TimeScape from MATLAB .....	6
4.2 Load FTSE 100 Index basket definition from TimeScape.....	7
4.3 Load closing prices for each index constituent and index closing price.....	8
4.4 Utilise MATLAB to calculate weights for each index constituent. ....	9
4.5 Create a MATLAB timeseries from the calculated data.....	9
4.6 Use MATLAB to plot the calculated and actual closing prices.....	10
4.7 Save weighted basket definition to TimeScape.....	11
<b>5 Principal Component Analysis Example</b> .....	12
5.1 Create a matrix of the weighted closing prices for each component and calculate their covariance.....	12
5.2 Calculate eigenvalues and eigenvectors .....	13
5.3 Plot Principal Components.....	13
5.4 Create TimeScape properties to save eigenvalues and eigenvectors.....	14
5.5 Save eigenvalues and eigenvectors to TimeScape .....	15
<b>6 Alternative approaches and future direction</b> .....	17
<b>7 Conclusion</b> .....	19
<b>8 Technical Appendix</b> .....	20
8.1 TimeScape COM server / data access .....	20
8.2 Optional Arguments.....	20
8.3 Handling TimeScape Errors .....	21
8.4 TimeSeries.....	21
8.5 Mutliple Time Series .....	22
8.6 Reference Properties .....	22
8.7 Matrix Properties .....	23
8.8 List Properties .....	24
8.9 Queries.....	24
8.10 Saving Data to TimeScape.....	25

# 1 Introduction

---

[MATLAB](#) is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. MATLAB code is widely used for financial modelling and analysis and can be integrated with other languages and applications, and distributed as algorithms and applications.

Xenomorph TimeScope provides a complete analytics and data management suite that encompasses the capture, cleansing, validation, storage and analysis of complex data types. Its normalised data model enables data consistency, quality and auditability despite disparate and distributed sources, and presents a single common view for all downstream users and systems.

TimeScope's interface and query language enable the fast, easy analysis of historic data of any type and frequency, alongside hundreds of ready-to-run functions and an extensible framework for the easy integration of proprietary calculations.

This paper describes how TimeScope can be easily embedded within MATLAB using the TimeScope COM Interface as a source of data and analytics. It is intended for MATLAB users who wish to combine TimeScope analytics and data with MATLAB computational and visualization tools, to produce best-of-breed analytical solutions.

## 2 Objectives

---

This paper demonstrates how the strengths of TimeScope and MATLAB can be combined to provide enhanced data analysis and visualisation tools to organisations. In particular, it details how:

- TimeScope data and analytics can be accessed from MATLAB;
- MATLAB computational and visualization tools can be used to manipulate and analyse TimeScope data;
- Complex data sets generated in MATLAB can be saved back to TimeScope for persisted storage;
- MATLAB components can be called from TimeScope to enrich TimeScope hosted functionality.

This is achieved by exploring the following simple analytical scenarios which suitably demonstrate the power and advantages of the combined solution:

- Basket optimisation to track a reference series based on market data stored in TimeScope;
- Principal component analysis of a time-series based on a number of reference series.

### 3 TimeScope and MATLAB Integration

TimeScope provides a number of application programming interfaces (APIs) that allow external systems to access TimeScope data and analytics externally. Of these, the TimeScope Component Object Model (COM) API is particularly suited to interfacing from MATLAB to TimeScope as this can be used from both the 32 bit and 64 bit (via COM+) versions of MATLAB.

It should be noted that MATLAB also supports the ability to call external APIs written for Microsoft .NET and as such the TimeScope .NET API could also potentially be used for this purpose. However, this approach is not yet particularly well suited to accessing the full suite of TimeScope functionality due to various limitations that exist within the MATLAB .NET environment (as of 2011a).

Please refer to the MATLAB web site at:

[http://www.mathworks.com/help/techdoc/matlab\\_external/brpb5k6-1.html](http://www.mathworks.com/help/techdoc/matlab_external/brpb5k6-1.html)

for more details on those limitations.

The following diagram summarises the benefits that are immediately exposed to end-users by connecting TimeScope and MATLAB together via the TimeScope COM API:

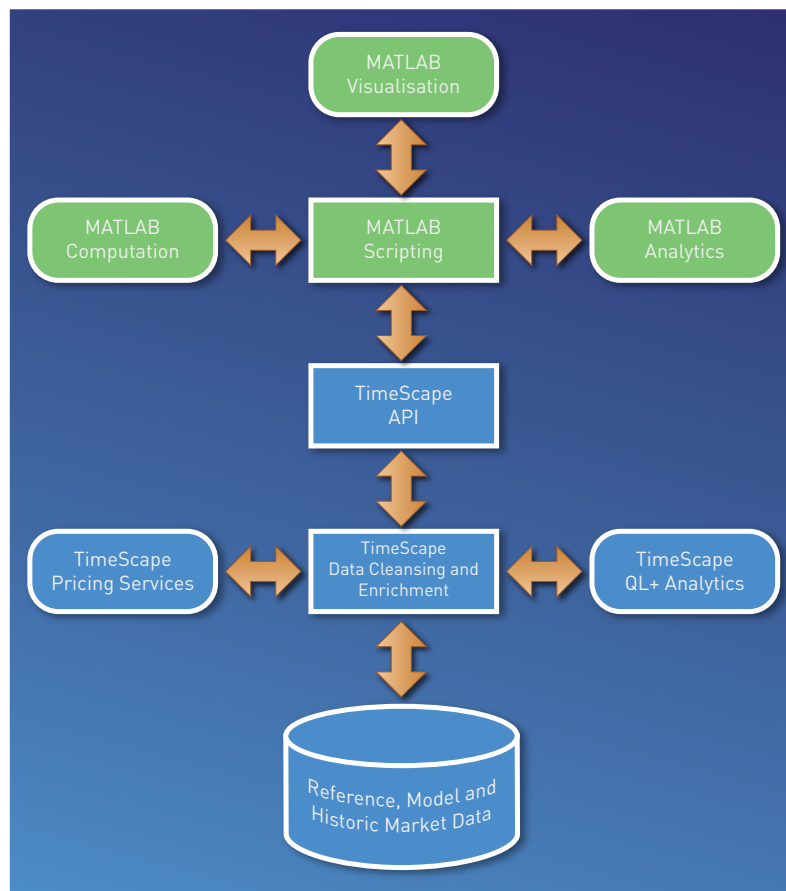


Figure 1 - Diagram of MATLAB and TimeScope Integration

Once connected, users are able to immediately combine data and analytics managed in TimeScope with the power of the MATLAB scripting and visualisation tools to produce combined solutions that harness the advantages of both environments.

The following sections within this paper provide some real-world examples that demonstrate the TimeScope-MATLAB integration in action. In addition, they also provide further detailed technical information on how that integration is achieved in practise.

## 4 Basket Optimisation Example

This example demonstrates the necessary steps to:

- Load data from TimeScope into MATLAB;
- Analyse the loaded TimeScope data and plot the results in MATLAB;
- Save the results of the MATLAB data analysis back to TimeScope for persisted storage.

To do this, we consider the example of an analyst who wishes to optimise the weightings of an index basket using MATLAB and who has all necessary time series and basket composition data already available in TimeScope. The example database 'XENO' provided with the standard installation of TimeScope contains an example basket definition for the FTSE 100 Index in which the components of the index are defined, although the weights of each component are equal to 1 (see Figure 2 screenshot below).

	Weight	Item 1	Item 2
1	1	A.A.L.L	
2	1	B.O.O.T.L	
3	1	H.A.S.L	
4	1	N.X.T.L	
5	1	S.H.E.L.L	
6	1	A.B.F.L	
7	1	B.P.L	
8	1	H.B.O.S.L	
9	1	O.M.L.L	
10	1	S.H.P.L	
11	1	A.L.L	
12	1	B.S.Y.L	
13	1	H.G.L	
14	1	O.O.M.L	
15	1	S.M.I.N.L	
16	1	A.L.L.D.L	
17	1	B.T.L	
18	1	H.N.S.L	
19	1	P.R.U.L	
20	1	S.N.L	

**Figure 2 - Un-weighted FTSE constituents in TimeScope held in the XENO database on the computer MATLAB (being viewed in TimeScope Workbench)**

To optimise this basket, our analyst can use the TimeScape COM API to:

1. Access TimeScape from MATLAB;
2. Load the FTSE index basket from TimeScape;
3. Load the cleansed closing prices for each index constituent and that of the index closing price itself from TimeScape;
4. Utilise MATLAB to optimise the weights for each index constituent so that the difference between the sum of the weighted basket series and the index closing price is minimised;
5. Calculate a MATLAB time series from the optimised weights and the constituent closing price data loaded from TimeScape;
6. Use MATLAB to plot the calculated and actual closing prices;
7. Save the optimised weighted basket definition back to TimeScape for persisted storage.

The example illustrates:

- Loading and saving complex TimeScape data types such as lists and timeseries into MATLAB;
- The speed with which TimeScape can load over 100 aligned timeseries into MATLAB;
- The reduced complexity of the MATLAB code as there are no 'NaN' values in the aligned series to consider;
- The mapping between a TimeScape timeseries and a MATLAB timeseries.

In order to maintain the readability of the examples provided, the error checking code has been excluded. The user should refer to Appendix 8.3 for programming guidelines on how best to handle TimeScape error information within MATLAB applications.

#### 4.1 Accessing TimeScape from MATLAB

MATLAB provides the ability for functionality exposed by external systems to be accessed programmatically from MATLAB scripts. This connectivity is supported in a number of ways which includes accessing functionality via Microsoft .NET Framework assemblies or via Microsoft Component Object Model (COM) libraries.

TimeScape provides full support for both of these programming models. However, at the time of writing (and as discussed in section 3) the MATLAB development environment is limited in its support for certain.NET features which prevents the full set of TimeScape functionality from being consumed. For this reason, the TimeScape COM API is used to demonstrate the integration of TimeScape and MATLAB together (although the same principles can equally be applied to an integration based on using the TimeScape .NET API as and when MATLAB fully supports the .NET environment).

To connect MATLAB to TimeScape, the analyst simply needs to register a connection with the TimeScape COM library from the MATLAB environment by using the MATLAB function `actxserver()`. Once that connection has been made, the desired TimeScape functions can then be called from within that connection as required.

The typical MATLAB code that is required to connect to TimeScape from MATLAB is as follows:

```
% Open TimeScape COM API interface
Timescape = actxserver('XTDataServices.XenoVB');
```

In this case, the `XenoVB` class of functions (part of the TimeScape COM API library) are being registered with MATLAB. This makes the suite of TimeScape `Xvb` functions (such as `XvbLoadProp()`, `XvbLoadSeries()` etc) available to MATLAB.

This can be illustrated by executing the following code:

```
% List all Xvb functions
XvbFunctions = Timescape.methods
```

This lists all available functions provided by the TimeScape connection. In addition, you can also use the `'-full'` parameter as follows:

```
% List all Xvb functions with function signatures
XvbFunctionsWithDetails = Timescape.methods( '-full' )
```

This lists all the available functions along with their signatures.

## 4.2 Load FTSE 100 Index basket definition from TimeScape

The first stage in optimising the TimeScape FTSE 100 Index basket from within MATLAB is to load the basket definition for the Index from TimeScape. To do this, a TimeScape connection is opened and then the TimeScape API function `XvbLoadProp()` is called to load the index composition for the FTSE 100 item using the `.FTSE` security identifier. Finally, the list of component item references (which identify the instruments contained in the basket) is extracted from the second column of the basket definition using standard MATLAB array indexing. This is illustrated in the following code fragment:

```
% Open TimeScape COM API interface
Timescape = actxserver('XTDataServices.XenoVB');

% Load FTSE 100 Index basket definition using the .FTSE identifier
FTSEBasket = Timescape.XvbLoadProp( 'XENO', '.FTSE', 'X_Basket Definition' );

% Extract component item references from the second column of the array
FTSEComponents = FTSEBasket(:,2);
```

Note:

Within TimeScope, item references are arrays containing 3 rows that identify the database location and identity of each individual index component item. They have the following content:

Row	Name	Description	Example
1	Database	The database containing the item	"XENO"
2	Code Type	The type of security identifier used to identify the item	"Reuters"
3	Code	The security id that identifies the item (expressed in the 'Code Type' coding system)	"AAL.L"

**Table 1 – TimeScope item reference array structure**

#### 4.3 Load closing prices for each index constituent and index closing price

The next stage in the process is to load the closing price series for the index components and the index itself. This is done using the standard TimeScope API function `XvbLoadNSeries()` which loads and aligns multiple timeseries and returns them as an array to MATLAB. This is illustrated in the following code:

```
% Create arrays of index component items in preparation for loading
% timeseries, using the database (row 1) and code (row 3) for each item
for i=1:length( FTSEComponents )
    Databases(i) = { FTSEComponents{i}{1} };
    Codes(i) = { FTSEComponents{i}{3} };
    Properties(i) = { 'Close' };
end
% Append the FTSE 100 index to the end of the components, so that we can load
% closing data for it as well
Databases( length(FTSEComponents) + 1 ) = { 'XENO' };
Codes( length(FTSEComponents) + 1 ) = { '.FTSE' };
Properties( length(FTSEComponents) + 1 ) = { 'Close' };

% Load aligned, closing price timeseries for all components and the FTSE
% (transform column-wise arrays to row wise using ')
XenoTS = Timescape.XvbLoadNSeries(Databases',Codes',Properties');
```

Following the execution of this code, the list of closing price timeseries for the instruments is contained within the variable `XenoTS` as an array of TimeScope timeseries arrays.

#### 4.4 Utilise MATLAB to calculate weights for each index constituent.

The next stage in the process is to optimise the index basket subject to the target constraints. In this case, we use the standard MATLAB function `lsqnonneg()` to optimise the weights for each index constituent so that the difference between the sum of the weighted basket series and the index closing price is minimised over time:

```
% The closing price timeseries loaded from Timescape are date-aligned with:
% - no missing values (NaN)
% - Dates are in column 1
% - Components closing prices are in columns 2 to (NumberOfColumns-1)
% FTSE Closing prices is contained in the last column
NumberOfColumns = size( XenoTS, 2 )
% Use lsqnonneg() to calculate component weights
FTSEWeights = lsqnonneg( XenoTS(:,2:NumberOfColumns-1),
XenoTS(:,NumberOfColumns) )
```

Following the execution of this code, the optimised weights for each of the component instruments are contained within the variable `FTSEWeights` as an array of numbers.

#### 4.5 Create a MATLAB timeseries from the calculated data

Now that the index weights have been optimised, we can use them to generate a retrospective theoretical series for the index based on the closing prices of the component instruments. This can be done by using standard MATLAB vector arithmetic and then transforming the resultant timeseries into MATLAB terms by applying a date offset:

```
% Construct a closing series from the optimized weights by scaling and
% summing the component time series values located in columns 2 to
% (NumberOfColumns-1)
FTSEDerivedClose = XenoTS(:,2:NumberOfColumns-1) * FTSEWeights

% Create a MATLAB timeseries for the derived (theoretical) closing price
% - add a date offset onto the TimeScale date values returned in column 1
% of the timeseries array to transform them into MATLAB-equivalent terms
FTSEDerived_TS = timeseries( FTSEDerivedClose,
                             datestr( XenoTS(:,1) + datenum( '30-Dec-1899' ) ));
FTSEDerived_TS.TimeInfo.Unit = 'day'

% For comparison purposes, create a MATLAB time series for the observed
% (ie.loaded) closing price
FTSEClose_TS = timeseries( XenoTS(:,NumberOfColumns),
                             datestr( XenoTS(:,1) + datenum( '30-Dec-1899' ) ) )
FTSEClose_TS.TimeInfo.Unit = 'day'
```

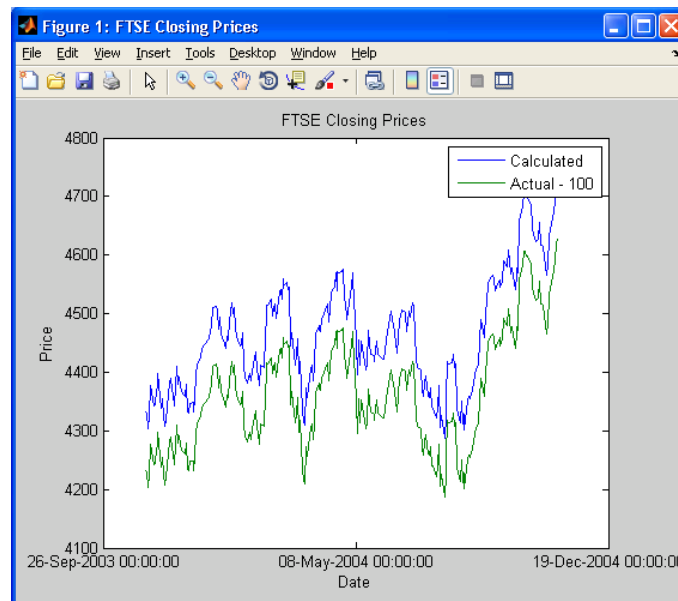
#### 4.6 Use MATLAB to plot the calculated and actual closing prices.

The actual observed market price for the FTSE 100 Index and that calculated for the theoretical (optimised) basket are now ready to be displayed graphically within MATLAB for comparison purposes. This is easily done using the built-in plot() functionality that is provided by MATLAB.

To make it easier to compare the characteristics of the two timeseries we also subtract 100 from the actual observed time series (so that it is shifted downwards on the graph) as follows:

```
% Display a simple graph with both series - (adjust Actual Price)
figure( 'Name', 'FTSE Closing Prices' )
plot( FTSEDerived_TS );
% Display on same figure
hold all;
plot( FTSEClose_TS - 100 );
hold off;
ylabel( 'Price' );
xlabel( 'Date' );
legend( 'Calculated', 'Actual - 100' );
title( 'FTSE Closing Prices' );
```

This results in the following chart then being displayed in MATLAB:



**Figure 3 – MATLAB plot of the calculated (theoretical) index price vs. actual price (shifted by 100)**

This graph clearly illustrates that the shapes of the observed and theoretical price series match, as would be expected from the weight optimisation process which is designed to minimise the error between them.

#### 4.7 Save weighted basket definition to TimeScape

The final stage in the process is to demonstrate that data generated in MATLAB can then be saved (persisted) into TimeScape. In this case, we will demonstrate that the optimised basket composition can be saved back to TimeScape as a TimeScape list using the TimeScape API function XvbSaveList(). The following code fragment illustrates how this can be done:

```

% Update the weights in the FTSE 100 basket definition that was loaded with
% the new optimised weights
for i = 1:length(FTSEComponents)
    FTSEBasket(i,1) = { FTSEWeights(i) };
end
% Save the optimised weights back to Timescape, overwriting any existing
% definition
Timescape.XvbSaveList( 'XENO', '.FTSE', 'X_Basket Definition', FTSEBasket )
    
```

After executing this MATLAB code, we then see that the content of the XENO database has changed to reflect the newly optimised basket weights:

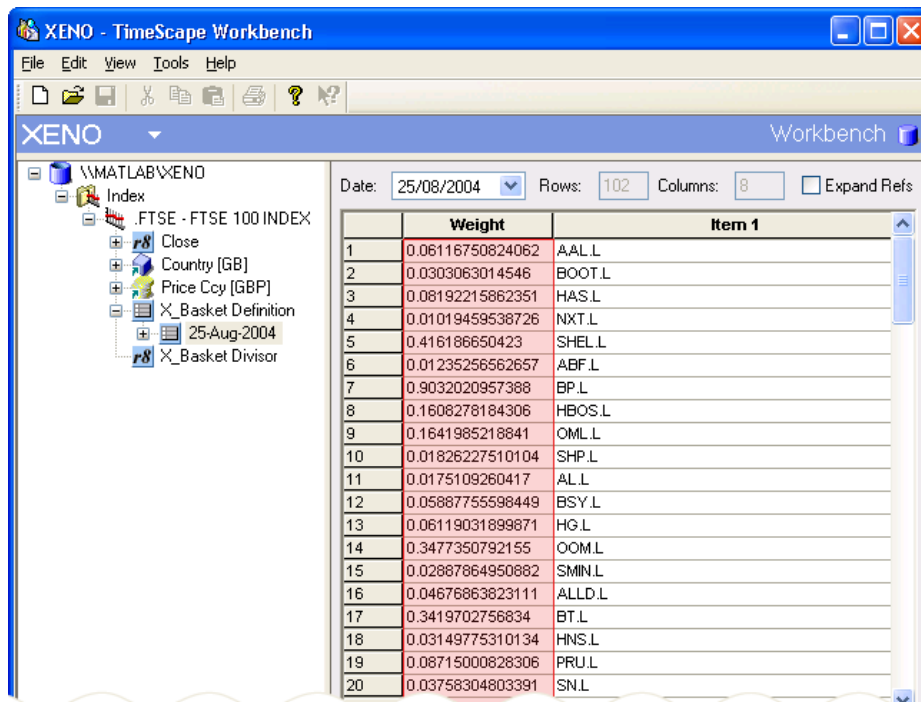


Figure 4 – Newly optimised FTSE constituent weightings being viewed in TimeScape Workbench

## 5 Principal Component Analysis Example

---

In this example, we continue to explore the FTSE 100 basket that was optimised in the previous example in section 4. Having calculated the optimised weights, we now use these to construct an array with the weighted closing price of each constituent and then perform a principal component analysis on that data.

As part of this principal component analysis, we first use MATLAB to calculate the eigenvalues and eigenvectors and we then use the TimeScape API to store that derived data in TimeScape. In addition, we also demonstrate using the TimeScape API to automatically create the necessary database schema within TimeScape to store this derived data.

In particular, the following sections demonstrate how to use the TimeScape and MATLAB integration to:

1. Calculate a matrix of the weighted component closing prices and their covariance;
2. Calculate eigenvalues and eigenvectors for the data set;
3. Plot the principal components for the data set graphically;
4. Create TimeScape properties to store the calculated eigenvalues and eigenvectors;
5. Save the calculated eigenvalues and eigenvectors to TimeScape.

The example illustrates

- The ease with which the TimeScape data model can be extended;
- The simplicity of persisting MATLAB matrix data within TimeScape;
- Using optional arguments to TimeScape COM API functions within MATLAB;
- Simple error checking of TimeScape COM API calls.

### 5.1 Create a matrix of the weighted closing prices for each component and calculate their covariance

The first stage in the process is to calculate the weighted closing prices and their covariance. Continuing from the example in section 4, we use the calculated weights that are stored in `FTSEWeights` and combine them with the prices stored in `XenoTS` to generate a matrix of weighted closing prices `WeightedClosingPrices`. We then use the standard MATLAB function `cov()` to calculate the covariance of those weighted closing prices as illustrated below:

```
% Create a matrix of the weighted closing prices
[m,n] = size( XenoTS(:,2:NumberOfColumns-1) )
for i = 1:n
    WeightedClosingPrices(:,i) = FTSEWeights(i) * XenoTS(:,i+1);
end
% Calculate the covariance the weighted closing prices
FTSEConstituents_cov = cov( WeightedClosingPrices );
```

This results in the covariance of the constituents being stored in the variable `FTSEConstituents_cov`.

## 5.2 Calculate eigenvalues and eigenvectors

The eigenvalues and eigenvectors for the covariance matrix are then calculated using the standard MATLAB function `eig()` as follows:

```
% Calculate eigenvalues and eigenvectors on the covariance matrix
[ EigenVectors, EigenValues ] = eig( FTSEConstituents_cov );
EigenValues = diag( EigenValues );
```

Here, the `eig()` function returns the eigenvalues on the diagonal of the `EigenValues` matrix when called in this way (see: <http://www.mathworks.com/help/techdoc/ref/eig.html>). As a result, the MATLAB function `diag()` is then subsequently being used to extract those eigenvalues and place them into a separate matrix.

## 5.3 Plot Principal Components

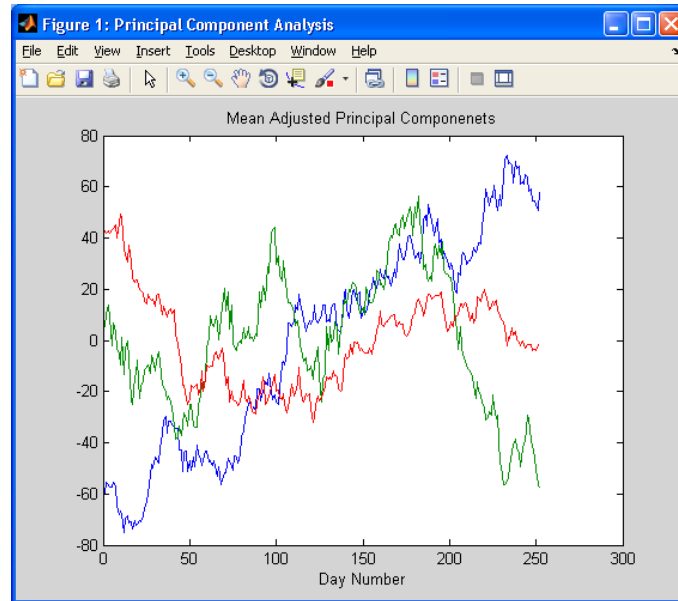
The principal components are now in a position to be calculated and charted graphically within MATLAB. The following code fragment handles this aspect of the data processing by calculating and plotting the mean adjusted principal components:

```
% Sort eigenvalues and eigenvectors into descending principal component order
[ SortedEigenValues, SortedIndex ] = sort( EigenValues, 'descend' );
% Create chart and plot
figure( 'Name', 'Principal Component Analysis' );
plot( PCAtransform( meanadjust( WeightedClosingPrices ),
    EigenVectors( :, SortedIndex ), 3 ));
title( 'Mean Adjusted Principal Components' );
xlabel( 'Day Number' );
```

In this case the following MATLAB code is used within `PCAtransform()` to convert the data sets to the correct dimensions:

```
%-----
% this function makes the transformation to the eigenspace for PCA
% parameters:
% adjusteddata = mean-adjusted data set
% eigenvectors = SORTED eigenvectors (by eigenvalue)
% dimensions = how many eigenvectors you wish to keep
%
% The first two parameters can come from the result of calling
% PCAprepare on your data.
% The last is up to you.
function [finaldata] = PCAtransform(adjusteddata,eigenvectors,dimensions)
    finaleigs = eigenvectors(:,1:dimensions);
    prefinaldata = finaleigs'*adjusteddata';
    finaldata = prefinaldata';
```

This results in the following plot being produced in MATLAB:



**Figure 5 – Graphical MATLAB plot of the mean adjusted principal components of the weighted component time series**

#### 5.4 Create TimeScape properties to save eigenvalues and eigenvectors

The next stage in the processing is to demonstrate how the calculated eigenvalues and eigenvectors can be easily stored in TimeScape from MATLAB.

To do this, the TimeScape database schema has first to be extended (if necessary) to support two new matrix data properties within the TimeScape Index database category (these are named [Eigenvectors](#) and [Eigenvalues](#)).

This is achieved by calling the TimeScape COM API function `XvbAddPropertyDef()`, with the necessary arguments to add the matrix properties to the Index category, as illustrated in the following code fragment:

```

% Create Timescape Matrix Properties (DataType=9) to store eigenvalues and
% eigenvectors- use the invoke method with a lower case version of the method
% name in order to provide optional arguments - ignore any error conditions
% for the time being as if they're important they will be detected when we
% attempt to save data

% Property data type = matrix (9)
MatrixDataType = 9;
% Add 'EigenValues' to Index category
res = Timescape.invoke('xvbaddpropertydef', 'XENO', 'Index', 'Eigenvalues',
    'Eigenvalues data for components', MatrixDataType,
    true );
% Add 'EigenVectors' to Index category
res = Timescape.invoke('xvbaddpropertydef', 'XENO', 'Index', 'Eigenvectors',
    'Eigenvectors data for components', MatrixDataType,
    true );

```

Once these properties have been defined, then data can be saved to them from MATLAB as described in the following section.

### 5.5 Save eigenvalues and eigenvectors to TimeScope

The final stage in the process is to save the calculated eigenvalues and eigenvectors matrix data to TimeScope from MATLAB. This is achieved using the TimeScope COM API function `XvbSaveMatrix()` as illustrated below:

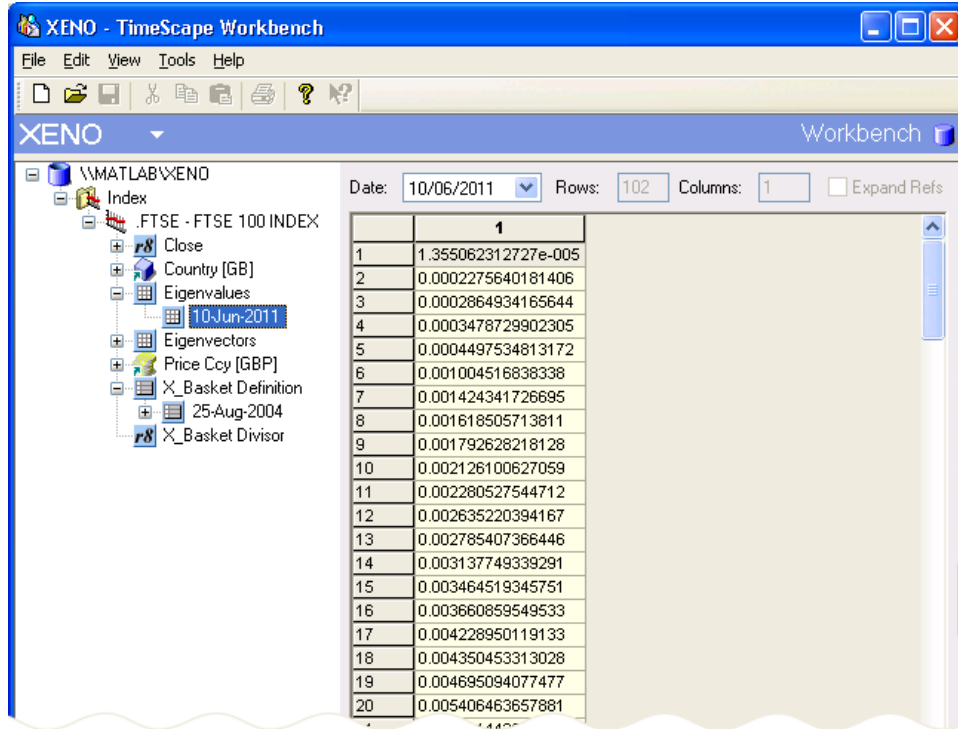
```

% Save eigenvalues
res = Timescape.XvbSaveMatrix( 'XENO', '.FTSE', 'Eigenvalues', EigenValues );
if res ~= 1
    error( ['Failed to save property Eigenvalues to Item .FTSE - '
        Timescape.XvbLastError()] )
end
% Save eigenvectors
res = Timescape.XvbSaveMatrix( 'XENO', '.FTSE', 'Eigenvectors', EigenVectors );
if res ~= 1
    error( ['Failed to save property Eigenvectors to Item .FTSE - '
        Timescape.XvbLastError()] )
end

```

In addition to demonstrating how the matrix property data can be saved to TimeScope, this example also illustrates how descriptive error information can be retrieved from TimeScope. This is done using the `XvbLastError()` function that is supplied by the TimeScope COM API. This function returns a string value describing the reason for the last API function failure. For more information on this function and in handling TimeScope errors in general from within MATLAB, you should refer to appendix section 8.3.

Once the MATLAB script has been executed, we then observe that the eigenvalue and eigenvector data has been successfully stored in TimeScape, as illustrated in the following screenshot:



**Figure 6 – Calculated eigenvalue and eigenvector data viewed in TimeScape Workbench following a successful save from MATLAB to TimeScape**

## 6 Alternative approaches and future direction

---

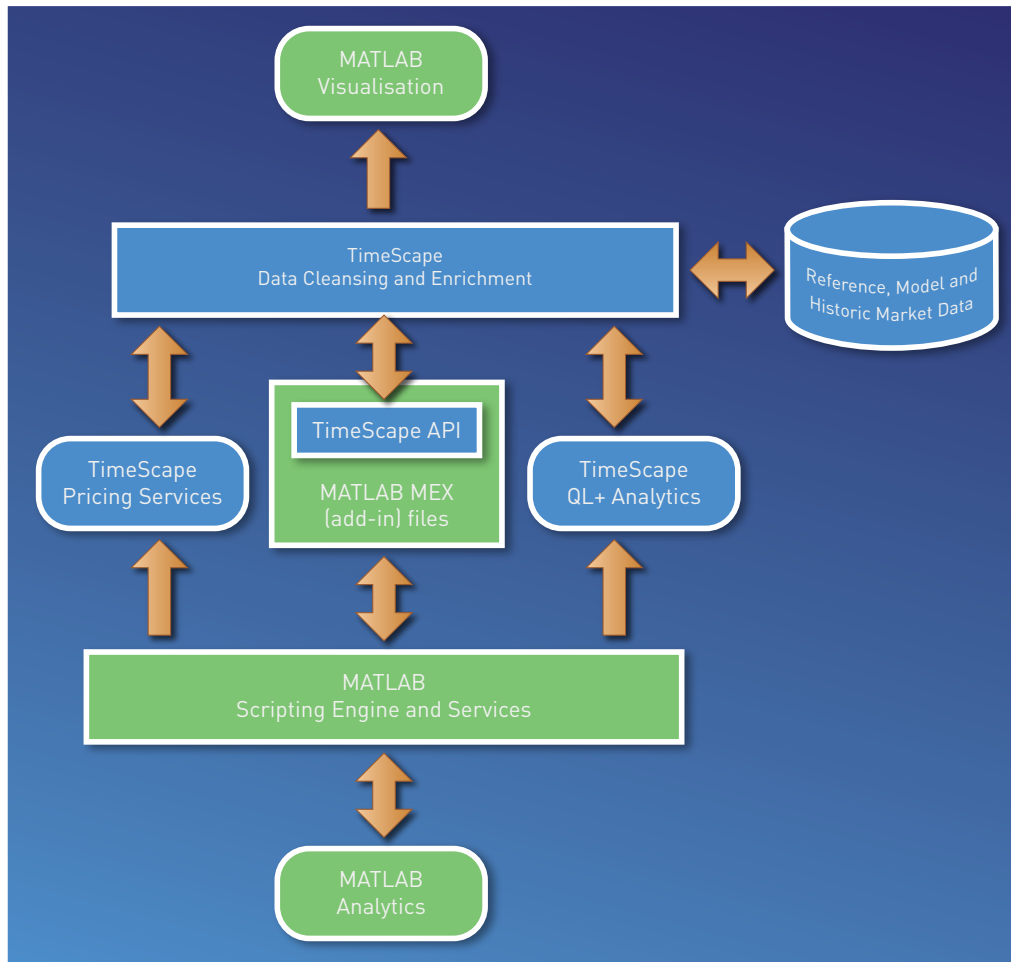
This white paper has focused to date on embedding TimeScape within MATLAB by using the TimeScape COM API to show how TimeScape data can be loaded into MATLAB, analysed within MATLAB scripts and the results optionally saved back for storage in TimeScape.

However, TimeScape also provides a number of other APIs that can also be used to provide further integration options with MATLAB. These allow developers to embed TimeScape deeper within MATLAB or to utilise MATLAB analytics from within TimeScape.

The following approaches are available:

- Building a customised MATLAB mex file using the TimeScape C++ API. MATLAB provides the concept of mex files which allow developers to create and register add-in functions for MATLAB that can then be used as native functions within the MATLAB environment (see: [http://www.mathworks.com/help/techdoc/matlab\\_external/f29322.html](http://www.mathworks.com/help/techdoc/matlab_external/f29322.html) for more details). As a result, this technique can be used to develop native functions that utilise the TimeScape C++ API.
- TimeScape Analytic Addins can be developed (using the TimeScape Analytics SDK) that execute MATLAB functions and services from within them to analyse data. Once created, the MATLAB functionality can then be utilised within TimeScape queries and applications as needed.
- TimeScape Pricing Addins can be developed (using the TimeScape Pricing SDK) that execute MATLAB functions and services from within them to perform instrument pricing. Once created, these pricing algorithms can then be utilised by TimeScape Pricing Services and applications as needed.

These development options are illustrated in the following diagram:



**Figure 7 – Additional MATLAB integration options with TimeScape**

MATLAB has also been extended with a data feed toolbox that offers interfaces to a number of data vendors including both Reuters and Bloomberg. Key features of the datafeed toolbox are that it:

- Integrates the numerical, computational, and graphical capabilities of MATLAB with financial data providers
- Provides access to market, time series, and historical market data in MATLAB
- Fetches data fields for multiple securities in a single call
- Enables you to look up security ticker symbols from the MATLAB command line or the toolbox GUI

These features are readily accessible through the TimeScape interfaces, but can be wrapped into toolkit functions that support the fetch and insert methods utilised in the datafeed toolbox. This would be a first step towards adding TimeScape as a supported data provider in the MATLAB datafeed toolbox.

For more information see: <http://www.mathworks.com/products/datafeed/>

## 7 Conclusion

---

The examples provided in this document have shown the ease, speed and simplicity with which TimeScape can be integrated with MATLAB. Complex data types such as MATLAB cell arrays map simply onto the TimeScape List structure for both storage and retrieval, whilst the basic MATLAB matrix and time series data types have a direct equivalent in TimeScape.

The flexibility and extensibility of the TimeScape data model have been used to demonstrate how TimeScape's best of breed data capture and storage can be combined with the analytical capabilities of MATLAB to produce compelling solutions to real-world problems encountered within financial services.

Although not explicitly demonstrated, it is important to note that exactly the same interfaces that are described in this document for accessing data stored in a TimeScape database can also be used to access real time data directly from any supported data feed that has been integrated into TimeScape (e.g. Reuters and Bloomberg).

## 8 Technical Appendix

---

### 8.1 TimeScape COM server / data access

TimeScape can be called from MATLAB using `actxserver` to create a TimeScape COM server.

```
>> Timescape = actxserver( 'XTDataServices.XenoVB' )
```

The list of methods available can be viewed using:

```
>> Timescape.methods           % List of all methods
>> Timescape.methods( '-full' ) % List of all methods with function signature
```

In general, where the output of a TimeScape function is a variant data type this will be loaded into MATLAB in a cell array. Numerical data can be easily converted to/from MATLAB matrices using the functions `cell2mat()` and `mat2cell()`.

TimeScape timeseries can be easily converted to MATLAB time series, but the Excel style dates may need to be adjusted to MATLAB dates with the addition of the value:

```
datenum( '30-Dec-1899' )
```

which will adjust the date values with the correct offset to make them equivalent.

### 8.2 Optional Arguments

In comparing the function signature for `XvbLoadProp()` with the TimeScape documentation, we can see that the optional arguments have been lost. This is due to a limitation in the way MATLAB interfaces with ActiveX components. For example, we observe the following function signature in MATLAB:

```
XvbLoadProp( handle, string, string, string )
```

when the full function signature is really as follows:

```
XvbLoadProp( Database, Code, PropName, DateOpt, CodeTypeOpt, DataSrcOpt,
             MissingRuleOpt ) As Variant
```

Properties can be loaded using the method but, if required, the optional arguments may have to be specified as part of the property argument.

```
>> Timescape.XvbLoadProp( 'XENO', 'AAL.L', 'Close("Reuters")' )
```

An alternative way of specifying the optional arguments is to invoke the method using the lower-case version of its name:

```
>> Timescape.invoke( 'xvbloadprop', 'XENO', '.FTSE', 'Close', [], [], 'Reuters' )
```

where empty arguments are passed as `[]`.

### 8.3 Handling TimeScape Errors

The TimeScape COM API returns errors as VT\_ERROR variant values that contain the underlying TimeScape error code as part of that VT\_ERROR value. Programmers developing with Microsoft Visual Basic or Microsoft Visual Basic for Applications ordinarily check whether the variant contains an error value using, for example, the standard Visual Basic function `IsError()`. Once this is detected, they then typically call the TimeScape COM API function `XvbError()` to convert that VT\_ERROR variant to a text value that describes the error.

However, MATLAB does not support the VT\_ERROR variant value and simply converts the error variant to a string value containing the value "ActiveX VT\_ERROR". In order to overcome this limitation, it is therefore necessary to use an alternative technique to detect and decode the error. In the event that the string "ActiveX VT\_ERROR" is detected, MATLAB script developers should utilise another TimeScape COM API function `XvbLastError()` to decode the error. This function returns a string value describing the most recent error condition encountered within the TimeScape COM API. As such, it is suitable for use in providing error diagnosis within MATLAB scripts which offer no support for VT\_ERROR.

The following code fragment illustrates this technique:

```
% Execute a load series request for a non-existent item code
Series=TimeScape.invoke('xvblogseries', 'XENO', 'BAD ITEM CODE', 'Close');
% If the return value contains the text 'ActiveX VT_ERROR' then extract the error
% message
if strfind(Series, 'ActiveX VT_ERROR')
    Error=XvbLastError();
end
```

In this case, the `Error` variable then contains the string "Error 1 (type 16): The database item could not be found".

### 8.4 TimeSeries

When a time series is loaded into MATLAB from TimeScape, it is loaded as a 2 column array of dates and values. The dates are in string format, and the values are loaded as a cell array. For example:

```
>> XenoTS = Timescape.XvbLoadPropSeries( 'XENO', 'AAL.L', 'Close' )
XenoTS =

    '24/07/2002' [    915]
    '25/07/2002' [    959]
    '26/07/2002' [    854]
    ...
    '07/01/2005' [   1224]
    '10/01/2005' [   1221]
    '11/01/2005' [   1223], ,
```

This can be converted to a MATLAB timeseries `MATLAB_TS` using the following technique:

```
>> MATLAB_TS = timeseries(cell2mat( XenoTS(:,2) ), datenum( XenoTS(:,1),
    'dd/mm/yyyy' ) )
>> MATLAB_TS.TimeInfo.Unit = 'day'
```

### 8.5 Multiple Time Series

In order to load multiple series using `XvbLoadNSeries()`, the input parameters must be specified as row arrays e.g.

```
>> Databases = { 'XENO'; 'XENO' }
>> Codes = { 'AAL.L'; '.FTSE' }
>> Properties = { 'Close'; 'Close' }

>> XenoTS = Timescape.XvbLoadNSeries( Databases, Codes, Properties )
```

In this case the output is returned as a three column matrix with Excel style dates in the first column and the aligned time series data in columns two and three.

```
XenoTS =

    1.0e+004 *

    3.7461    0.0915    0.3777
    3.7462    0.0959    0.3966
    3.7463    0.0854    0.4017
    ...
    3.8359    0.1224    0.4854
```

This can be converted to a MATLAB timeseries as follows:

```
>> MATLABTS = timeseries( XenoTS(:,2:3), XenoTS(:,1) + datenum( '30-Dec-1899' ) )
```

### 8.6 Reference Properties

TimeScope item reference properties are loaded into MATLAB as one-dimensional arrays composed of three rows as follows:

Row	Name	Description	Example
1	Database	The database containing the item	"XENO"
2	Code Type	The type of security identifier used to identify the item	"Reuters"
3	Code	The security id that identifies the item (expressed in the 'Code Type' coding system)	"AAL.L"

Hence, the following request to load the `Country` associated with `AAL.L`:

```
>> Timescape.XvbLoadProp( 'XENO', 'AAL.L', 'Country' )
```

results in the following answer being obtained:

```
ans =
    '\\MATLAB\COMMON'
    'COUNTRY ISO'
    'GB'
```

## 8.7 Matrix Properties

Xenomorph matrix property data is loaded into MATLAB as a cell array. Note that for this example the property `Matrix` was created and does not exist in the example XENO database installed with TimeScope.

Hence, we load matrix data in the normal way:

```
>> Timescape.XvbLoadProp( 'XENO', 'AAL.L', 'Matrix' )
```

which results in a cell array of data being loaded and returned to MATLAB as follows:

```
ans =
    [ 1]    [ 2]    [ 9]    [10]
   [-1]    [ 3]    [ 8]    [11]
    [ 3]    [ 4]    [ 7]    [12]
   [-5]    [ 5]    [ 6]    [12]
```

This cell array data can be converted to a MATLAB matrix using the MATLAB function `cell2mat()` as follows:

```
> cell2mat( Timescape.XvbLoadProp( 'XENO', 'AAL.L', 'Matrix' ))
```

which results in a matrix being produced as follows:

```
ans =
     1     2     9    10
    -1     3     8    11
     3     4     7    12
    -5     5     6    12
```

### 8.8 List Properties

TimeScope List data is loaded into MATLAB as arrays with data generally held in cell arrays, depending upon the list definition.

For example, if we load a basket definition:

```
>> XenoList = Timescape.XvbLoadProp( 'XENO', '.FTSE', 'X_Basket Definition' )
```

we find that the list data returned contains 8 columns where some contain data, empty or item reference (3x1) cell data:

```
XenoList =
    [1]    {3x1 cell}    ''    ''    [0]    [NaN]    ''    ''
    [1]    {3x1 cell}    ''    ''    [0]    [NaN]    ''    ''
    ...
    [1]    {3x1 cell}    ''    ''    [0]    [NaN]    ''    ''
    [1]    {3x1 cell}    ''    ''    [0]    [NaN]    ''    ''
```

### 8.9 Queries

A TimeScope timeseries query can return any shape of data depending on how the query is constructed. For example, following query will return data in a cell array with a text value (the Code) in the first column and a time series ( Close) embedded in the second column of the result:

```
>> XenoQuery = Timescape.XvbQuery( 'DatabaseItems( "XENO", "GB Equity",
                                     "Reuters", "A*" ).VALUES( Code, Close )' )
```

If we view this result we find it appears as follows:

```
XenoQuery =
    'AAL.L'    {625x2 cell}
    'ABF.L'    {628x2 cell}
    'AL.L'     {628x2 cell}
    'ALLD.L'   {626x2 cell}
    'ANL.L'    {581x2 cell}
    'ANTO.L'   {457x2 cell}
    'AUN.L'    {631x2 cell}
    'AV.L'     {621x2 cell}
    'AVZ.L'    {627x2 cell}
    'AZN.L'    {633x2 cell}
```

## 8.10 Saving Data to TimeScape

Data can be saved to TimeScape using the `XvbSave...()` suite of functions. However, given that the function signatures do not include the optional `CodeType` and `DataSource` parameters the invoke method with the lowercase version of the function name should generally be used.

In addition, date values need to be adjusted to Excel format dates by subtracting `datenum( '30-Dec-1899' )` from the MATLAB date value prior to save. For example, to save the value 1224 on 12-Jan-2008 the target save date must be adjusted prior to save as illustrated in the following code:

```
result = Timescape.invoke( 'xvbsavevalue', 'XENO', 'AAL.L', 'Close',  
                           datenum( '12-Jan-2008' ) - datenum( '30-Dec-1899' ), 1224 )
```

When dealing with a timeseries `TS` (as opposed to a single date) then, assuming the units are days, the MATLAB series can be saved to TimeScape using the following technique:

```
xDates = TS.Time - datenum( '30-Dec-1899' )  
xValues = TS.Data  
result = Timescape.invoke( 'xvbsavepropseries', 'XENO', 'AAL.L', 'Close', 3,  
                           xDates, xValues )
```

## About Xenomorph

---

Xenomorph delivers Analytics and Data Management (ADM) solutions to the financial markets.

Our TimeScape technology leverages our clients' proprietary expertise, enabling them to analyse and manage more data with greater control and transparency.

Our focus is to make our clients more successful by closing the productivity gaps between high performance database technology, data management and end-user data analysis. Through unified and transparent access to data and data analysis, our clients achieve even higher levels of financial innovation, business process efficiency and regulatory compliance.

Trading, research, risk, product control, IT and back-office staff use Xenomorph's TimeScape data management platform at investment banks, hedge funds and asset management institutions across the world's main financial centres.

Established in 1995, Xenomorph has offices in London, New York and Singapore.

---

For more information about Xenomorph:

Web: [www.xenomorph.com](http://www.xenomorph.com)

Email: [info@xenomorph.com](mailto:info@xenomorph.com)

London: +44 (0)20 7614 8600

New York: +1-212-401-7894

