insightsoftware

Developer's Guide

Longview

Version 26

Document Information

Notices

Copyright

The information contained in this document represents the current view of insightsoftware.com on the issues discussed as of the date of publication. This document is for informational purposes only. insightsoftware.com makes no representation, guarantee or warranty, expressed or implied, that the content of this document is accurate, complete or up to date.

Disclaimer

This guide is designed to help you to use the Longview applications effectively and efficiently. All data shown in graphics are provided as examples only. The example companies and calculations herein are fictitious. No association with any real company or organization is intended or should be inferred.

Contents

Longview Application Framework Overview

Longview Application Framework is the interface used for developing custom business logic within your Longview system.

Longview Application Framework performs the following tasks:

1. Handles interaction between the Longview Server and an already-deployed Longview App through the Longview API layer.
2. Can be run directly from a command prompt in batch mode, and permits the scheduling of tasks via third-party software.

Syntax with connection parameters for Longview and Windows Authentication

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax:

```
lv_af User/Password Host Port Role Group Proc [LanguageCode] [Param=Value]
```

where:

- User/Password is the Data Server user name and password. If using Windows authentication, use "/".
- Host is the host machine of the Data Server.
- Port is the listener port number of the Data Server.
- LID is the Longview Identifier of the Data Server.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ACCESS
- Group is the name of a user group.
- Proc is the name of the Procedure document to run, or the path of the Procedure document if it is in a different directory. The Procedure document must be a text file, and the file name can have any extension.
- LanguageCode is the optional language code, with a default value of EN.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Note: You must enclose values that contain spaces or Windows Command Shell special characters, such as ampersands (&), pipes (|), and parentheses, in double quotation marks.

Example 1: Code

```
lv_af Admin/Admin1 e3000 20001 LVProd V3_COMPATIBLE_ACCESS Analyst
proccopy.txt EN
```

Example 2: Code

```
lv_af Admin/Admin1 e3000 20001 LVProd V3_COMPATIBLE_ACCESS Administrators
proccopy.txt EN Entities=TCANADA
```

Example 3: Code

```
lv_af Admin/Admin1 e3000 20001 LVProd V3_COMPATIBLE_ACCESS Administrators
proccopy.txt EN Entities="Total Canada"
```

Example 4: Code

```
lv_af Admin/Admin1 e3000 20001 LVProd V3_COMPATIBLE_ACCESS Administrators
proccopy.txt EN Entities="Canada|USA"
```

Syntax with connection parameters for Single Sign-on authentication

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax for Single Sign-on Authentication:

```
lv_af ClientId/Secret SSO Scope Host Port LID Role Group Proc [LanguageCode]
[Param=Value]
```

where:

- ClientID/Secret is the Client ID set in the OAuth flow of your IDP to create the Longview session with. The Client ID must exist in the Longview Database as a third-party user.
- Scope is the value configured in the IDP application, if the IDP application and Longview SSO Service has been configured to use a scope. If the application does not use scope then use "".
- Host is the host machine of the Data Server.
- Port is the listener port number of the Data Server.
- LID is the Longview Identifier of the Data Server.

- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ACCESS.
- Group is the name of a user group.
- Proc is the name of the Procedure document to run, or the path of the Procedure document if it is in a different directory. The Procedure document must be a text file, and the file name can have any extension.
- LanguageCode is the optional language code, with a default value of EN.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Example 1: Code

```
lv_af 0oagk9dfghK83OO405d7/qPa-pz0D8mgi4zDZRR2E SSO "" e3000 9701 LVProd V3_
COMPATIBLE_ACCESS Administrators proccopy.txt EN Entities="Canada|USA"
```

Example 2: Code

```
lv_af 0oagk9dfghK83OO405d7/qPa-pz0D8mgi4zDZRR2E SSO LVRestAPI e3000 9701
LVProd V3_COMPATIBLE_ACCESS Administrators proccopy.txt EN
Entities="Canada|USA"
```

Syntax with connection parameters for ISW platform (AAD) authentication

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax when the Longview System is configured to use ISW platform with an Azure Active Directory service account:

```
lv_af ClientId/Secret AAD Tenant Environment ClientProxyURL LID Role Group
Proc [LanguageCode] [Param=Value]
```

where:

- ClientID/Secret is the application registered in the Azure Active Directory server that is used to request an access token.
- Tenant is the name of the Azure Active Directory server that is used to populate the AAD URL to request access token.
- Environment is the target B2C platform name that is used to populate the scope of the access token.

- ClientProxyURL is the URL of the client proxy
- LID is the Longview Identifier of the Data Server.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.
- Group is the name of a user group.
- Proc is the name of the Procedure document to run, or the path of the Procedure document if it is in a different directory. The Procedure document must be a text file, and the file name can have any extension.
- LanguageCode is the optional language code, with a default value of EN.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Example : Code

```
lv_af 4c796c6b-21u5-463b-8ddd-c5839e5c41e7/qPa-pz0D8mgi4zDZRR2E AAD
AzureServer iswb2c https://localhost:8080/client/ LVProd V3_COMPATIBLE_
ACCESS Administrators proccopy.txt EN Entities="Canada|USA"
```

Syntax with connection parameters for ISW platform organization key authentication

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax when the Longview system is configured to use ISW platform organization API Keys for authorization.

You must have an ISW platform organization key configured in your system to use this connect syntax.

```
lv_af APIKey APIKEY ClientProxyURL LID Role Group Proc [LanguageCode]
[Param=Value]
```

where:

- APIKey is the organization API Key that was generated in the ISW platform.
- ClientProxyURL is the URL of the client proxy.
- LID is your Longview identifier of the Data Server.
- Role is the name of the symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.

- Group is the name of the user group.
- Proc is the name of the Procedure document to run, or the path of the Procedure document if it is in a different directory. The Procedure document must be a text file, and the file name can have any extension.
- LanguageCode is the optional language code, with a default value of EN.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Example: Code

```
lv_af 1234567890abcdef1234567890abcdef12345678 APIKEY
https://localhost:8080/client/  LVProd V3_COMPATIBLE_ACCESS Administrators
proccopy.txt EN Entities="Canada|USA"
```

Syntax with connection parameters for ISW platform external (OPENID) authentication

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax when the Longview System is configured to use ISW platform with an external authentication service account:

```
lv_af ClientId/Secret OPENID Issuer Environment ClientProxyURL LID Role
Group Proc [LanguageCode] [Param=Value]
```

where:

- ClientID/Secret is the application registered in the Azure Active Directory server that is used to request an access token.
- Issuer is the external IDP issuer URI server that is used to request the access token.
- Environment is the target B2C platform name that is used to populate the scope of the access token.
- ClientProxyURL is the URL of the client proxy
- LID is the Longview Identifier of the Data Server.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.
- Group is the name of a user group.

- Proc is the name of the Procedure document to run, or the path of the Procedure document if it is in a different directory. The Procedure document must be a text file, and the file name can have any extension.
- LanguageCode is the optional language code, with a default value of EN.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Example: Code

```
lv_af 4c796c6b-21u5-463b-8ddd-c5839e5c41e7/qPa-pz0D8mgi4zDZRR2E OPENID
https://prod-u23s.us.auth0.com iswb2c https://localhost:8080/client/ LVProd
V3_COMPATIBLE_ACCESS Administrators proccopy.txt EN Entities="Canada|USA"
```

Syntax with connection parameters in Procedure document

```
lv_af Proc [Param=Value]
```

where:

- Proc is the text file that contains the commands or documents to execute or call, including connection parameters. This may include a command or an action to call or invoke another document.
- Param is optional and is the variable to which to pass the value. You can pass more than one parameter as long as the variable is unique.
- Value is the value to pass to the related Param.

Issuing commands

Longview Application Framework can be called from a System Administrator's computer, using the command prompt.

Commands and documents are available in Longview Application Framework. Commands may execute actions or call documents, which may invoke further commands. Documents are ASCII text files that outline parameters for actions, or additional actions to take.

The interface does not currently provide a Graphical User Interface (GUI) — it is command-line based. All parameter values are non-case sensitive.

Batch Mode syntax

To run Longview Application Framework in batch mode, the executable file (lv_af.exe) requires the following syntax:

```
lv_af Script
lv_af User/Password Host Port DSID AccessRole Script [LanguageCode]
```

For more information on calling commands from Longview Application Framework documents, see Understanding document types.

Understanding document types

A Longview Application Framework document is a text-based file containing instructions for Longview Application Framework. To ensure that all commands, functions, and Models invoked within your documents work as desired, you should save your Longview Application Framework documents as ASCII files rather than as Windows ASCII binary files.

Longview Application Framework has several types of documents used for different purposes. All Longview Application Framework documents can be run by users in batch mode, from a command prompt.

Caution: The use of the asterisk as a multiplication symbol in the application requires that it be enclosed in double quotation marks ("*"). An asterisk without double quotation marks is interpreted by the system as a wildcard search character, and operations that include it will not give the expected results.

Conforming to file extension conventions

Longview recommends that you follow standard naming conventions when you create Longview Application Framework documents.

Document type	Recommended Extension	Applies to...
Longview App Configuration File	lvapp	Longview Smart Client
DataArea Spec	lvdsp	Longview Smart Client Longview Application Framework
DataView definition	lvdvw	Longview Smart Client
DataTable definition	lvdtd	Longview Smart Client
Table View definition	lvtvw	Longview Smart Client
Calendar View definition	lvcvw	Longview Smart Client
Form document	lvfrm	Longview Smart Client
ImportSpec	lvimp	Longview Smart Client Longview Application Framework
ExportSpec	lvexp	Longview Smart Client Longview Application Framework
CalculatedJESpec	lvcje	Longview Application Framework

Document type	Recommended Extension	Applies to...
Symbol Map	lvmap	Longview Smart Client Longview Application Framework
Model	lvmod	Longview Smart Client Longview Application Framework
Procedure	lvpro	Longview Smart Client Longview Application Framework
Subroutine	lvsub	Longview Smart Client Longview Application Framework
UI	lvui	Longview Smart Client
HTML	html	Longview Smart Client

Conforming to naming conventions

For some commands and functions, you create a certain item in Longview, such as a document or symbol, and apply an appropriate name. Make sure your system can use the names you specify.

Symbol names

To create a valid symbol name, follow these guidelines:

- Symbol names can contain only letters, numbers, periods (.), and underscores (_); do not use spaces, other punctuation, or other special characters.
- Symbol names cannot contain more than 31 characters.
- Symbol names must be unique in the system.

Note: Numeric symbol names must be prefixed by the exclamation point character "!" when used in a Model or equation. For example !22342 identifies 22342 as a symbol name and not a numeric value.

Symbol descriptions

Symbol descriptions can contain a maximum of 100 characters.

Symbol hierarchies

Symbols are usually grouped together in relationships called parent, child, and leaf symbols.

Symbol Type	Description
Parent	A parent is any symbol under which other symbols are grouped. It is possible for a symbol to be both a parent symbol

Symbol Type	Description
Child	A child is any symbol grouped under another symbol. It is possible for a child symbol to itself be the parent symbol for another child symbol or symbols, as well as the child of another parent symbol.
Leaf	A leaf is any symbol with no child symbols grouped under it, regardless of how many levels deep in the hierarchy it resides.
Root	A root symbol resides in the first level of the hierarchy and does not have a parent symbol. It is possible for a root symbol to be the parent symbol of another symbol or symbols.

Symbol hierarchy is central in Longview applications to displaying the relationships between symbols and how their values relate to one another.

Rollup Indicator	Meaning
+	The child symbol's value is added to the parent symbol's value.
-	The child symbol's value is subtracted from the parent symbol's value.
0	The child symbol's value does not alter the value of the parent symbol.

User IDs

The requirements for user IDs depends on the authentication method used by your company.

User descriptions

A user description is usually the full name of the user. User descriptions can contain a maximum of 100 characters.

A user ID must follow these guidelines:

Authentication	Guidelines
Longview authentication	If your company uses Longview authentication, follow these guidelines for user IDs: ■ may include maximum of 63 characters ■ may include any number, and any letter, uppercase or lowercase ■ may contain spaces anywhere within the name, but not at the very end ■ may include the following special characters: # _ ! ~ ‘ and . ■ cannot include any other special characters, such as the backslash (\ )

Authentication	Guidelines
Windows authentication	If your company uses Windows authentication, use the user's Windows user ID, following these guidelines: ■ must include the domain name, a backslash, and the Windows user ID; for example: CONNECT \ Host Port LID [Group] ■ may include the following special characters: # _ ! ~ ‘ and . ■ cannot include any other special characters

Working with system variables

Longview provides you with several system variables to store commonly used system information. For example, you can use the ExportVariable function with system variables to export the information stored in the variable to a file. You can also use the Show Message command with system variables to display the information stored in the variable in a message window when you are creating Longview Apps.

The following system variables can be used in Procedures, Model, and Data View documents:

- Using the LVS_BATCH_IDS system variable
- Using the LVS_BLUR_ID system variable
- Using the LVS_BUTTONCLICKED system variable
- Using the LVS_ERRORCODE system variable
- Using the LVS_ERRORMESSAGE system variable
- Using the LVS_EVENT_ID system variable
- Using the LVS_FETCHINDEX system variable
- Using the LVS_FETCHSTATUS system variable
- Using the LVS_FILESCUSTOMROOTPATH system variable
- Using the LVS_FOCUS_ID system variable
- Using the LVS_HTTPPROTOCOL system variable
- Using the LVS_IDENTIFIER system variable
- Using the LVS_LANGCODE system variable
- Using the LVS_LISTDELIMITER system variable
- Using the LVS_SESSIONID system variable

- Using the LVS_USERID system variable
- Using the LVS_USERGROUP system variable
- Using the LVS_WEBBRIDGE system variable
- Using the LVS_WEBBRIDGEPATH system variable
- Using the LVS_WEBSERVER system variable

Using the LVS_BATCH_IDS system variable

This variable stores the batch IDs of the latest batches that result from any of the following commands and functions:

- Upload (for DataAreas)
- Copy Data
- Reconcile Copy
- Reconcile Upload
- JournalEntryPost()
- JournalEntryUnpost()

Note: When using LVS_BATCH_IDS in conjunction with JournalEntryPost() and JournalEntryUnpost() functions, only the last batch ID is returned.
Therefore, you should poll the LVS_BATCH_IDS value after each JournalEntryPost() and JournalEntryUnpost() action in order to retrieve the appropriate batch ID.

Syntax example:

```
Create Variable slvs_Batch_IDS[] as String
Set Variable slvs_Batch_IDS[] = STRREPLACE( "$LVS_BATCH_IDS$", "@", "|" )
```

Using the LVS_BLUR_ID system variable

This variable stores the label of the tab in a Data Grid that has lost focus after a user switches from one tab to another in a tabbed Data Grid.

Syntax example:

```
Create Variable slvs_Blur_ID as String
Set Variable slvs_Blur_ID = "The $LVS_BLUR_ID$ tab has lost focus"
Show Message "$slvs_Blur_ID$"
```

Using the LVS_BUTTONCLICKED system variable

This variable stores a user's button selection such as OK or Cancel, for example, in a dialog using the ShowFileChooser or ShowSymbolSelector function.

Syntax example:

```
Create Variable slvs_ButtonClicked as String
Set Variable slvs_ButtonClicked = $LVS_BUTTONCLICKED$
```

Using the LVS_ERRORCODE system variable

If the system encounters an error, the error code is stored in this variable.

Syntax example:

```
Create Variable nlvs_ErrorCode as Num
Set Variable nlvs_ErrorCode = $LVS_ERRORCODE$
```

Using the LVS_ERRORMESSAGE system variable

If the system encounters an error, the error message is stored in this variable.

Syntax example:

```
Create Variable slvs_ErrorMessage as String
Set Variable slvs_ErrorMessage = $LVS_ERRORMESSAGE$
```

Using the LVS_EVENT_ID system variable

This variable stores the event IDs of the current process. If the process has not started as a result of an event, the value returned is -1.

Syntax example:

```
Create Variable nlvs_Event_ID as Num
Set Variable nlvs_Event_ID = $LVS_EVENT_ID$
```

Using the LVS_FETCHINDEX system variable

Use this variable to retrieve the row index of the currently fetched row in the cursor. You could use this variable in conjunction with the SetDataTableCell function to, for example, update an existing value with a new value.

Syntax example:

```
Set Variable nResult = SetDataTableCell (myDataTable, $LVS_FETCHINDEX$,
$colIndex$,
$newSalary$)
```

Using the LVS_FETCHSTATUS system variable

Use this variable to retrieve the status of the Fetch command.

This variable is set to one of the following values, depending on the outcome of the Fetch command:

- 0 — The Fetch command was successful, and a string is stored in the relevant variable.
- -1 — The Fetch command was unsuccessful (the cursor moved past the result).
- -2 — The Fetch command was unsuccessful (the result is marked as deleted or ignored). This could occur when a row has been deleted through the user interface, but the changes have not yet been submitted.

Syntax example:

```
If $LVS_FetchStatus$ EQ 0
Run Model TransferTableData.lvmod
End If
```

Using the LVS_FILESCUSTOMROOTPATH system variable

This variable stores the path of the Files Custom Root Folder.

Syntax example:

```
Create Variable sFilesCustomRootPath as String
Set Variable sFilesCustomRootPath = "The Files Custom Root Path is $LVS_
FILESCUSTOMROOTPATH$"
Show Message “$sFilesCustomRootPath$”
```

Using the LVS_FOCUS_ID system variable

This variable stores the label of the tab in a Data Grid that has gained focus after a user switches from one tab to another in a tabbed Data Grid.

Syntax example:

```
Create Variable slvs_Focus_ID as String
Set Variable slvs_Focus_ID = "The $LVS_FOCUS_ID$ tab has gained focus"
```

```
Show Message "$slvs_Focus_ID$"
```

Using the LVS_HTTPPROTOCOL system variable

This variable stores the type of http protocol. Possible return values are HTTP and HTTPS.

Syntax example:

```
Create VARIABLE sHttpProtocol as String
Set VARIABLE sHttpProtocol = "The Http Protocol is: $LVS_HTTPPROTOCOL$"
Show Message "$sHttpProtocol$"
```

Using the LVS_IDENTIFIER system variable

This variable stores the Longview Identifier of the system.

Syntax example:

```
Create Variable slvs_Identifier as String
Set Variable slvs_Identifier = "The Identifier is: $LVS_IDENTIFIER$"
Show Message "$slvs_Identifier$"
```

Using the LVS_LANGCODE system variable

This variable stores the language code of the system. Possible return values are EN and FR.

Syntax example:

```
Create VARIABLE sLangCode as String
Set VARIABLE sLangCode = "The Language code is: $LVS_LANGCODE$"
Show Message "$sLangCode$"
```

Using the LVS_LISTDELIMITER system variable

This variable stores the delimiter for STRING and NUM list variables in the current instance, as set by the UseListDelimiter command.

Syntax example:

```
Create Variable slvs_ListDelimiter as String
Set Variable slvs_ListDelimiter = "$LVS_LISTDELIMITER$"
```

Using the LVS_RDBMS system variable

This variable stores the current database type (SQL or ORACLE).

Syntax example:

```
Create Variable slvs_RDBMS as String
Set Variable slvs_RDBMS = $LVS_RDBMS$
```

Using the LVS_SESSIONID system variable

This variable stores the current session ID.

Syntax example:

```
Create VARIABLE sSessionID as String
Set VARIABLE sSessionID = "The session ID is: $LVS_SessionID$"
Show Message "$sSessionID$"
```

Using the LVS_USERID system variable

This variable stores the username of the current user.

Syntax example:

```
Create Variable slvs_UserID as String
Set Variable slvs_UserID = "Hello $LVS_USERID$"
Show Message "$slvs_UserID$"
```

Using the LVS_USERGROUP system variable

This variable stores the user group of the current user.

Syntax example:

```
Create Variable slvs_UserGroup as String
Set Variable slvs_UserGroup = "Current user is in the $LVS_USERGROUP$ user
group"
Show Message "$slvs_UserGroup$"
```

Using the LVS_WEBBRIDGE system variable

This variable stores the path and file for the .cgi or .dll bridge file for the web bridge, such as "/cgi-bin/LID/lvweb.cgi", where LID is the Longview Identifier for the system.

Syntax example:

```
Create VARIABLE sWebBridge as String
```

```
Set VARIABLE sWebBridge = "The web bridge is: $LVS_WebBridge$"
Show Message "$sWebBridge$"
```

Using the LVS_WEBBRIDGEPATH system variable

This variable stores the path for the .cgi or .dll bridge file for the web bridge, such as "/cgi-bin/LID/", where LID is the Longview Identifier for the system.

Syntax example:

```
Create VARIABLE sWebBridgePath as String
Set VARIABLE sWebBridgePath = "The web bridge path is: $LVS_WebBridgePath$"
Show Message "$sWebBridgePath$"
```

Using the LVS_WEBSERVER system variable

This variable stores the web server, such as ca127dev12.

Syntax example

```
Create VARIABLE sWebServer as String
Set VARIABLE sWebServer = "The web server is: $LVS_WebServer$"
Show Message "$sWebServer$"
```

Working with system variables for persistent event rules

Longview provides you with the LVS_EVENTINITIALIZE, LVS_EVENTSHUTDOWN, and LVS_PERSISTENT system variables, which are used in conjunction with persistent event rules. A single Management Server process is launched to handle persistent events in the following scenarios:

- when the server is started
- if maintenance is turned off
- if application framework is disconnected and a persistent event rule is triggered

The Management Server process remains in the system until the data server is shut down, in order to handle each successive triggering of the persistent event. For more information on persistent event rules, see the *Longview Application Administrator Guide*.

> **Note:** If you make any changes to the list of persistent event rules (in either the Persistent Event Rule Setup dialog, or the rulepersists.txt file) or the name of the procedure launched by the persistent event rules, you must restart the servers for your changes to take effect.

Using the LVS_EVENTINITIALIZE system variable

This variable indicates if the current process is run as a result of the Management Server starting, or not. While the servers are starting, the corresponding event procedure is run.

You can use this variable within the event procedure to define what to run upon initialization. This variable should be used to ensure that the actual calculation is not run when the event procedure is run as a result of the Management Server starting.

For instance, (in the Persistence syntax example) if the servers are starting, the `LVS_EVENTINITIALIZE` variable indicates that the **create persistent lock** procedure should be run.

Using the LVS_EVENTSHUTDOWN system variable

This variable indicates if the current process is run as a result of the Management Server stopping, or not. While the servers are stopping, the corresponding event procedure is run.

You can use this variable within the event procedure to define what to run upon shutdown. This variable should be used to ensure that the actual calculation is not run when the event procedure is run as a result of the Management Server stop- ping.

For instance (in the Persistence syntax example), if the servers are stopping, the `LVS_EVENTSHUTDOWN` variable indicates that the **delete persistent lock** procedure should be run.

Using the LVS_PERSISTENT system variable

This variable indicates if the current process is persistent or not. When an event is triggered, the corresponding event procedure is run. You can use this variable within the event procedure to define what to run for a persistent event. This variable should be used to ensure that commands that connect to and disconnect from Application Framework are not run when the event is a persistent event.

For instance (in the Persistence syntax example), in cases where the current process is not persistent, the `LVS_PERSISTENT` variable indicates that locks for event rule processes that are not persistent should be created and deleted.

Persistence syntax example

Syntax:

```
If $LVS_EventInitialize$ AND $LVS_Persistent$
// create persistent lock
Create LOCK USER using "bigdataspec.lvdsp"
Else
If $LVS_EventShutDown$ AND $LVS_Persistent$
// delete persistent lock
Delete LOCK USER using "bigdataspec.lvdsp"
Else
If Not $LVS_Persistent$
// create lock if persistence not used
```

```
Create LOCK USER using "smalldataspec.lvdsp"
END If
// create and download data area
Create DATAAREA da using "smalldataspec.lvdsp" Download da
// run model
Run MODEL "calc.lvmod" on da
// upload data area Upload da
If Not $LVS_Persistent$
// delete lock if persistence not used
Delete LOCK USER using "smalldataspec.lvdsp"
END If
END If
END If
```

Working with context variables

Longview provides you with several context variables populated with App context- related information. When a user right-clicks a cell or clicks a toolbar button in a Data Grid or a Table, an action is triggered, and specific context variables are populated.

The following context variables can be used in Longview Application Framework documents:

Variable	Returns…	Example
C_CONTEXT (for Data Grids)	a user's click selection in a Data Grid. If the user clicks a cell, the return value is CELL. If the user clicks a toolbar action, the return value is DATAGRID.	$C_CONTEXT$ resolves to CELL or $C_CONTEXT$ resolves to DATAGRID

Variable	Returns...	Example
C_CONTEXT (for Tables)	■ If the user right-clicks on the row header, the return value is ROW. ■ If the user clicks a toolbar action, the return value is TABLE. ■ If the user manually edits data in a cell or edits data using a UI control, the return value is DYNAMICCELLEDIT. ■ If the user invokes the standard row operation Add, the return value is DYNAMICADD. ■ If the user invokes the standard row operation Duplicate, the return value is DYNAMICDUPLICATE. ■ If the user invokes the standard row operation Delete, the return value is DYNAMICDELETE. ■ If the user invokes the standard row operation Edit, the return value is DYNAMICEDIT. ■ If the user invokes the standard row operation Reset, the return value is DYNAMICRESET. ■ If the user invokes the standard row operation Reset All, the return value is DYNAMICRESETALL. ■ If the user pastes multiple values into a table, when pasting the first value and subsequent values (except the last value), the return value is BEGINMULTI-PASTE. ■ If the user pastes multiple values into a table, when pasting the last value, the return value is ENDMULTIPASTE.	$C_CONTEXT$ resolves to ROW or TABLE
C_ DATAAREA	the name of the related data area.	$C_DATAAREA$ resolves to SampleDA
C_ DATATABLE (for Tables)	the name of the Data Table object.	$C_DATATABLE$ resolves to SampleDT
C_ CELLVALUE (for Data Grids)	the value of the intersection after a user right-clicks a cell in a Data Grid. This variable is not populated for toolbar actions.	$C_CELLVALUE$ resolves to 200

Variable	Returns...	Example
C_ CELLVALUE (for Tables)	the value of the intersection of an edited cell in a Table. This variable is not populated for toolbar actions or row context actions.	$C_CELLVALUE$ resolves to 200
C_ CELLVALUE PRIOR (for Tables)	the previous value of the intersection after a user has edited a cell in a Data Table. This variable is not populated for toolbar actions or row context actions.	$C_ CELLVALUEPRIOR $ resolves to 100
C_ ROWINDEX	the number of the corresponding row index in the Data Table object in which the user has right-clicked. If C_CONTEXT resolves to TABLE, this variable resolves to 0.	$C_ROWINDEX$ resolves to 2
C_ROWS	the dimension in the rows orientation of the Data Grid. If the Data Grid has nested dimensions in the rows orientation, the variable returns the dimensions in order from outer to inner.	C_ROWS resolves to ENTITIES, ACCOUNTS
C_SOURCE ROWINDEX	the number of the corresponding row index in the Data Table Object that the user has duplicated.	$C_ SOURCEROWINDEX$ resolves to 1 when the first row in the table is duplicated
C_ COLUMNS	the dimension in the column orientation of the Data Grid. If the Data Grid has nested dimensions in the columns orientation, the variable returns the dimensions in order from outer to inner.	$C_COLUMNS$ resolves to TIMEPER
C_ DIMENSION	the name of the symbol for each dimension at the intersection that the user right-clicked. For toolbar actions, these variables are populated only for dimensions that are not in the rows or columns orientation. The names of the variables in Longview Application Framework to be used in these context variables are limited to 31 characters. The convention is C_DIMENSION (where DIMENSION is the first 26 characters of the dimension name). In the case where the first 26 characters of the dimension names are the same, the variables are named C_ DIMENSION_N (where DIMENSION is the first 26 characters of the dimension, and N is the dimension number).	$C_CURRENCIES$ resolves to CAD or $C_ ENTITIES$ resolves to TORONTO

Working with global variables

Longview provides you with several global variables to use when creating Longview Apps. Unlike system variables, App Developers can set the value of global variables. Longview Application Framework or Longview Apps can use the values to which these global variables are set to trigger an action or behavior such as displaying a visual warning indicator.

The following global variables can be used in Longview Application Framework documents:

- LVG_FormValid
- LVG_FormMessage
- LVG_TableRedraw

Using the LVG_FormValid global variable

This variable returns custom validation logic. For each control in a Longview Form, you can optionally specify an OnChange procedure to run each time an input value changes. If LVG_FormValid is set to 0, the input item that triggered the OnChange procedure is considered invalid. A visual indicator appears to the user to indicate the invalid control. You can optionally specify a customizable validation message to display using the LVG_FormMessage.

For more information on OnChange, see Using the OnChange Keyword.

Syntax example:

```
Show Message "Checking your email address..." continue
If StrContains ("$sEmail$", "@")
Set Variable LVG_FormValid = 1
Else
Set Variable LVG_FormValid = 0
Set Variable LVG_FormMessage = "Your email address must include the @ sign."
End if
```

Using the LVG_FormMessage global variable

This variable returns a customizable message to a user after a form is validated. You can use this variable in conjunction with the LVG_FormValid global variable to display a message to a user when an input item is considered invalid.

Syntax example:

```
Set Variable LVG_FormMessage = "Your email address must include the @ sign."
```

Using the LVG_TableRedraw global variable

This variable returns custom redraw logic for a Table. After a dynamic procedure is run on a Table, there may be instances where the row needs to be redrawn or the entire table needs to be redrawn to reflect changed values. There are also instances where a redraw is not needed.

You can optionally specify the redraw logic by setting the `LVG_TableRedraw` global variable to one of the following:

Variable	Description
NONE	To indicate that no part of the table should be redrawn.
ROW	To indicate the current row should be redrawn.
TABLE	To indicates the entire table should be redrawn.

Note: The default value for LVG_TableRedraw is Row.

For more information on Dynamic Procedures, see Exclusive.

Syntax example:

```
If ("$C_CONTEXT$" == "DYNAMICCELLEDIT")
RUN PROCEDURE "RecalcInventory.lvpro"
Set VARIABLE LVG_TableReDraw = "TABLE"
Else
Set VARIABLE LVG_TableReDraw = "NONE"
End If
```

Working with event variables

Longview provides you with several event variables populated with event-related information. When an event is triggered, the event variables are populated with the corresponding symbols that triggered the event. There is one event variable for each dimension in the system.

For example, $E_ACCOUNTS$ resolves to "AnnualSalary" or $E_TIMEPERIODS$ resolves to "April."

Note: Event variables are supported for base dimensions only; event variables are not supported for schedule dimensions.

The names of the variables in Longview Application Framework to be used in these event variables are limited to 31 characters. The convention is E_Dimension (where Dimension is the first 26 characters of the dimension name).

In the case where the first 26 characters of the dimension names are the same, the variables are named E_Dimension_N (where Dimension is the first 26 characters of the dimension, and N is the dimension number).

Using the LVS_EVENTERRORFILES variable

This variable stores the location of where error files are generated for events, as specified by the data server configuration parameter Events Error Files Location. You may use this variable in an event procedure. If this parameter has not been defined in the data server configuration, the value will be an empty string.

Syntax example:

```
CREATE Variable slvs_EventErrors as String
Set Variable slvs_EventErrors = $LVS_EVENTERRORFILES$
```

Using the LVS_EVENTHISTORYFILES variable

This variable stores the location of where history files are generated for events, as specified by the data server configuration parameter Events History Files Location. You may use this variable in an event procedure. If this parameter has not been defined in the data server configuration, the value will be an empty string.

Syntax example:

```
CREATE Variable slvs_EventHistory as String
Set Variable slvs_EventHistory = $LVS_EVENTHISTORYFILES$
```

Using the LVS_EVENTLOGFILES variable

This variable stores the location of where log files are generated for events, as specified by the data server configuration parameter Event Log Files Location. You may use this variable in an event procedure. If this parameter has not been defined in the data server configuration, the value will be an empty string.

Syntax example:

```
CREATE Variable slvs_EventLogs as String
Set Variable slvs_EventLogs = $LVS_EVENTLOGFILES$
```

Working with symbol specifications

Symbol specifications are used in numerous instances in retrieving information about symbols and the data they contain. In situations where you are querying data, the following specifications are possible:

Specification	Meaning
symbol###	all leaf symbols under symbol, unless the symbol is a leaf then it will just return the specified symbol
symbol#n	all symbols under symbol, n levels down, including symbol
symbol##n	all parent symbols under symbol, n levels down
symbol##+n	all parent symbols under symbol, n levels down, including symbol
symbol#*	all roots of symbol
symbol#-n	all symbols exactly n levels down from symbol
symbol##^n	all ancestors of symbol that are n levels above symbol, including symbol
symbol##^n:root	all symbols above symbol, n levels up, under root, and including symbol, where root is the root symbol (e.g., BALSHEET##^2:TRIALBAL)

Data Grid functions that support symbol specifications include:

- ConditionalCellStyle
- Filter
- Include
- LIDInputOnly
- NumericInputOnly
- Protect
- SymbolWidth
- TextInputOnly

Data Table functions that support symbol specifications include:

- SymbolFilter

Using conditional operators

Conditional operators enable you to specify the desired relationship between values in the database or DataAreas and to create instructions based on whether or not the conditions they define exist. They are crucial in processes involving the use of the If commands or functions.

Valid conditional operators in Longview Application Framework include the following:

Symbol	Meaning
NE or !=	is not equal to
EQ or ==	is exactly equal to
GT or >	is greater than
GE or >=	is greater than or equal to
LT or <	is less than
LE or <=	is less than or equal to

Setting values in Longview Application Framework

When you set values in Longview Application Framework commands, functions, etc., double quotation marks enclosing the values may be recommended or mandatory. Follow these rules:

Value	Double Quotation Marks?	Example
Empty, or null value	Required	""

Value	Double Quotation Marks?	Example
Space(s)	Required	""
Space anywhere in the value (leading, embedded, trailing)	Required	"John Smith"
No spaces	Optional	3681559 or "3681559"

Using tokens, variables, and strings

You can use tokens, variables, and strings to customize certain aspects of your code. Tokens are resolved and variables are evaluated dynamically at the time of execution. As a Developer, you can insert tokens into code, usually in place of strings or text.

Attribute tokens

You can use a token to retrieve attribute values, such as the description for a user or symbol, or the user's email address. Use a "[[ ]]" pair to identify the presence of a token.

Syntax:

```
[[AttrClass,AttrName,ObjectName]]
```

where:

- AttrClass is the attribute class. Type one of the following:

Value	Description
SYSTEM	Use this class for system attributes.
SYMBOL	Use this class for symbol attributes.
USER	Use this class for user attributes.

- AttrName is the attribute name. Type one of the following:

Value	Description
DESCRIPTION	For symbol and user attributes, use this attribute name to retrieve the symbol or user's description.
EMAIL	For user attributes, use this attribute name to retrieve the user's email address.

- ObjectName is the attribute object. Type one of the following:

Attribute Type	Use...
SYSTEM	DBDefault
SYMBOL	The symbol name
USER	The user ID or THIS for the current user

Syntax example:

```
[[SYMBOL,DESCRIPTION,A40000]]
[[SYSTEM,ASCalendarYr,DBDefault]]
[[USER,EMAIL,BSummers]]
```

Resource Tokens

You can use a token to retrieve resource strings from a resource bundle residing on the Longview Data Server. The resource bundle must be loaded into memory before using this token.

Use a "[[ ]]" pair to identify the presence of a token. You may find these tokens useful when creating Longview Apps that need to be supported for multiple languages. You may use this token in various documents such as: Longview Forms, Longview Table Views, Longview Data Views, Longview Procedures, and Longview DataAreaSpecs.

The language and culture of the operating system (OS), along with the name of the resource bundle determines how the resource key lookup is performed.

The names of resource bundles can be one of the following formats:

Order of Lookup	Name of resource	Example
1	resourceBundle.Language-Culture.extension	Strings.en-CA.lvres
2	resourceBundle.Language.extension	Strings.en.lvres
3	resourceBundle.extension	Strings.lvres

For example, for an OS that is en-CA (English language, Canada culture), the look up will occur in the following files in the following order:

- Strings.en-CA.lvres
- Strings.en.lvres
- Strings.lvres

By organizing resources properly, you can develop Longview Apps that will support multiple languages.

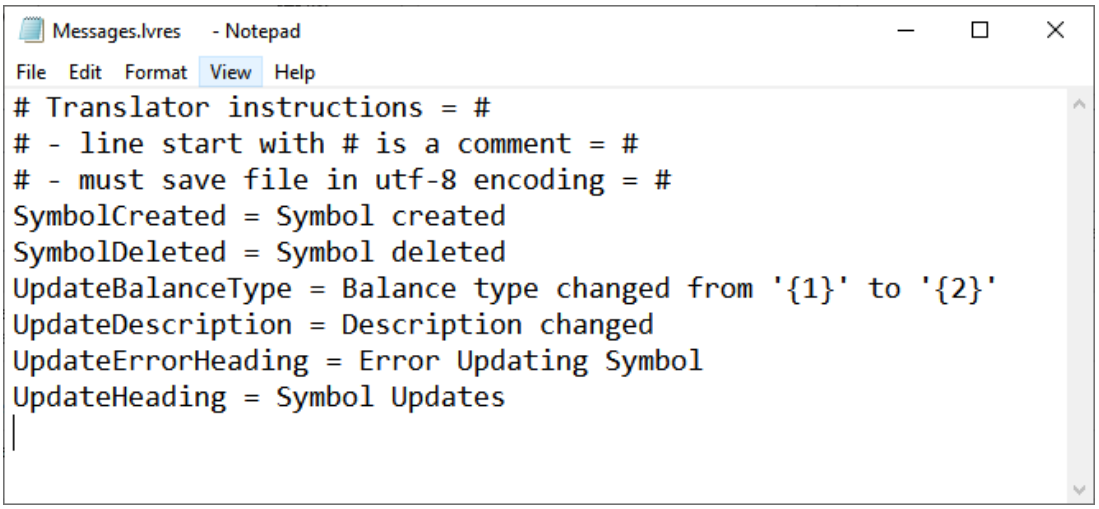
Syntax example:

```
[[RESOURCE, "resourceBundle", "resourceKey"]]
```

where:

- resourceBundle is the name of the resource bundle.
- resourceKey is the name of the resource key.

Sample format of resource file



Syntax Example 1

```
// Using a resource token in a Button in a Longview Form

Button "Update", "Text:[[RESOURCE, "Messages", "UpdateHeading"]] ",
Validate:TRUE, "OnClick:button.lvpro"
```

Syntax Example 2

```
// Using a resource token (with variables) in a Toolbar Action in a Longview
Data View

ACTION "[[RESOURCE, "$source$", "$resourceKey$]]" ", "Image.png", "RUN
PROCEDURE MyProc.lvpro", "This is help text"
```

Syntax Example 3

```
// Using a resource token (with variables) in a SHOW MESSAGE command

SHOW MESSAGE "[[RESOURCE, "$source$", "$resourceKey$"]] "
```

Variables

A variable is a placeholder for dynamic data. There are different types of variables to support the storage of a single string value or list of string values, a single numeric value or list of numeric values, a list of symbol names, and a single object instance or list of object instances. You must first create your

variable using the Create Variable command, assign a value to your variable using the Set Variable command, and then enclose your variable in $ $ to retrieve the value.

For object type variables, you can also assign values to its properties by using the Create Property command, followed by the Set Property command.

For more information on creating variables, see Create Variable.

For information on creating variables that you can use within any model or procedure called from your main procedure, see Create GlobalVariable.

In Longview Apps, you can pass variables between Longview Procedure and HTML documents by prefixing your variable with “LV_” in your HTML document.

For more information, see Passing variables between Longview Apps and Longview Application Framework.

Syntax Example 1: For Your Longview Apps Procedure File

```
Create Variable EmpName as String Set Variable EmpName = “John Doe”
Show Message “Welcome $EmpName$!”
```

Syntax Example 2: For Your Longview Apps HTML File

```
<input id=“LV_EmpName” name=“EmpName” type=“text” /input>
```

Strings

Strings can be joined together through a process called concatenation. Concatenation is useful for joining two individual values to create combined values, or two-character strings to create a symbol name.

For example, suppose a complete account number in your system appears in the format A11234 — consisting of a prefix code (A), followed by two digits (11) representing an entity code, followed by three digits (234) representing an account code.

You can combine any two of the elements by concatenating them. The concatenation provided by your system depends on the format of the source items:

Format of Sources	Value of TargetSymName
If either of the sources contains a numeric value	Numeric source not used in concatenation.
If both sources contain numeric values	Resulting string remains unchanged.

Syntax:

```
STRING 1 + STRING 2
```

Syntax example:

```
CREATE VARIABLE StrTemp1 AS STRING
SET VARIABLE StrTemp1 = "aaaaa" + "bbbbb"
StrTemp1 will equal "aaaaabbbbb" (without the double quotation marks)
```

Developing Longview Procedures

This section contains information on these main topics:

- Creating Procedure documents
- Using Procedure commands
- Using Procedure functions

Creating Procedure Documents

To perform tasks in Longview Application Framework, you can use a Procedure document.

A Procedure document defines processes in Longview Application Framework. This document is stored as an ASCII file and can be edited in any text editor. It contains a sequence of commands to perform activities, and comments. Those commands are described in Using Procedure Commands.

```
// Procedure Document - to run commands and some functions

create variable Date1 AS STRING
set variable Date1 = GetDate()
History on "History_$Date1$.txt"
//History on "History_$TestCase1$.txt"

create variable TestCase1 AS STRING
create variable DataArea1 AS STRING
set variable TestCase1 = "AFCOOxxx"
set variable DataArea1 = "AFCOOxxx"


create dataarea data1 using $DataArea1$.da
create lock user using $DataArea1$.da "Testing Functions"
download data1

run model $TestCase1$.mod on data1
upload data1
run bridge $TestCase1$.bri

delete lock all using $DataArea1$.da
delete dataarea data1
disconnect
```

A Procedure document can also be used to call other Procedure documents.

One script can be run from within another script by means of issuing a RUN PROCEDURE command, using the following syntax:

```
RUN PROCEDURE proc
```

where:

- proc is the name of the script to be run.

Note: To add a comment to syntax, insert two forward slashes (//) at the beginning of the comment.

Caution: Variables to be used in a procedure triggered by a server event rule are defined by the event rule itself. Therefore, you do not need to create them or define them in the Procedure document. For more information, see the *Longview Application Administrator Guide*.

Using Procedure Commands

This topic contains the Longview Application Framework commands you can use in a procedure.

Click the links in the following table for detailed usage and syntax information:

Activate User	Get AppInfo	RestAPI SetResponseHeader
Append Attribute	Get Category	Rest DeleteRequestHeader
Assign App	Get Data	Rest ExecuteDelete
Assign DataRole	Get DataAuditTrail	Rest ExecuteGet
Assign Group	v26.2 Get EffectiveAuthorization	Rest ExecuteHead
Assign ProcessMap	Get JEHeaders	Rest ExecutePost
Assign Symbol	Get JEDetails	Rest ExecutePut
Assign User	Get Log	Rest GetResponseHeader
AuditTrail	Get Mappings	Rest GetResponseStatus
AutoLoad	Get Maps	Rest SetRequestEncoded
	Get MetadataAuditTrail	Rest SetRequestHeader
BatchComment	Get ProcessMapInfo	Run (for ExportSpecs)
BatchWait	Get Template	Run (for ImportSpecs)
Begin AdminService	Get TemplateInfo	Run (for Models)
CancelClose	Get WebObject (image or icon)	Run (for Procedures)
Clear DataArea	Get WebObject (page or panel)	Save
Close Cursor	Get WebObjectInfo	Send Email
	Get Workflow	ServerMath
Collect Statistics	Get WorkflowLog	Set Attribute
Connect	History	Set Category
Copy Data	If...	Set DataArea
Copy File	ImportObject	Set DataRole
Create Attribute	Insert DataTableRow	Set DataTable
Create Category	Launch App	
Create DataArea	Launch Component	
Create DataRole	Launch File	Set Email
Create DataTable	Launch ProcessMap	Set FTP
Create Directory	Launch ReportViewer	Set Group
Create Document	Launch SystemApp	Set Property
Create GlobalVariable	Launch URL	Set Rule
Create Group	Load	Set Schedule
Create Lock	LoadKar	Set Symbol
Create Property	LVMGRL	Set User

Create Rule	Maintenance	Set Variable
Create Schedule	Move File	Set WorkflowStatus
Create Symbol	OnClose	Show DataArea
Create User	OnError	Show DataReport
Create Variable	Open Cursor	Show DataTable
DataReport	OpSys	Show Error
Deactivate User	Profile	Show Form
Delete Attribute	Publish App	Show HTML
Delete Category	Publish ProcessMap	Show Message
Delete Data	Publish Template	Show Return
Delete DataArea	Pushdown	Show UI
Delete DataRole	Put Category	Sleep
Delete DataTable	Put Mappings	Sort DataTable
Delete DataTableRows	Put Maps	Stream
Delete File	Put WebObject (image or icon)	Switch Symbol
Delete Group	Put WebObject (page or panel)	SyncHier
Delete Lock	Put WebObjectInfo	TaxDTNRRec
Delete Rule	Put Workflow	TaxFX
Delete Schedule	QuickFormat	ThrowError
Delete Statistics	Reconcile Copy	Timer
Delete Symbol	Reconcile Upload	Unload
Delete User	Refresh Attribute	Unpublish App
Disconnect	Refresh Dimension	Unpublish ProcessMap
Download (for DataAreas)	Remove App	Unpublish Template
Download (for DataTables)	Remove DataRole	Update DataTableRows
End AdminService	Remove Group	Upload (for DataAreas)
Exclusive	Remove ProcessMap	Upload (for DataTables)
ExportCommands	Remove Symbol	UseInputRelatedSchedules
ExportHierarchy	Remove Template	UseListDelimiter
ExportProperty	Remove User	While... End While
ExportVariable	Rename File	Write
Fetch	Reset MetadataAuditTrail	
For Each	Resolve	
FTP Put	Resource Reset	
FTP Put	RestAPI GetRequestHeader	

Activate User

Use this command to activate a user that is currently deactivated.

Syntax (regular)

```
ACTIVATE USER [DomainName\]UserId
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the name of the user to which you want to activate.

Syntax example:

```
ACTIVATE USER JSmith
```

Creating an ASCII input file

To activate multiple users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId
```

Syntax (with An ASCII File)

```
ACTIVATE USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
ACTIVATE USER @users.txt
MAINTENANCE OFF
```

See also

- Deactivate User

Append Attribute

Use this command to append values to a LIST type Attribute.

An attribute is data used to describe the characteristics of an object in Longview. For example, SGPCOMPANYNAME is an Attribute representing the name of your company, and its value is your company name.

This command is useful when you want to append values to an Attribute that is set to a list of values.

Syntax (regular) Longview

```
APPEND ATTRIBUTE AttrClass AttrName VALUE AttrObject AttrValue
```

where:

- AttrClass is the Attribute class. Select one of the following:

Value	Description
SYSTEM	Describes the entire system at the highest level. Attributes of this Attribute class specify system-wide characteristics. There is only one object in the SYSTEM Attribute class called DBDEFAULT which represents the application itself.
USER	Describes the Attributes of a particular user. Each user is an object in the USER Attribute class.
SYMBOL	Describes the characteristics of individual symbols. Each symbol is an object in the SYMBOL Attribute class.

- AttrName is the Attribute to which you want to append Attribute values.
- AttrObject is the Attribute object. In the SYSTEM Attribute class, there is only one object — DBDEFAULT.
- AttrValue is the data you want to append to the Attribute. Separate multiple values with a pipe (|).

Syntax example:

```
APPEND ATTRIBUTE SYSTEM SGPFloatingTimePeriods VALUE DBDEFAULT
ISRAB|ISRPFY|ISPNHR
```

Creating an ASCII input file

To append data for a large number of Attributes, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

Syntax

```
AttrClass{AttrName{VALUE{AttrObject{AttrValue
```

Syntax (with an ASCII file):

```
APPEND ATTRIBUTE @Filename
```

where:

- If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
APPEND ATTRIBUTE @sysatr.asc
MAINTENANCE OFF
```

Assign App

Use this command to assign a Longview App to an existing category. You must first publish the Longview App before you can assign it to categories. Use separate lines to assign a Longview App to multiple categories. For information on publishing an app, see Publish App. For information on creating categories, see Put Category.

Syntax:

```
ASSIGN APP AppName CATEGORY Category
```

where:

- AppName is the name of the Longview App to be assigned to the category. If the Longview App name includes spaces, enclose it in double quotation marks. You don't have to include the .lvapp extension in the name. Longview App names cannot exceed 100 characters.
- Category is the name of the category to which the Longview App is assigned. If the category name includes spaces, enclose it in double quotation marks.

Syntax example:

```
ASSIGN APP SampleApp CATEGORY Administrators
```

Syntax (with an ASCII file):

```
ASSIGN APP @FileName
```

where:

- FileName is an existing ASCII file containing the categories to which the Longview App is assigned, with the following syntax:

  ```
  AppName{CATEGORY{Category
  ```

- If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

  Syntax example

  ```
  ASSIGN APP @"Sample App Categories.txt"
  ```

See also

- Publish App
- Put Category

Assign DataRole

Use this command to specify the database privileges of a DataRole established by CREATE DATAROLE.

Syntax (regular)

```
ASSIGN DATAROLE RoleName ACCESS DimName RESTRICTED SymName AccessType
[NumLevels] [Priority]

ASSIGN DATAROLE RoleName ACCESS DimName FIXED SymName AccessType [Priority]

ASSIGN DATAROLE RoleName ACCESS DimName FULL [Priority]
```

where:

- RoleName is the name of a symbol access role.
- DimName is the dimension to which the DataRole is to be given access.
- RESTRICTED means that access is defined as a subset of the dimension spanning more than one level.
- SymName is the top symbol in the hierarchy to which you want to give the role access.
- AccessType is a setting that determines whether the role's users have read/write access. Select one of the following:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- NumLevels is a number representing the number of hierarchy levels below SymName.
- FIXED means that access is constrained to a single symbol in the dimension.
- Priority is a numeric value between 0 and 99 (0 representing the lowest priority) representing the position of the symbol with respect to its parent.
- FULL enables write access to all symbols in the dimension.

Creating an ASCII input file

To assign a large number of DataRoles, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
RoleName{ACCESS{DimName{RESTRICTED{SymName{AccessType[{NumLevels]
[{Priority]
RoleName{ACCESS{DimName{FIXED{SymName{AccessType[{Priority]
RoleName{ACCESS{DimName{FULL[{Priority]
```

Syntax (with an ASCII file):

```
ASSIGN DATAROLE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Data.txt"

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax:

```
MAINTENANCE ON
ASSIGN DATAROLE @datarole.txt
MAINTENANCE OFF
```

See also

- Create DataRole

Assign Group

Use this command to assign symbol access, authorizations, and users to a user group.

You can assign access to a total of 767 separate Longview symbols to a user. For example, suppose User1 belongs to Group1. Group1 has access settings to 225 symbols. User1 has access to an additional and separate 542 symbols not listed in the group he belongs to. In total, User1 has access to 767 symbols. Symbol access roles can also been assigned to users and user groups.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Note: v26.2 For systems on the ISW Platform, when a user belongs to a group with authorizations configured, the user's individual authorizations are ignored and only the group authorizations apply. If the group has no authorizations configured, the user's individual authorizations are used instead. Any group authorizations that exceed what the user's license permits are also ignored.

Syntax (regular)

```
ASSIGN GROUP GroupName ACCESS Dimension RoleName Inherit SymbolName
AccessType NumLevels [Priority]

ASSIGN GROUP GroupName AUTHORIZATION AuthorizationName [AuthorizationGroup]

ASSIGN GROUP GroupName USER [DomainName\]UserId
```

where:

- GroupName is the name of the group to which you want to assign access, authorizations, or users.
- Dimension is a dimension containing the symbols.
- RoleName is the name of a symbol access role.
- Inherit causes access to be inherited from the role. Valid values are TRUE (inheriting access from the role) and FALSE (setting specific access). If Inherit is TRUE, do not include SymbolName, AccessType, NumLevels, and Priority.
- SymbolName is the top symbol in the hierarchy to which you want to give the group access. It must be a subset of the role. If Inherit is TRUE, do not include this parameter.
- AccessType is a setting that determines whether the user has read/write access. If Inherit is TRUE, do not include this parameter. Select one of the following:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- NumLevels is a number representing the number of hierarchy levels below SymbolName. If Inherit is TRUE, do not include this parameter.
- Priority is optional and is a number that designates a symbol's position in the hierarchy relative to its parent. Symbols are listed in order of ascending priority, with zeros falling at the bottom of the list. If Inherit is TRUE, do not include this parameter.
- AuthorizationName is a name that designates the type of authorization being granted to the user group and can be the following:

Value	Description	Platform License
ADDINFOROFFICE	Access the Longview Add-In for Office. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Longview Add-In for Office.	Designer Power User Viewer
ADDINFOROFFICE SUBMITDATA	Submit data in the Longview Add-In for Office.	Designer Power User
ANALYSISREPORTING AUTHOR	Access Longview Analysis and Reporting as a Report Author.	Power user
ANALYSISREPORTING PUBLISHER	Access Longview Analysis and Reporting as a Report Publisher.	Designer
ANALYSISREPORTING USER	Access Longview Analysis and Reporting as a Report User.	Viewer
APPLICATION ADMINISTRATOR	Access Longview Application Administrator.	Admin Designer Service
ATTRIBUTE	Manage attributes.	Designer
BATCH	Manage batches.	Admin Designer

Value	Description	Platform License
CONNECTVIA APPLICATION FRAMEWORK	Connect to the server using Longview Application Framework. **Note:** You must assign this authorization to allow users or user groups to access Longview Apps, Longview Designer, Longview tools and editors, the Longview Add-In for Office, and Longview Tax.	Admin Designer Power User Viewer Service
DASHBOARDDESIGNER	Access Longview Dashboard Designer.	Designer
DELETECOMMENTS	Delete any existing comments in the Data Server. Users without Delete Comments authorization can delete only their own comments before they are submitted to the database. **Note:** Delete existing comments functionality is available in Data Grids only.	Designer
DESIGNER	Access Longview Designer. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSCREATE	Create data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSDELETE	Delete data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer

Value	Description	Platform License
DESIGNERDATA IMPORTSMODIFY	Edit data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSPUBLISH	Publish data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERLONGVIEW APPSPUBLISH	Publish Longview Apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
FOREIGNEXCHANGE SETTINGS	Manage foreign exchange settings.	Admin Designer
GROUPADMINISTRATION	Perform group administration.	Does not apply
GROUPSYMBOLACCESS	Manage symbol access for groups.	Does not apply
INTERCOMPANYSETTINGS	Manage intercompany settings.	Designer
JOURNALENTRIES	Access Longview Journal Entries.	Designer Power User
JOURNALENTRY	Manage journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYCURRENTPERIODCREATE	Create current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODPERMPOST	Permanently post current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODREVIEWPOST	Review post current period journal entries.	Designer Power User Service
JOURNALENTRYDELETE	Delete non-shared journal entries.	Designer Power User Service
JOURNALENTRYFUTUREPERIODCREATE	Create future period journal entries.	Designer Power User Service
JOURNALENTRYFUTUREPERIODPERMPOST	Permanently post future period journal entries.	Designer Power User Service
JOURNALENTRYFUTUREPERIODREVIEWPOST	Review post future period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYOWNPERMPOST	Permanently post own journal entries.	Designer Power User Service
JOURNALENTRYOWNREVIEWPOST	Review post own journal entries.	Designer Power User Service
JOURNALENTRYPRIORPERIODADJCREATE	Create prior period journal entries.	Designer Power User Service
JOURNALENTRYPRIORPERIODADJPERMPOST	Permanently post prior period journal entries.	Designer Power User Service
JOURNALENTRYPRIORPERIODADJREVIEWPOST	Review post prior period journal entries.	Designer Power User Service
JOURNALENTRYRESTATEMENTCREATE	Create restatement journal entries.	Designer Power User Service
JOURNALENTRYRESTATEMENTPERMPOST	Permanently post restatement journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYRESTATEMENTREVIEWPOST	Review post restatement journal entries.	Designer Power User Service
LOCK	Manage locks.	Designer Service
MAPPINGSEDITOR	Access the Mappings editor. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MAPPINGSMANAGE	Create, modify, delete mappings. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MAPSMANAGE	Create, modify, delete maps. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MODIFYDATA	Submit data.	Designer Power User Service
NDDSETTINGS	Manage NDD settings.	Designer
ROLE	Manage symbol access roles.	Designer Service

Value	Description	Platform License
RULE	Manage rules.	Designer Service
SCHEDULE	Manage schedules.	Designer Service
SERVERMANAGER	Access Longview Server Manager.	Admin Service Designer
SERVERMANAGERSTART	Start/stop the Longview server.	Admin Service Designer
SERVICEACCOUNTUSER ADMINISTRATOR	Sets the user to be a User Administrator.	Service
SERVICEACCOUNTRESTAPI	Sets user to be a Service Account User. **Note:** Can only be set using PUSER or a user who has User Administrator Authorization.	Admin Designer Service
SYMBOL	Manage symbols.	Designer Service
SYMBOLASSIGN	Assign symbols to a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLATTRIBUTECREATE	Create symbol attributes. Must be used with ATTRIBUTE.	Designer Service

Value	Description	Platform License
SYMBOLATTRIBUTEDELETE	Delete symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEMODIFY	Modify symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLCANCREATEROOT	Create root symbols. Must be used with SYMBOL, CREATESYMBOL.	Designer Service
SYMBOLCANDELETEROOT	Delete root symbols. Must be used with SYMBOL, SYMBOLDELETE.	Designer Service
SYMBOLCREATE	Create symbols. Must be used with SYMBOL.	Designer Service
SYMBOLDELETE	Delete symbols. Must be used with SYMBOL.	Designer Service
SYMBOLREMOVE	Remove symbols from a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLSET	Modify symbols. Must be used with SYMBOL.	Designer Service
SYMBOLSWITCH	Switch symbols in a hierarchy. Must be used with SYMBOL.	Designer Service

Value	Description	Platform License
SYSTEMATTRIBUTECREATE	Create system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEDELETE	Delete system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEMODIFY	Modify system attributes. Must be used with ATTRIBUTE.	Designer Service
USERADMINISTRATION	Perform user administration	Does not apply
USERATTRIBUTECREATE	Create user attributes. Must be used with ATTRIBUTE.	Designer Service
USERATTRIBUTEDELETE	Delete user attributes. Must be used with ATTRIBUTE.	Designer Service
USERATTRIBUTEMODIFY	Modify user attributes. Must be used with ATTRIBUTE.	Designer Service
USERRESETPASSWORD	Reset passwords.	Does not apply
USERSYMBOLACCESS	Manage symbol access for users.	Does not apply
VIEWDATA	View data.	Designer Power User Viewer Service
WORKFLOWDESIGNER	Access Longview Workflow Designer.	Designer

- AuthorizationGroup is required only when AuthorizationName is USERADMINISTRATION or USERRESETPASSWORD, and is the group for which you want the indicated group to have authorization.

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the ID of a user.

Syntax example:

```
MAINTENANCE ON
ASSIGN GROUP Administrator ACCESS ACCOUNTS ANALYSIS_REPORTING_ACCESS FALSE
TrialBalance W 99 2
MAINTENANCE OFF
```

Creating an ASCII input file

To change the access privileges of a large number of users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
GroupName{ACCESS{Dimension{RoleName{Inherit{SymName{Access{NumLevels
[Priority]
GroupName{AUTHORIZATION{AuthorizationName
GroupName{USER{[DomainName\]UserId
```

Note: Include SymName, Access, NumLevels, and Priority only if Inherit is set to FALSE.

Use the QUICKFORMAT command, if necessary, to make formatting changes to the ASCII file.

Syntax (with an ASCII file):

```
ASSIGN GROUP @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Data.txt"

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax

```
MAINTENANCE ON
```

```
ASSIGN GROUP @Access.txt
MAINTENANCE OFF
```

See also

- QuickFormat

Assign ProcessMap

Use this command to assign a Longview Process Map to an existing category. You must first publish the Longview Process Map before you can assign it to categories. Use separate lines to assign a Longview Process Map to multiple categories. For information on publishing a process map, see Publish ProcessMap. For information on creating categories, see Put Category.

Syntax:

```
ASSIGN PROCESSMAP ProcessID CATEGORY Category
```

where:

- ProcessID is the ID of the Longview Process Map to be assigned to the category. If the Longview Process Map ID includes spaces, enclose it in double quotation marks. Longview Process Map IDs cannot exceed 100 characters.
- Category is the name of the category to which the Longview Process Map is assigned. If the category name includes spaces, enclose it in double quotation marks.

Syntax example:

```
ASSIGN PROCESSMAP SampleProcessMap CATEGORY Administrators
```

See also

- Create Category
- Put Category

Assign Symbol

Use this command to specify a parent symbol for a child symbol in the Data Server repository.

You can use the Assign Symbol command to specify a parent symbol for an unattached child symbol in the Data Server repository, or to specify multiple parent symbols for an attached child symbol in the Data Server repository. You can have a maximum of 47 levels in a hierarchy in the Data Server repository, and a symbol can have up to 99 parents.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Note: If you have an existing database that is already partitioned, and you want to add child symbols and assign parent symbols, add the child symbols first, and then assign them to the appropriate parents. If you add a symbol, and then make it a parent, Longview Application Framework moves the symbol to a different partition, and you will not be able to assign the parent/child relationship properly.

Syntax (regular)

```
ASSIGN SYMBOL DimName ChildSymName PARENT ParentSymName ["Weight"
[Priority]]
```

where:

- DimName is a dimension containing the symbol.
- ChildSymName is a child symbol.
- ParentSymName is a parent symbol. You can have a maximum of 47 levels in a hierarchy in the Data Server repository, and a symbol can have up to 99 parents.
- Weight is the mathematical effect of the child symbol on its parent symbol, enclosed in double quotation marks. Valid options are "+", "-", and "0". This parameter is optional.
- Priority is a number that designates the symbol's position in the hierarchy relative to its parent. Symbols are listed in order of ascending priority, with zeros falling at the bottom of the list. This parameter is optional; however, you can specify Priority only if you also specify Weight.

Syntax example:

```
MAINTENANCE ON
ASSIGN SYMBOL ACCOUNTS NewSales PARENT SaleT "+" 1
MAINTENANCE OFF
```

Creating an ASCII input file

To specify parent symbols for a large number of symbols, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each symbol:

```
ChildSymName{PARENT{ParentSymName{Weight[Priority]
```

Syntax (with an ASCII file):

```
ASSIGN SYMBOL DimName @FileName
```

where:

- DimName is a dimension containing the symbols.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
ASSIGN SYMBOL ACCOUNTS @C:\Temp\Test.asc
MAINTENANCE OFF
```

See also

- Create Symbol

Assign User

Use this command to assign symbol access, authorizations, and group membership to a user.

You can assign access to a total of 767 separate Longview symbols to a user. For example, suppose User1 belongs to Group1. Group1 has access settings to 225 symbols. User1 has access to an additional and separate 542 symbols not listed in the group he belongs to. In total, User1 has access to 767 symbols. Symbol access roles can also been assigned to users and user groups.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax (regular)

```
ASSIGN USER [DomainName\]UserId ACCESS Dimension RoleName Inherit SymbolName
AccessType NumLevels [Priority]

ASSIGN USER [DomainName\]UserId AUTHORIZATION AuthorizationName
[AuthorizationGroup]

ASSIGN USER [DomainName\]UserId GROUP GroupName
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the name of the user to which you want to assign access, authorizations, or groups.
- Dimension is a dimension containing the symbols.
- RoleName is the name of a symbol access role.
- Inherit designates that the privileges are inherited from the role for the dimension specified. Valid values are TRUE (inheriting access from role) and FALSE (setting specific access). If Inherit is TRUE, do not include SymbolName, AccessType, NumLevels, and Priority.
- SymbolName is the top symbol in the hierarchy to which you want to give the user access. It must be a subset of the role. If Inherit is TRUE, do not include this parameter.
- AccessType is a setting that determines whether the user has read/write access. If Inherit is TRUE, do not include this parameter. Select one of the following:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- NumLevels is a number representing the number of hierarchy levels below SymName. If Inherit is TRUE, do not include this parameter.

- Priority is optional and is a number that designates a symbol's position in the hierarchy relative to its parent. Symbols are listed in order of ascending priority, with zeros falling at the bottom of the list. If Inherit is TRUE, do not include this parameter.
- AuthorizationName is a name that designates the type of authorization being granted to the user.

Value	Description	Platform License
ADDINFOROFFICE	Access the Longview Add-In for Office. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Longview Add-In for Office.	Designer Power User Viewer
ADDINFOROFFICE SUBMITDATA	Submit data in the Longview Add-In for Office.	Designer Power User
ANALYSISREPORTING AUTHOR	Access Longview Analysis and Reporting as a Report Author.	Power user
ANALYSISREPORTING PUBLISHER	Access Longview Analysis and Reporting as a Report Publisher.	Designer
ANALYSISREPORTING USER	Access Longview Analysis and Reporting as a Report User.	Viewer
APPLICATION ADMINISTRATOR	Access Longview Application Administrator.	Admin Designer Service
ATTRIBUTE	Manage attributes.	Designer
BATCH	Manage batches.	Admin Designer
CONNECTVIA APPLICATION FRAMEWORK	Connect to the server using Longview Application Framework. **Note:** You must assign this authorization to allow users or user groups to access Longview Apps, Longview Designer, Longview tools and editors, the Longview Add-In for Office, and Longview Tax.	Admin Designer Power User Viewer Service

Value	Description	Platform License
DASHBOARDDESIGNER	Access Longview Dashboard Designer.	Designer
DELETECOMMENTS	Delete any existing comments in the Data Server. Users without Delete Comments authorization can delete only their own comments before they are submitted to the database. **Note:** Delete existing comments functionality is available in Data Grids only.	Designer
DESIGNER	Access Longview Designer. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSCREATE	Create data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSDELETE	Delete data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSMODIFY	Edit data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSPUBLISH	Publish data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer

Value	Description	Platform License
DESIGNERLONGVIEWAPPSPUBLISH	Publish Longview Apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
FOREIGNEXCHANGESETTINGS	Manage foreign exchange settings.	Admin Designer
GROUPADMINISTRATION	Perform group administration.	Does not apply
GROUPSYMBOLACCESS	Manage symbol access for groups.	Does not apply
INTERCOMPANYSETTINGS	Manage intercompany settings.	Designer
JOURNALENTRIES	Access Longview Journal Entries.	Designer Power User
JOURNALENTRY	Manage journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODCREATE	Create current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODPERMPOST	Permanently post current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODREVIEWPOST	Review post current period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYDELETE	Delete non-shared journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODCREATE	Create future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODPERMPOST	Permanently post future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODREVIEWPOST	Review post future period journal entries.	Designer Power User Service
JOURNALENTRYOWN PERMPOST	Permanently post own journal entries.	Designer Power User Service
JOURNALENTRYOWN REVIEWPOST	Review post own journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJCREATE	Create prior period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYPRIOR PERIODADJPERMPOST	Permanently post prior period journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJREVIEWPOST	Review post prior period journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTCREATE	Create restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTPERMPOST	Permanently post restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTREVIEWPOST	Review post restatement journal entries.	Designer Power User Service
LOCK	Manage locks.	Designer Service
MAPPINGSEDITOR	Access the Mappings editor. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service

Value	Description	Platform License
MAPPINGSMANAGE	Create, modify, delete mappings. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MAPSMANAGE	Create, modify, delete maps. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MODIFYDATA	Submit data.	Designer Power User Service
NDDSETTINGS	Manage NDD settings.	Designer
ROLE	Manage symbol access roles.	Designer Service
RULE	Manage rules.	Designer Service
SCHEDULE	Manage schedules.	Designer Service
SERVERMANAGER	Access Longview Server Manager.	Admin Service Designer
SERVERMANAGERSTART	Start/stop the Longview server.	Admin Service Designer

Value	Description	Platform License
SERVICEACCOUNTUSERADMINISTRATOR	Sets the user to be a User Administrator.	Service
SERVICEACCOUNTRESTAPI	Sets user to be a Service Account User. **Note:** Can only be set using PUSER or a user who has User Administrator Authorization.	Admin Designer Service
SYMBOL	Manage symbols.	Designer Service
SYMBOLASSIGN	Assign symbols to a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLATTRIBUTECREATE	Create symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEDELETE	Delete symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEMODIFY	Modify symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLCANCREATEROOT	Create root symbols. Must be used with SYMBOL, CREATESYMBOL.	Designer Service
SYMBOLCANDELETEROOT	Delete root symbols. Must be used with SYMBOL, SYMBOLDELETE.	Designer Service

Value	Description	Platform License
SYMBOLCREATE	Create symbols. Must be used with SYMBOL.	Designer Service
SYMBOLDELETE	Delete symbols. Must be used with SYMBOL.	Designer Service
SYMBOLREMOVE	Remove symbols from a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLSET	Modify symbols. Must be used with SYMBOL.	Designer Service
SYMBOLSWITCH	Switch symbols in a hierarchy. Must be used with SYMBOL.	Designer Service
SYSTEMATTRIBUTECREATE	Create system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEDELETE	Delete system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEMODIFY	Modify system attributes. Must be used with ATTRIBUTE.	Designer Service
USERADMINISTRATION	Perform user administration	Does not apply
USERATTRIBUTECREATE	Create user attributes. Must be used with ATTRIBUTE.	Designer Service

Value	Description	Platform License
USERATTRIBUTEDELETE	Delete user attributes. Must be used with ATTRIBUTE.	Designer Service
USERATTRIBUTEMODIFY	Modify user attributes. Must be used with ATTRIBUTE.	Designer Service
USERRESETPASSWORD	Reset passwords.	Does not apply
USERSYMBOLACCESS	Manage symbol access for users.	Does not apply
VIEWDATA	View data.	Designer Power User Viewer Service
WORKFLOWDESIGNER	Access Longview Workflow Designer.	Designer

- AuthorizationGroup is required only when AuthorizationName is USERADMINISTRATION or USERRESETPASSWORD, and is the group for which you want the indicated user to have authorization.
- GroupName is the name of a group to which the user membership is assigned membership.

Syntax example:

```
MAINTENANCE ON
ASSIGN USER JSmith GROUP Administrators
MAINTENANCE OFF
```

Creating an ASCII input file

To change the access privileges of a large number of users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId{ACCESS{Dimension{RoleName{Inherit{SymbolName{AccessType
{NumLevels[{Priority]
[DomainName\]UserId{AUTHORIZATION{AuthorizationName[{AuthorizationGroup]
```

```
[DomainName\]UserId{GROUP{GroupName
```

Note: Include SymName, Access, NumLevels, and Priority only if Inherit is set to FALSE.

Use the QUICKFORMAT command, if necessary, to make formatting changes to the ASCII file.

Syntax (with an ASCII file):

```
ASSIGN USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
ASSIGN USER @Access.txt
MAINTENANCE OFF
```

See also

- QuickFormat

AuditTrail

Use this command to create comments in the Data Server.

The Longview Data Server Log file, lv_dataserver.log, records a wide variety of activities in the Data Server. To make certain actions more noticeable, you may want to insert a comment to create an audit trail — for example, to draw attention to the beginning of a process.

Syntax:

```
AUDITTRAIL Keyword "String"
```

where:

- Keyword is any character string. Use Keyword to identify a type of item to insert into the Longview Data Server Log file, for ease of search at a later time.
- String is a character string to be used as a comment for Keyword, enclosed in double quotation marks.

Syntax example:

```
AUDITTRAIL Symbol_Maintenance "Gilda's test"
```

See also

- Get Log

AutoLoad

Use this command to enable or disable the automatic population and use of all documents within your procedure to the object cache. For Longview Apps, this command can be used in conjunction with the <kars> parameter in your .lvapp file and/or the Load and LoadKar commands. This command is helpful if, for example, you have a Longview App with many supporting documents and images and want to ensure all necessary documents are added to the cache.

Syntax:

```
AUTOLOAD ON|OFF
```

where:

Value	Description
ON	Automatically loads all documents into the object cache.
OFF	Documents are not automatically loaded into the object cache. This is the default. Use the <kars> parameter in your .lvapp file and/or the Load and LoadKar commands to load documents into the object cache.

Syntax example:

```
AUTOLOAD ON
```

See also

- Load
- LoadKar
- Unload

BatchComment

Use this command to specify a batch comment to associate with the next data submission. This command can also be used to specify a batch comment when posting and unposting calculated journal entries using the functions JournalEntryPost() and JournalEntryUnpost().

When checking on the status of a batch by means of the User Submissions tool, you may want to make it easy to identify. To do so, you use the BatchComment command to store a comment associated with the batch.

Note: This command must be used before submissions, the Upload command, or the JournalEntryPost() and JournalEntryUnpost() functions.

Syntax:

```
BATCHCOMMENT "String"
```

where:

- String is a character string to be used as a comment.
- Enclose in double quotation marks.
- To include double quotation marks in the character string, precede the interior double quotation marks with a backslash (\ ).

Syntax example:

```
BATCHCOMMENT "Spencer’s batch"
```

BatchWait

Use this command in conjunction with data submissions to check if the submission is fully processed before proceeding to the next command in the Procedure document. This command can also be used in conjunction with the functions JournalEntryPost() and JournalEntryUnpost().

> **Note:** BatchWait must be turned on prior to any submissions, or the JournalEntryPost() and JournalEntryUnpost() functions.

The BATCHWAIT command polls the server every specified number of seconds to check the status of a batch. Once the batch has been completed, the next command is executed.

Syntax:

```
BATCHWAIT Setting [WaitTime]
```

where:

- Setting can be one of the following:
  - ON — Use this value if you want to wait for the batch to complete.
  - OFF — Use this value if you want to exit without waiting for the batch to complete.

  The default value is OFF.
- WaitTime is optional, and indicates the period of time to wait (in seconds) between polling the server for batch status. The default value is 60 seconds.

Syntax example:

```
BATCHWAIT ON 120
```

See also

- Upload (for DataAreas)
- Reconcile Upload

Begin AdminService

Use this command create a session to the Data Servers Administration Port.

Syntax:

```
BEGIN AdminService "FileName" [Host|URL] [Port]
```

where:

- FileName is the name of a text file to which the output from the Longview Server Manager commands will be saved, enclosed in double quotation marks.
- Host is the name of the host computer of the data servers. This parameter is optional if specified in the lv_af.cfg file
- URL is the Admin HTTP proxy URL when using Longview HTTP Proxy Server.
- Port is the Administrator port number (typically ending in 0). This parameter is used with the Host parameter only. This parameter is optional if specified in the lv_af.cfg file

This command will also use with the lv_af.cfg file. If the host and port exist in the lv_af.cfg file found in the working directory of the lv_af.exe then the command will use these to get the Host and Port when the command does not specify them. To specify the host and port, open the lv_af.cfg file in your preferred text editor, and add the ADMIN_HOST, ADMIN_PORT

For example:

ADMIN_HOST = e3000
ADMIN_PORT = 9700

Syntax example:

```
BEGIN AdminService "OutputFile.txt" e3000 9700
LVMGRL "User Admin Admin1" "restate unadjusted" "archive monitor"
END AdminService
```

See also

- End AdminService
- LVMGRL

CancelClose

Use this command to prevent a Data Grid or HTML window from closing when a user clicks the window close button (the x in the top-right of the window's title bar) in a Longview Data Grid or HTML window of a Longview App. For information on specifying close behavior, see OnClose.

Syntax:

```
CANCELCLOSE
```

Syntax example:

```
Set VARIABLE ExitOption = ShowPrompt("Would you like to save your changes
before closing?", "YesNoCancel")
If "$ExitOption$" EQ "YES"
//Upload changes to DataArea and Close
Upload DataArea1
End If
If "$ExitOption$" EQ "Cancel"
//Cancel the Close action
CancelClose
End If
```

Clear DataArea

Use this command to clear all the non-zero values in a specific DataArea. In order to use this command, you must first select and download a DataArea. The cleared values are temporarily marked and will be submitted to the server during the next upload process.

Syntax:

```
CLEAR DataAreaName [ProtectedMode]
```

where:

- DataAreaName is the name of the DataArea containing the non-zero values to be cleared.
- ProtectedMode is optional and is either TRUE or FALSE. If TRUE, unlocked cells are not affected. If FALSE, all cells are affected.

Note: When ProtectedMode is TRUE, the DataArea is cleared only for cells covered by the data lock.

Syntax example:

```
CLEAR daMaster
```

See also

- Create DataArea
- Download (for DataAreas)

Close Cursor

Use this command, in conjunction with the Fetch and Open Cursor commands, to close an open cursor.

Syntax:

```
Close Cursor CursorName
```

where:

- CursorName is the name of the open cursor.

Syntax example:

```
Create Variable amountRow[] as String
Create DataTable EmployeeSalary Using AllSalaries.lvdtd
Download EmployeeSalary
Show DataTable EmployeeSalary
Open Cursor amounts Select "Employee ID, Entity, Amount" From EmployeeSalary
While $LVS_FetchStatus$ != -1
Fetch Next DataTableRow from amounts Into amountRow
If $LVS_FetchStatus$ EQ 0
Run Model TransferTableData.lvmod
End If
End While
Close Cursor amounts
```

Collect Statistics

Use this command to gather statistics that might help improve model performance for the specified calculation dimension. Note that these statistics are used by the SkipNoData model function to maximize efficiency (therefore, they will have no benefit unless SkipNoData is also used). You can collect statistics for multiple dimensions (useful if executing models using multiple calculation dimensions).

After this command is called, any changes to the specified DataArea will automatically update the statistics until the Delete Statistics command is called. You can collect statistics for multiple dimensions by calling the command additional times using different dimension parameters.

Syntax:

```
COLLECT STATISTICS ON DataAreaName DimName
```

where:

- DataAreaName is the name of the DataArea for which to collect statistics.
- DimName is the calculation dimension.

Syntax example:

```
COLLECT STATISTICS ON testarea accounts
COLLECT STATISTICS ON testarea entities
```

See also

- Delete Statistics
- SkipNoData

Connect

Use this command to sign on to the Data Server repository. This command is issued from within a Procedure document.

You can use this command with the user ID of an Administrator, or a regular user.

Syntax for Longview authentication

```
CONNECT User/Password Host Port LID Role [UserGroup] [LanguageCode]
```

where:

- User/Password is your user ID and password (separated by a forward slash). If the user name contains a space, it must be enclosed in double quotation marks ("").
- Host is the name of the host computer.
- Port is the port number used by the Listener (typically ending in 1).
- LIDis your Longview identifier.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.
- UserGroup is a user group. This is a mandatory parameter if V3_COMPATIBLE_ACCESS is used for the Role parameter.
- LanguageCode is the optional language code.

Syntax example:

```
CONNECT demov7/demov7 e3000 9701 LVProd V3_COMPATIBLE_ACCESS Analyst
```

Syntax for Windows authentication

```
CONNECT / Host Port LID Role [LanguageCode]
```

where:

- Host is the name of the host computer.
- Port is the port number used by the Listener (typically ending in 1).
- LID is your Longview identifier.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_

ACCESS.

- LanguageCode is the optional language code.

Syntax example:

```
CONNECT / e3000 9701 LVProd V3_COMPATIBLE_ACCESS Analyst
```

Syntax for Single Sign-on

If your system is setup to use OAuth authentication, then you can use the following connect command to connect to AF using single sign-on:

```
CONNECT ClientId/Secret SSO Scope Host Port LID Role [UserGroup] [LangCode]
```

where:

- ClientID/Secret is the Client ID set in the OAuth flow of your IDP to create the Longview session with. The Client ID must exist in the Longview Database as a third-party user.
- Scope is the value configured in the IDP application, if the IDP application and Longview SSO Service has been configured to use a scope. If the application does not use scope then use "".
- Host is the name of the host computer.
- Port is the port number used by the Listener (typically ending in 1).
- LID is your Longview identifier.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.
- UserGroup is a user group. This is a mandatory parameter if V3_COMPATIBLE_ACCESS is used for the Role parameter.
- LanguageCode is the optional language code.

Syntax example:

```
CONNECT 0oagk9dfghK83OO405d7/qPa-pz0D8mgi4zDZRR2E SSO "" e3000 9701 LVProd
V3_COMPATIBLE_ACCESS Administrators

CONNECT 0oagk9dfghK83OO405d7/qPa-pz0D8mgi4zDZRR2E SSO LVRestAPI e3000 9701
LVProd V3_COMPATIBLE_ACCESS Administrators
```

Syntax for ISW platform (AAD)

If your system is on ISW platform with AAD, then the following syntax must be used to be able to connect. The user must also be a service account with Connect to Application Framework authorization.

You must have an Azure Active Directory setup to be able to use this connect syntax.

```
CONNECT ClientId/Secret AAD Tenant Environment ClientProxyURL LID Role
[userGroup] [LangCode]
```

where:

- ClientID/Secret is the application registered in the Azure Active Directory server that is used to request an access token.
- Tenant is the name of the Azure Active Directory server that is used to populate the AAD URL to request access token.
- Environment is the target B2C platform name that is used to populate the scope of the access token.
- ClientProxyURL is the URL of the client proxy
- LID is your Longview identifier.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ ACCESS.
- UserGroup is a user group. This is a mandatory parameter if V3_COMPATIBLE_ACCESS is used for the Role parameter.
- LanguageCode is the optional language code.

Syntax example:

```
CONNECT 4c796c6b-21u5-463b-8ddd-c5839e5c41e7/qPa-pz0D8mgi4zDZRR2E AAD
            AzureServer iswb2c https://localhost:8080/client/ LVProd V3_
COMPATIBLE_
            ACCESS Administrators
```

Syntax for ISW Platform Organisation key

If your system is on ISW platform and uses ISW platform Organization API Keys for authentication, then the following syntax must be used to be able to connect. The user must be a service account with Connect to Application Framework authorization.

You must have an ISW platform organization key configured in your system to use this connect syntax.

```
CONNECT APIKey APIKEY ClientProxyURL LID Role [UserGroup] [LangCode]
```

Where:

- APIKey is the organization API Key that was generated in the ISW platform.
- ClientProxyURL is the URL of the client proxy.

- LID is your Longview identifier.
- Role is the name of the symbol access role. The value typically used is V3_COMPATIBLE_ACCESS.
- UserGroup is a user group. This is a mandatory parameter if V3_COMPATIBLE_ACCESSS is used for the Role parameter.
- LangCode is the optional language code.

Syntax example:

```
CONNECT 1234567890abcdef1234567890abcdef12345678 APIKEY
https://localhost:8080/client/  LVProd V3_COMPATIBLE_ACCESS Administrators
```

Syntax for ISW platform external authentication (OPENID)

If your system is on ISW platform and are using an external authenticated service account, then the following syntax must be used to be able to connect. The user must also be a service account with Connect to Application Framework authorization.

You must have an external authenticated service using OPENID setup to be able to use this connect syntax.

```
CONNECT ClientId/Secret OPENID Issuer Environment ClientProxyURL LID Role
            [userGroup] [LangCode]
```

where:

- ClientID/Secret is the application registered in the Azure Active Directory server that is used to request an access token.
- Issuer is the external IDP issuer URI server that is used to request the access token.
- Environment is the target B2C platform name that is used to populate the scope of the access token.
- ClientProxyURL is the URL of the client proxy
- LID is your Longview identifier.
- Role is the name of a symbol access role. The value typically used is V3_COMPATIBLE_ACCESS.
- UserGroup is a user group. This is a mandatory parameter if V3_COMPATIBLE_ACCESS is used for the Role parameter.
- LanguageCode is the optional language code.

Syntax example:

```
CONNECT 4c796c6b-21u5-463b-8ddd-c5839e5c41e7/qPa-pz0D8mgi4zDZRR2E

 OPENID https://prod-u23s.us.auth0.com iswb2c https://localhost:8080/client/
LVProd V3_COMPATIBLE_ACCESS Administrators
```

Copy Data

Use this command to copy non-zero data in the Data Server repository from one symbol to another.

Use this command to directly manipulate data in the Data Server repository, based on an intersection of data specified. Copy Data From does not overwrite existing non-zero data in the target area unless there is non-zero data in the source area for that data. When copying data from a parent symbol to a child symbol, the data is copied even if the parent symbol is a virtual parent.

If a user has any symbol access that is read-only in the area defined by the lock to which the command pertains, the Copy Data command is allowed to continue and all read-only symbols will be ignored (not copied). If you try to copy to an area that is read-only, an error message appears.

Syntax:

```
COPY DATA FROM DimName SourceSymName TO TargetSymName [DataType] USING
dataspec.lvdsp
```

where:

- DimName is a dimension containing the symbols to copy data.
- SourceSymName is a symbol whose data you want to copy.
- TargetSymName is a symbol that will contain the copied data.
- DataType is optional and can be one of the following:

Value	Description
PARENT	To copy data from (UP + AP) to UL table. The PARENT option does not copy the rollup value of a carryforward parent. If SourceSymName is a virtual parent, the value of the source symbol is recalculated before copying it.
LEAF	To copy data from UL to UL table.
CALCLEAF	To copy data from (UC + UL + AC + AL) to UL table.
CTA	To copy data from (UT, AT) to (UT, AT) table.
ADJUSTEDLEAF	To copy data from (UL + UA) to UL table.

- dataspec.lvdsp is the name of the DataSpec file.

The following table explains the abbreviations in the DataType parameters:

Parameter	Description
U	unadjusted
A	adjusted

Parameter	Description
C	calculated
L	leaf
P	parent
T	CTA
+	add value for the identical coordinate
,	the corresponding table (for example, unadjusted to unadjusted)

Syntax example:

```
CREATE LOCK USER USING CopyNI.lvdsp "Copy plan income"
COPY DATA FROM TIMEPER A0312 TO P0401 CALCLEAF USING CopyNI.lvdsp
DELETE LOCK USER USING CopyNI MATCH
```

Copy File

Use this command to duplicate an existing file. This command works with absolute and relative paths.

The allowed paths for the Copy File command are specified by the PERMITTED_FILEMAINTENANCE_PATH parameter in the lv_af.cfg file, located in the working directory of the lv_af.exe. Files that cannot be copied are defined by the EXCLUDED_FILEMAINTENANCE_FILES parameter in the same file. These parameters only apply when they are set.

Syntax:

```
COPY FILE FROM "SourceFile" TO "[Destination]/[FileName]" [Overwrite]
```

where:

- SourceFile is the name of the file to be copied. The source file can be specified by it's full path or by the name as long as it is located in the working directory.
- Destination is optional and is used to specify the location where the file should be copied to. If it's not specified the SourceFile location will be used, unless the SourceFile location is not specified then it will be copied to the current working directory. If Destination is not specified, the FileName must be.
- FileName is optional and is used to name the copied file. If the FileName is not used, the file will be copied with the same name. If FileName is not specified, then the Destination must be.
- Overwrite is optional and specifies whether to overwrite a file that already exists with the same name.

Syntax example:

```
COPY FILE FROM "logs\export\export.log" TO "copy_export.log" OVERWRITE
```

See also

- Create Directory
- Delete File
- FileExists
- GetFileProperties
- Move File
- Rename File

Create Attribute

Use this command to create an Attribute definition.

An attribute is data used to describe the characteristics of an object in Longview. For example, SWFAdminEMail is an Attribute representing the email address of an administrator.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax (regular)

```
CREATE ATTRIBUTE AttrClass AttrName AttrDesc AttrType AccessType
DefaultValue
```

where:

- AttrClass is the Attribute class. Select one of the following:

Value	Description
SYSTEM	Describes the entire Longview system at the highest level. Attributes of this Attribute class specify system-wide characteristics. There is only one object in the SYSTEM Attribute class – the application itself.
USER	Describes the Attributes of a particular user. Each user is an object in the USER Attribute class.
SYMBOL	Describes the characteristics of individual symbols. Each symbol is an object in the SYMBOL Attribute class.

- AttrName is the Attribute name.
- AttrDesc is the Attribute description. Descriptions for new attributes are automatically saved as English. If you want the attribute to have an alternate language description, you must use the Set Attribute command. For more information, see Set Attribute.
- AttrType is the Attribute type. Select one of the following:

Value	Description
DATE	Date and time.
DATELIST	List of one or more Date values.
DOUBLE	Numeric value.
DOUBLELIST	List of one or more numeric values.
INTEGER	Integer value.
INTEGERLIST	List of one or more INTEGER values.
STRING	Alphanumeric character string.

Value	Description
STRINGLIST	List of one or more STRING values.
SYMBOL	A valid symbol.
SYMBOLLIST	List of one or more SYMBOL values.

- AccessType is a setting. Select one of the following:

Value	Description
READ	Read-only access, to allow the user to view the attribute value but not change it. This is the default. Symbol and System Attributes are always Read type.
WRITE	Write access, to allow the user to set the attribute value. Write can only be used for User Attributes.

- DefaultValue is the default value for the Attribute.

Syntax example:

```
MAINTENANCE ON
CREATE ATTRIBUTE SYSTEM SYSTEST "System Test" LONG Read "100"
MAINTENANCE OFF
```

Creating an ASCII input file

To create a large number of Attribute definitions, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

Syntax:

```
AttrClass{AttrName{AttrDesc{AttrType{AccessType{AttrDefault
```

Syntax (with an ASCII file):

```
CREATE ATTRIBUTE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
CREATE ATTRIBUTE @sysattr.asc
```

See also

- Append Attribute
- Delete Attribute
- Refresh Attribute
- Set Attribute

Create Category

Use this command to create a new category.

Syntax (regular)

```
CREATE CATEGORY CategoryName ["Description"]
```

where:

- CategoryName is the name of the category you want to create. The following characters are invalid: backslash (\ ), forward slash (/), apostrophe (’), double quotation marks (“), comma (,), left brace ({), and double brackets ([[ ) or ( ]]). The category name can have a maximum length of 40 characters.
- Description is the string describing the category (for example, the category’s function), enclosed in double quotation marks. If the Description is not specified, the CategoryName will be used as the description. The following characters are invalid: left brace ({). The category description can have a maximum length of 100 characters.

Syntax example:

```
CREATE CATEGORY ConsolInput "Consolidation Input"
```

Creating an ASCII input file

To create multiple categories, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user group:

Syntax:

```
CategoryName{Description
```

Syntax (with an ASCII file):

```
CREATE CATEGORY @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Data.txt"

Syntax example:

```
CREATE CATEGORY @newCategories.txt
```

Create DataArea

Use this command to create a DataArea for use with Models, ImportSpecs, and ExportSpecs. This is also used in conjunction with other commands as well, such as UPLOAD. You can also create a DataArea that you can view as a Data Grid in a Longview App.

Initially, a DataArea is created with structure only and the DOWNLOAD command must be used to populate it with data from the database. The UPLOAD command is used to send any data changes back to the database.

Note: When using the UPLOAD command, the maximum number of open DataAreas allowed at any given time is 31.

The following rules apply to the use of DataAreas created by another procedure:

- If a DataArea is created in the parent procedure, it can be used in a child procedure (i.e., a child procedure inherits all the DataAreas created in its parent(s)).
- If a DataArea is created in the child procedure, it is only visible in that procedure and all of its child procedures.
- If an inherited DataArea is deleted in the child procedure, the data will not be cleared from the memory and is still accessible from the parent procedure. However, the child procedure will lose its reference to that DataArea.
- Downloading or uploading an inherited DataArea in a child procedure has the same effect as doing so in the main procedure.

Note: For the purposes of more readily identifying DataAreas as such, it is recommended that DataArea names be prefixed with da (e.g., daBalSheet). Note that this is only a recommendation and not an actual requirement of the command.

If you use this command in conjunction with the UseInputRelatedSchedules command, the original DataArea is created along with the related DataAreas for line item details, comments, and attachments.

For information on how this command is affected by input-related schedules, see UseInputRelatedSchedules.

Syntax:

```
CREATE DATAAREA DataAreaName USING dataspec.lvdsp
```

where:

- DataAreaName is the name to assign to the new DataArea.
- dataspec.lvdsp is the name of a DataSpec file.

Syntax example:

```
CREATE DATAAREA daMaster USING Master.lvdsp
```

See also

- Create Lock
- Delete DataArea
- Download (for DataAreas)
- Show DataArea
- Upload (for DataAreas)

Create DataRole

Use this command to create a new DataRole.

Syntax:

```
CREATE DATAROLE RoleName
```

where:

- RoleName is the name of the DataRole to create.

Syntax:

```
CREATE DATAROLE NorthAmAccounts
```

Creating an ASCII input file

To create a large number of DataRoles, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
RoleName
```

Syntax (with an ASCII file):

```
CREATE DATAROLE @ FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Create DataTable

Use this command to create a new, in-memory DataTable object that can be rendered in the user interface in Table or Calendar format.

Initially, a DataTable is created with structure only and you must use the Download (for DataTables) command or an ImportSpec to populate it with data. You can use the Upload (for DataTables) command to send any data changes back to the relevant App table.

Note: When using the UPLOAD command, the maximum number of open DataTables allowed at any given time is 31.

Syntax:

```
CREATE DATATABLE DataTableName USING "DataTableDefinition"
```

where:

- DataTableName is the name to assign to the new DataTable object.
- DataTableDefinition is the DataTable definition document (with the recommended extension .lvdtd).
  For information on creating this document, see Defining DataTable objects.

Syntax example:

```
CREATE DATATABLE Salaries USING "AllSalaries.lvdtd"
```

See also

- Upload (for DataTables)
- Download (for DataTables)
- Delete DataTable
- Show DataTable

Create Directory

Use this command to create a new directory (or folder). This command works with absolute and relative paths.

The allowed paths for the CREATE DIRECTORY command are specified in the lv_af.cfg file, defined by the PERMITTED_FILEMAINTENANCE_PATH parameter, located in the working directory of the lv_af.exe. This parameter only applies when it is set.

Syntax:

```
CREATE DRIECTORY "Directory"
```

where:

- Directory is the name of the folder or directory that to be created.

Syntax example:

```
CREATE DIRECTORY "D:\Longview\CustomFolderPath\LVProd\MyFolder"
```

See also

- Copy File
- Move File
- Rename File

Create Document

Use this command in conjunction with the Write command to create a document in the document cache. You may find this useful when your application needs to create code dynamically. For example, you may need to create a DataSpec document in which the number of temporary symbols is unknown until run time.

Syntax:

```
CREATE DOCUMENT DocName
```

where:

- DocName is the name for the document in the document cache and must be unique. Optionally, you can include a file path. All file paths are relative to the document cache. If the path includes spaces, you must enclose it in double quotation marks.

Note: You do not need to specify a file extension; however, keep in mind that when you Write the document or Save it to a physical file, you must specify the name exactly as in the Create Document statement, including the path and extension.

Syntax example:

```
Create DOCUMENT MyDataView.lvdvw
```

Synta Xexample - With A File Path

```
Create DOCUMENT \Testing\MyDataView.lvdvw
```

Syntax Example - With A File Path That Has Spaces

```
Create DOCUMENT "\My Testing\MyDataView.lvdvw"
```

See also

- Write
- Save
- Unload

Create GlobalVariable

Use this command to create a new global variable. Global variables can be used by any model or procedure within the main procedure. For example, if main procedure A calls two procedures B and C, and procedure B creates a global variable, procedures A and C can also recognize that variable.

Syntax:

```
CREATE GLOBALVARIABLE VariableName [] AS STRING|NUM|RANGE|OBJECT
```

where:

- VariableName is the name of the variable to create, with the following recommended naming conventions for the purposes of more readily identifying a variable's type:

Variable Type	Recommended naming convention
STRING	Prefix STRING variables with s.
NUM	Prefix NUM variables with n.
RANGE	Prefix RANGE variables with r.
OBJECT	Prefix OBJECT variables with o.
all	Variables in a list or array should also have their names prefixed with the lowercase letter l (for example, a variable for a list of a string values might be named lsTemp1.) This convention is also advisable in preventing potential conflicts caused by the overlapping of variable names and keywords that might be implemented in Longview Application Framework in future releases. Note that this is only a recommendation and not an actual requirement of the command.

- [ ] designates the variable is to be created as a list.
- STRING designates a string value.
- NUM designates a numeric value.
- RANGE designates a list of symbols.
- OBJECT designates an object instance.

Syntax example:

```
CREATE GLOBALVARIABLE rSymbols AS RANGE
SET VARIABLE rSymbols = "SYMBOL1 | SYMBOL2 | SYMBOL3 | SYMBOL4"
```

See also

- Create Variable
- Set Variable

Create Group

Use this command to create a new user group.

Syntax regular:

```
CREATE GROUP GroupName "Description"
```

where:

- GroupName is the name of the user group you want to create.
- Description is the string describing the group (for example, the group's function), enclosed in double quotation marks.

Note: For more information on the guidelines for user group settings, see the *Longview Application Administrator Guide*.

Syntax example:

```
CREATE GROUP ABC "ABC Administrators"
```

Creating an ASCII input file

To create multiple user groups, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user group:

```
GroupName{Description
```

Syntax (with an ASCII file):

```
CREATE GROUP @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
CREATE GROUP @newgroups.txt
MAINTENANCE OFF
```

Create Lock

Use this command to lock sections of data in the Data Server repository. A lock is a Data Server repository security feature preventing a data intersection from being changed by another user or process until an operation is completed and the data intersection is released.

Use this command to create an explicit lock. The lock is “explicit” because it occurs only when you issue the appropriate commands. In conjunction with the Create Lock command, specify a DataSpec to limit the scope of symbols to lock.

If you use this command in conjunction with the UseInputRelatedSchedules command, the original DataArea is locked along with the related DataAreas for line item details, comments, and attachments.

For information on how this command is affected by input-related schedules, see UseInputRelatedSchedules.

Syntax:

```
CREATE LOCK USER|ADMIN USING dataspec.lvdsp [JECHECK] ["Comment"]
[PERSISTENT]
```

where:

- USER creates a lock that is owned by the current user.
- ADMIN creates a lock that is owned by the DBO. Admin locks are always maintained when an Application Framework session ends because a timeout is reached or a connection is lost.
- USING creates the lock based on the DataSpec defined by a DataSpec file.
- dataspec.lvdsp is the name of a DataSpec file.
- JECHECK checks for unposted or temporarily posted journal entries before creating the lock. If any exist, the lock does not occur, and an error message appears.
- "Comment" is a character string to be used as the lock comment.
  - Enclose in double quotation marks.
  - To include double quotation marks in the character string, precede the interior double quotation marks with a backslash (\ ).
  - PERSISTENT applies only to USER locks when your system is configured to use session locks. This parameter indicates that the lock should be created as a persistent lock. If you do not use this parameter, user locks are created as session locks, which are released when an Application Framework session ends because a timeout is reached or a connection is lost. If your system is not configured to use session locks, all user locks are persistent and must be deleted manually.

Syntax example:

```
CREATE LOCK User USING SubmitAdj.lvdsp "Submission of Adjustments"
PERSISTENT
```

See also

- Delete Lock

Create Property

Use this command to create a property on an existing variable of object type.

Syntax:

```
CREATE PROPERTY Object.Property AS STRING|NUM|OBJECT
```

where:

- Object.Property is the name of the property to create.

Syntax example:

```
// Create a variable of type object
CREATE VARIABLE TestObject as Object
// Create properties of the object
CREATE PROPERTY TestObject.comments as String
CREATE PROPERTY TestObject.anyNumber as Num
```

Create Rule

Use this command to specify server rules. The following table lists the available server rule types:

Rule Type	Description
model	Expression specifying a mathematical formula involving Longview symbols. When data is processed on the server, the system detects dependencies on these model rules, performs the required mathematical calculations, and creates output data.
rollup	Defines which areas of the database should be included or excluded from rollup logic.
query	Expression equating two Longview DataAreas. Query rules specify the placement of data virtually in a data intersection and are used to avoid storing redundant data in a system. When data is processed on the server, your application detects any query rules, retrieves data from the source intersection, and places it virtually into the target intersection as if it actually exists in the target intersection.
event	An event rule can be configured to run a process or email when data changes in the rule-specified section of the database. **Note:** If you make any changes to the list of persistent event rules in the rulepersists.txt file or the name of the procedure launched by the persistent event rules, you must restart the servers for your changes to take effect. For more information on persistent event rules, see the *Longview Application Administrator Guide*.
validation	A validation rule compares two cubes of data in the Longview database to make sure that certain cells within those cubes equal each other. For example, Total Accounts must equal Total Liabilities and Owner's Equity, for all symbols in the ENTITIES, CURRENCY, and TIMEPER dimensions.

You can work with server rules only if you are a user with administrator privileges.

Syntax (regular)

```
CREATE RULE [OVERWRITE] @FileName
```

where:

- OVERWRITE is an optional parameter enabling the system to overwrite an existing file specified by FileName.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Data.txt"

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

The ASCII file should be formatted as follows:

ASCII content	Description
3500	Rule ID
MODEL	Server rule type
{	Delimiter
KLX(REVCON###,YTDROOT###, APCO,TOTPROJ###,CSAR,DIM5SET) =KLX(REVCON###,YTDROOT###,APCO100, TOTPROJ###,CSAR,DIM5SET)*0.3	Server rule
{	Delimiter
N/A	Email message for an Event rule type.
{	Delimiter
APCO PROPORTIONING - REV & CONT	Rule description
{	

Syntax example:

```
CREATE RULE @C:\Temp\Rule.asc
```

Note: Descriptions for new rules are automatically saved as English. If you want the rule to have an alternate language description, you must use the Set Rule command. For more information, see Set Rule.

Create Schedule

Use this command to create schedules using a text file. Create a new text file, and for each new schedule, on a separate line, include the following:

Item File Syntax (for Standard Or Static Schedule)

Syntax

```
SchedName
SchedDesc
SchedType
ListDims
ExtraDimsNum
ExtraDimName
ExtraDimDesc
MirrorType
For MirrorType = MIRROR:
DimName;Attr=Value
LeafInclusion
SortType
For MirrorType = SIMPLE SYMBOL:
SymName1,SymName2,...
SortType
For MirrorType = NUMERIC:
StartRange,EndRange,Padding
SortType
```

where:

- SchedName is the name of the schedule.
- SchedDesc is the description of the schedule.
- SchedType is STANDARD or STATIC.
- ListDims is a list of the dimensions that do not receive rollups.
- ExtraDimsNum is the number of extra dimensions this schedule will have.
- ExtraDimName is the name of an extra dimension.
- ExtraDimDesc is the description of the extra dimension.

- MirrorType is one of MIRROR, SIMPLE SYMBOL, or NUMERIC.
  - For MirrorType = MIRROR:
    - DimName;Attr=Value is the name of a dimension, an attribute, and the value of that attribute.
    - LeafInclusion is either TRUE to include leaf symbols or FALSE and to not include leaf symbols.
    - SortType is the way symbols are sorted and can be UNSORTED to not sort symbols, SORT_NAME to sort symbols by name, or SORT_DESC to sort symbols by description.
  - For MirrorType = SIMPLE SYMBOL:
    - SymName1,SymName2,... is the list of symbols to be used in a schedule where MirrorType is SIMPLE SYMBOL.
  - For MirrorType = NUMERIC:
    - StartRange is the value at the beginning of a range in a schedule where MirrorType is NUMERIC.
    - EndRange is the value at the end of a range in a schedule where MirrorType is NUMERIC.
    - Padding is either TRUE to use decimal padding or FALSE to not use decimal padding.

Item File Syntax (for Intercompany Schedule)

Create a new text file, and for each new schedule, on a separate line, include the following:

Syntax:

```
SchedName
SchedDesc
SchedType
ListDims
ExtraDimsNum
ExtraDimName
ExtraDimDesc
MirrorType
For MirrorType = MIRROR:
DimName;Attr=Value
LeafInclusion
SortType
For MirrorType = SIMPLE SYMBOL:
SymName1,SymName2,...
SortType
For MirrorType = NUMERIC:
StartRange,EndRange,Padding
SortType
```

```
OffsetFlag
FixedDimFlag
FixedDimSymName
```

where:

- SchedName is the name of the schedule.
- SchedDesc is the description of the schedule.
- SchedType is STANDARD or STATIC.
- ListDims is a list of the dimensions that do not receive rollups.
- ExtraDimsNum is the number of extra dimensions this schedule will have.
- ExtraDimName is the name of an extra dimension.
- ExtraDimDesc is the description of the extra dimension.
- MirrorType is one of MIRROR, SIMPLE SYMBOL, or NUMERIC.
  - For MirrorType = MIRROR:
    - DimName;Attr=Value is the name of a dimension, an attribute, and the value of that attribute.
    - LeafInclusion is either TRUE to include leaf symbols or FALSE and to not include leaf symbols.
    - SortType is the way symbols are sorted and can be UNSORTED to not sort symbols, SORT_NAME to sort symbols by name, or SORT_DESC to sort symbols by description.
  - For MirrorType = SIMPLE SYMBOL:
    - SymName1,SymName2,... is the list of symbols to be used in a schedule where MirrorType is SIMPLE SYMBOL.
  - For MirrorType = NUMERIC:
    - StartRange is the value at the beginning of a range in a schedule where MirrorType is NUMERIC.
    - EndRange is the value at the end of a range in a schedule where MirrorType is NUMERIC.
    - Padding is either TRUE to use decimal padding or FALSE to not use decimal padding.
- OffsetFlag is either TRUE or FALSE and indicates whether or not there is an offset dimension.
- FixedDimFlag is either TRUE or FALSE and indicates whether or not there is a fixed dimension. This is used only when OffsetFlag is FALSE in an intercompany schedule.
- FixedDimSymName is the name of the fixed dimension symbol. This is used only when OffsetFlag is FALSE and FixedDimFlag is TRUE.

Syntax Example - For STANDARD MIRROR Schedule Type

```
MirrorSched1
Standard Mirror Schedule Example with 1 dimension
STANDARD
NONE
1
DIMNAME1
"Dimension 1 Description
MIRROR
ACCOUNTS;DIM0SET
FALSE
UNSORTED
```

Syntax Example - For STANDARD MIRROR Schedule Type With Two Dimensions

```
MirrorSched2
Standard Mirror Schedule Example with two dimensions
STANDARD
NONE
2
DIMNAME1
"Dimension 1 Description
MIRROR
ACCOUNTS;DIM0SET
FALSE
UNSORTED
DIMNAME2
"Dimension 2 Description
MIRROR
ENTITIES
FALSE
UNSORTED
```

Syntax Example - For INTERCOMPANY MIRROR Schedule Type With An Offset Dimension

```
MirrorSched3
Intercompany Mirror Schedule Example with offset dimension
INTERCOMPANY
NONE
1
OFFSET
"Dimension 1 Description
MIRROR
ENTITIES
FALSE
UNSORTED
TRUE
```

Syntax Example - For INTERCOMPANY MIRROR Schedule Type Without An Offset Dimension

```
MirrorSched4
Intercompany Mirror Schedule Example without offset dimension
INTERCOMPANY
NONE
1
NONOFF
"Dimension 1 Description
MIRROR
ACCOUNTS;DIM0SET
FALSE
UNSORTED
FALSE
TRUE
DIM0SET
```

Syntax Example - For NUMERIC Schedule Type

```
NumericSched
Numeric Schedule
STANDARD
```

```
NONE
1
DIMENSION1
"Dimension 1 Description
NUMERIC
1,100,False
UNSORTED
```

Syntax Example - For SIMPLE SYMBOL Schedule Type

```
SimpleSymSched
Simple Symbol Schedule
STANDARD
NONE
1
DIMNAME
"Dimension Description 1
SIMPLE SYMBOL
Taxes,Fees,Other
UNSORTED
```

Syntax (with an ASCII file):

```
CREATE SCHEDULE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Create Symbol

Use this command to create a symbol in the Data Server repository.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Note: Descriptions for new symbols are automatically saved as English. If you want the symbol to have an alternate language description, you must use the Set Symbol command. For more information, see Set Symbol.

Syntax regular:

```
CREATE SYMBOL DimName SymName "SymDesc" SymType SortOption BalanceType
[PARENT ParentName ["Weight" [Priority]]]
```

where:

- DimName is a dimension that will contain the new symbol.
- SymName is a name for the new symbol. For more ifnromation, see Symbol names.
- "SymDesc" is a character string to be used as a description of SymName, in English, enclosed in double quotation marks. For more ifnromation, see Symbol descriptions.
- SymType specifies the way SymName rolls up to its parent symbol, and must be one of the following:

Value	Description
Standard	To add the symbol into the parent (for example, monthly expenses added into annual expenses).
Carryforward	When the parent value equals the value of the cell immediately preceding it (for example, cash balance as a running year-to-date, where the year-end amount equals the amount for December).
Static	When the child value has no effect on the parent value – for example, for price symbols. (The parents of such symbols are used only to group symbols, not to aggregate their values.) Static symbols do not roll up in any direction, and override any assigned weights.

- SortOption is one of the following:

Value	Description
Ascending	Sorts the children symbols by name in ascending order.
Descending	Sorts the children symbols by name in descending order.
Manual	Allows manual sorting of children symbols based on priority.

- BalanceType is one of: Credit|Debit|Neither. Credit and Debit are only used for the Accounts dimension and indicate whether the account is a credit account or a debit account. All other dimensions use Neither.

- ParentName is the name of the new symbol's parent and is required if you are creating a child symbol but not required if you are creating a root symbol. If you are creating a child symbol, you must also include the keyword PARENT. Your authorizations determine the type of symbols you can create. Use the following table to review the type of symbols you can create based on your authorizations.

Do you have the SYMBOLCREATE authorization?	Do you have the SYMBOLCAN CREATEROOT authorization?	ParentName is...
YES	NO	mandatory – you must specify a parent because you are not authorized to create root symbols.
YES	YES	optional – if you choose to create a child symbol you must specify PARENT and ParentName. If you choose to create a root symbol then PARENT and ParentName are not required.
NO	YES	not available – you can create only root symbols and cannot specify a parent because you are not authorized to create child symbols.

- "Weight" is optional; however, you can specify Weight only if you also specify PARENT and ParentName. Weight must be enclosed in double quotation marks, and can be one of the following:

Value	Description
+	To add the symbol's value to its parent.
-	To subtract the symbol's value from its parent.
0	To ensure the symbol has no mathematical effect on its parent.

- Priority is a number that designates a symbol's position in the hierarchy relative to its parent. Symbols are listed in order of ascending priority, with zeros falling at the bottom of the list. This parameter is optional; however, you can specify Priority only if you also specify Weight.

Syntax example:

```
MAINTENANCE ON

CREATE SYMBOL ACCOUNTS NewSales "New Sales" Standard Manual Credit PARENT
Sales "0" 4

MAINTENANCE OFF
```

Creating ASCII input files

To create a large number of symbols in the Data Server repository, specify the parameters for this command in two ASCII files.

1. Create an ASCII file named SymFileName containing the following information for each symbol:

```
SymName{SymDesc{SymType{SortOption{BalanceType
```

2. Create an ASCII file named ParentChildFileName containing the following information for each symbol.

```
SymName{PARENT{NewParentSymName{Weight{Priority
```

Syntax (with an ASCII file):

```
CREATE SYMBOL DimName @SymFileName @ParentChildFileName
```

where:

- DimName is the dimension that will contain the new symbols.
- SymFileName is an ASCII file containing data on the characteristics of the symbols.
- ParentChildFileName is an ASCII file containing data on the parent/child relationships for the symbols.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

```
@"C:\My Documents\My Data.txt"
```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
CREATE SYMBOL ACCOUNTS @C:\Temp\SymName.asc @C:\Temp\PC.asc
MAINTENANCE OFF
```

See also

- Set Symbol for help in adding an alternate description to a symbol.
- Set Variable

Create User

Use this command to create a new user.

Syntax regular:

```
CREATE USER [DomainName\]UserId AuthType "Description" ["Password"]
"FirstName" "LastName" "Email" "HomePhone" "OfficePhone"
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the ID of the user you want to create.
- AuthType is the authentication method and can be EXTERNAL, LONGVIEW, or WINDOWS.
- Description is a description based on the user's role in the system, enclosed in double quotation marks.
- Password is mandatory for LONGVIEW authentication only, and is the user's password, enclosed in double quotation marks. This parameter should be omitted for other types of authentication.

  Note: Passwords cannot contain any of the following characters: double quotation marks ("), pipes (|), dollar sign ($), square brackets ([ ),( ]) or spaces ( ), and cannot start with the at sign (@).

- FirstName is the user's first name, enclosed in double quotation marks.
- LastName is the user's last name, enclosed in double quotation marks.
- Email is the user's email address, enclosed in double quotation marks.
- HomePhone is the user's home phone number, enclosed in double quotation marks.
- OfficePhone is the user's office phone number, enclosed in double quotation marks.

Note: You can include or exclude values for any combination of strings by using two double quotation marks ("") for null values preceding the parameters you want to include. For more information on the guidelines for user settings, see the Longview Application Administrator Guide.

Syntax example:

```
CREATE USER JSmith LONGVIEW "Admin" "password1" "John" "Smith"
"jsmith@abc.com" "555-555-5555" "555-555-5556"
```

Syntax example:

```
CREATE USER JDoe LONGVIEW "Admin" "password1" "Jane" "Doe" "" "" "555-555-
5678"
```

Creating an ASCII input file

To create multiple users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId{AuthType{Description{Password{FirstName{LastName{Email
{HomePhone{OfficePhone
```

Syntax (with an ASCII file):

```
CREATE USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
CREATE USER @newusers.txt
MAINTENANCE OFF
```

Create Variable

Use this command to create a variable in Longview Application Framework. Variables are recognized only in the procedure in which they are created and any document called by the procedure subsequent to the creation of the variable.

For information on creating variables that can be used by other procedures and models, see Create GlobalVariable.

A variable can have either a string value, a numeric value, or an object value and can be created either as a single item or can contain a list or array of items.

You can use the CREATE VARIABLE command in conjunction with the SET VARIABLE command to retrieve the balance type or description for a symbol. For more information, see the Syntax examples.

Note: The names of variables in Longview Application Framework to be used in data event triggers are limited to 31 characters. The convention is E_DIMNAME (where DIMNAME is the first 26 characters of the dimension name). In the case where the first 26 characters of the dimension names are the same, the variables are named E_DIMNAME_N (where DIMNAME is the first 26 characters of the dimension, and N is the dimension number). For more information, see Working with event variables.

Syntax:

```
CREATE VARIABLE VariableName [] AS STRING|NUM|RANGE|OBJECT
```

where:

- VariableName is the name of the variable to create, with the following recommended naming conventions for the purposes of more readily identifying a variable's type:

Variable Type	Recommended naming convention
STRING	Prefix STRING variables with s.
NUM	Prefix NUM variables with n.
RANGE	Prefix RANGE variables with r.
OBJECT	Prefix OBJECT variables with o.
all	Variables in a list or array should also have their names prefixed with the lowercase letter l (for example, a variable for a list of a string values might be named lsTemp1.) This convention is also advisable in preventing potential conflicts caused by the overlapping of variable names and keywords that might be implemented in Longview Application Framework in future releases. Note that this is only a recommendation and not an actual requirement of the command.

- [ ] designates the variable is to be created as a list.
- STRING designates a string value.
- NUM designates a numeric value.

- RANGE designates a list of symbols.
- OBJECT designates an object instance.

Syntax example:

```
CREATE VARIABLE rSymbols AS RANGE
SET VARIABLE rSymbols = "SYMBOL1 | SYMBOL2 | SYMBOL3 | SYMBOL4"
// Create a variable of type object
CREATE VARIABLE TestObject as Object
```

Syntax Example - For Retrieving A Symbol's Balance Type

```
CREATE VARIABLE BALTYPE AS STRING
SET VARIABLE BALTYPE = [[SYMBOL, BALANCETYPE, A10021]]
```

Syntax Example - For Retrieving A Symbol's Description

```
CREATE VARIABLE DESC AS STRING
SET VARIABLE DESC = [[SYMBOL, DESCRIPTION, A10021]]
```

See also

- Create GlobalVariable
- ExportVariable
- Set Variable

DataReport

Use this command to run a report on values from the Data Server repository.

Caution: Problems may occur when you use the DataReport command to create very large output files. Do not allow the file to exceed 3 GB.

Syntax:

```
DATAREPORT InputFile OutputFile
```

where:

- InputFile is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If InputFile includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Input Data.txt"
  ```

  For more information, see Creating the ASCII input file for basic database reports and Creating the ASCII input file for intercompany reports.

- OutputFile is a file created by this command, which will be saved in the Longview working folder. It can include a complete or partial folder path in the format C:\...\FileName. If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If OutputFile includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Output Data.out"
  ```

Syntax example:

```
DATAREPORT "C:\Program Files\Longview\KLXPROD\Source.txt" rep1.out
```

Creating the ASCII input file for basic database reports

Use any text editor program or word processor. Specify a descriptive file name. The file must contain the parameters described in the following table. Make sure each parameter appears on a separate line. For parameters with multiple parts, such as Parents, each part must appear on a separate line:

Parameter	Description
Table	To specify the type of data for the report, type one of the following: ■ LEAF — To include leaf data only. ■ PARENT — To include parent data only. ■ LEAF_AND_PARENT — To include leaf and parent data. ■ VALIDATION — To include validation data. ■ CTA — To include data on the net gain or loss resulting from a foreign currency translation.
Table Type	To specify whether to include journal entry data (string data), type one of the following: ■ UNADJUSTING — To include regular data only. ■ BOTH — To include all types.
Number of Dims to be Extracted	Type the number of dimensions that will appear in the output file. If you specify a number of dimensions lower than the actual number of dimensions in the database, by default all data for the other dimensions is exported.
List of DimNums to be Extracted	Type the dimension number of each dimension that will appear in the output file. Each number must appear on a separate line. The number of dimensions you specify must equal the Number of Dims to be Extracted parameter. The order in which you specify each dimension number determines the order in which they appear in the report.
List of Number of Parent Syms in each Dim Extracted	Type the total number of parent symbols contained in each dimension. Each number must appear on a separate line. The number of symbols you specify must equal the Number of Dims to be Extracted parameter.
Parents	Type the combination of parent and leaf symbols for the report. The total number of parent statements must equal the Number of Dims to be Extracted parameter. For more information, see Syntax for symbols.
Expand TIMEPER Dim	To specify how selected time period symbols appear in the report, type one of the following: ■ Y — To display all time period symbols in a single row ■ N — To display each time period symbol in a separate row
Precision	Specifies the number of digits after the decimal.
Delim	Specifies the character used as a delimiter in the output file.
Output FileName	Specifies the name of the output file. This file name must match the one used in the syntax.
	If your text editor program does not automatically add an end-of-file marker, add an empty hard return as the last line of the file.

Syntax for symbols

For details on how to use pound signs (#) in the Parents parameter to specify the number of symbol levels in a hierarchy, refer to the following table:

Format	Description
Parent|##Num	To view all parents down to number levels from Parent.
Parent|##SNum	To view all parents exactly number levels from Parent.
Parent|##+Num	To view all parents down to number levels, including Parent.
Sym|##+SNum	To view all parents exactly number levels, including Parent.
Parent|### or Parent	To view all leaf symbols under Parent.
Parent|###SNum	To view all leaf symbols exactly number levels down from Parent.
Parent|#Num	To view all parent and child symbols down to number levels from Parent, including Parent.
Parent|#SNum	To view all parent and child symbols exactly number levels down from Parent, including Parent.

Caution: When specifying leaf symbols (Dim0Set, for example), you cannot use the Parent parameter. Instead, you must use the syntax LeafSymName|##+S0.

Creating the ASCII input file for intercompany reports

Use any text editor program or word processor. Specify a descriptive file name. The file must contain the parameters described in the following table. Make sure each parameter appears on a separate line. For parameters with multiple parts, such as Restricting Values, each part must appear on a separate line:

Parameter	Description
Report Type {Paper Size {Orientation	Distinguishes this report input file from a regular input file. The syntax for this parameter is: `SPECIAL{PaperSize{Orientation` where: ■ PaperSize is one of LETTER, LEGAL, EXECUTIVE, TABLOID, BSIZE, CSIZE, DSIZE, ESIZE, A3, A4, A5, B4, B5. ■ Orientation is one of PORTRAIT, LANDSCAPE. The default is LETTER {PORTRAIT.

Parameter	Description
Number of Columns to Restrict [x]	An integer specifying the number of columns that will be restricted during the selection on database tables.
Number of Columns to Display [y]	An integer specifying the number of columns that will be displayed in the report output.
Report Output Type	Specify the output file to be produced: ■ FORMATTED: The output file will contain simple formatting (bolding, underlining, headers, footers) when displayed using the Show DataReport command. For more information, see Show DataReport. ■ FIXED_ASCII: The output will be an ASCII file with a fixed width for all columns. ■ CSV: The output will be a comma delimited csv file. The data will be exported without any regional settings applied. **Note:** If the CSV format is specified, the following parameters should be set to FALSE, otherwise unexpected results may occur: ■ Use Commas ■ Calculate Parent Totals ■ Group Flag ■ Suppress Printing Values Multiple Times ■ Print Totals for the Group ■ Display in Report Header If the CSV format is specified, the Report Title should not be defined, otherwise unexpected results may occur.
Report title	Allows the user to specify the title of the report. The backslash character (\ ) in the title string causes the character string of the title to appear on separate lines. The report title must be on a single line in the input file.
Report tables	Specifies tables to select data from. Type ELIM. If one of the display columns or restrict columns is ELIM_ENTITY, data will be selected from klx_ic_header, klx_icl_txn, klx_icc_txn, klx_ict_txn, and klx_ic_trans_elim. Otherwise data will be selected only from klx_ic_header, klx_icl_txn, klx_icc_txn, and klx_ict_txn. For more information, see Valid ELIM columns.

Parameter	Description
Calculate Variance	To specify whether to calculate and print the variance between the base data value and the inner-most group total or the variance between the two amount columns selected in the report. Type one of the following: ■ TRUE — To calculate variance. Use TRUE only when base data is being retrieved or if it is a double select-type report and two amount columns are being displayed. ■ FALSE — To specify no variance calculation.
Threshold Variance for displaying groups	A double value that allows the user to specify a threshold variance amount such that groups with an absolute value less than the threshold are not printed in the report. Use this parameter only if Calculate Variance is TRUE.
Threshold Variance for displaying each row	A double value that allows the user to specify a threshold variance amount such that rows with an absolute value less than the threshold are not printed in the report. Use this parameter only if Calculate Variance is TRUE. The value in this parameter is ignored for investment reports. See Special Investment Report Flag.
Special Investment Report Flag	Identifies the report as a customer-specific Investment report. Type one of the following: ■ TRUE — When the second select on the ELIM tables is done the following criteria will be used: ◦ Contra_Reporter — Reporter ◦ Account — OffsetAccount ■ FALSE — To specify no variance calculation.
Calculate Parent Totals	Type one of the following: ■ TRUE — To calculate and print the parent totals for all leaf data showing in the Report. ■ FALSE — To not calculate parent totals. The outer-most group by column (the first column in the display block that has a Group Flag of TRUE) determines the hierarchy that will be used in the parent total calculation. If there are no restricting values for the outer-most group by column, all roots in the hierarchy are traversed in order to calculate parent totals.
Get CTA Detail data	Type one of the following: ■ TRUE — To retrieve CTA detail data from the klx_ict_txn table. ■ FALSE — To retrieve leaf and calculated data from the klx_icl_txn and klx_icc_txn tables.

Parameter	Description
Column Display Block	The following parameters will be repeated [y] times for each column that will be displayed. The order that the columns are specified is the order they will appear in the output.
Column Name	Type a column name. For valid column names, see Valid ELIM columns.
Column Header Flag	Indicates whether a special column title should be printed for the column in the report. Type one of the following: ■ TRUE — To use the character string specified in the Column Header parameter, instead of using the column name as a header for this column. ■ FALSE — To use the column name as a header.
Column Header	A character string representing the column header that will be displayed for the current column. The character string must appear on one line in the input file. Use this parameter only if Column Header Flag is TRUE.
Column Width Flag	Indicates whether a special column width will be used in the report output. Type one of the following: ■ TRUE — To use the width specified in the Column Width parameter instead of using the default column width. For more information, see Default Column Widths. ■ FALSE — To use the default column width.
Column Width	An integer representing the column width (in number of characters) that will be used to display the current column. Use this parameter only if Column Width Flag is TRUE.
Group Flag	Type one of the following: ■ TRUE — To group the output by this column each time the value in this column changes. ■ FALSE — To specify no grouping by this column. You can create subgroups by designating multiple display columns as group columns. Each time a group or subgroup changes, a total is printed in the report.
Suppress Printing Values Multiple Times	This parameter is only specified if the current display column has the Group Flag parameter specified as TRUE. Type one of the following: ■ TRUE — To print only the value of the column once for each group since it does not change within the group. ■ FALSE — To repeat the value for each row in the group.

Parameter	Description
Print Totals for the Group	This parameter is only specified if the current display column has the Group Flag parameter specified as TRUE. Type one of the following: ■ TRUE — To print all totals at the end of this group. ■ FALSE — To specify no printing of totals for this group.
Display Symbol Description	This parameter is only specified if the current display column holds a symbol index value. Type one of the following: ■ TRUE — To print the symbol description for the symbol value in the column. ■ FALSE — To specify no printing of the symbol description.
Column Restriction Block	
Column Name	Type a valid column name. For valid column names, see Valid ELIM columns.
Display in Report Header	Indicates whether the restricting values listed for this column will appear in the report header. Type one of the following: ■ TRUE — To display this column's restricting values in the header for the report. ■ FALSE — To specify no printing of the column's restricting values in the report header.
Number of Restricting Values [z]	Specify the number of restricting values for this column.
Restricting Values [1 ... z]	Only long, symbol idx, and calculated idx type columns can have restricting values. ■ Long columns: The output will be rows with one of the specified long values. ■ Symbol idx columns: The output will be rows with either the specified restricting symbol or its children. ■ Calculated idx columns: To specify the output as rows with the values from ACC_GR1_COL and either their corresponding offset accounts (if the number of elements in ACC_GR2_COL is 0 and this is flagged as a special investment Report) or the corresponding account in the ACC_GR2_COL (if the number of elements in ACCOUNT_GR2_COL is equal to the number of elements in ACC_GR1_COL). Restricting values for ACC_GR1_COL and ACC_GR2_COL must be leaf symbols with no pattern as above with symbol idx columns.
End of Column Restrict Block	

Parameter	Description
Variance Calculation Method	Type one of the following to indicate the variance calculation method: ■ AMOUNT-AMOUNT_CALC_COL — To calculate the variance by subtracting the amount found in the AMOUNT_CALC_COL column from the amount found in the AMOUNT column. ■ AMOUNT_CALC_COL-AMOUNT — To calculate the variance by subtracting the amount found in the AMOUNT column from the amount found in the AMOUNT_CALC_COL column. ■ AMOUNT-BASE_CALC_COL — To calculate the variance by subtracting the unadjusted amount retrieved from the base tables from the elimination amount found in the AMOUNT column. ■ BASE_CALC_COL-AMOUNT — To calculate the variance by subtracting the elimination amount found in the AMOUNT column from the unadjusted amount retrieved from the base tables. ■ AMOUNT+AMOUNT_CALC_COL — To calculate the variance by adding the amount found in the AMOUNT column to the amount found in the AMOUNT_CALC_COL column. ■ AMOUNT_CALC_COL+AMOUNT — Same as AMOUNT+AMOUNT_CALC_COL. ■ AMOUNT+BASE_CALC_COL — To calculate the variance by adding the elimination amount found in the AMOUNT column to the unadjusted amount retrieved from the base tables. ■ BASE_CALC_COL+AMOUNT — Same as AMOUNT+BASE_CALC_COL. Use this parameter only if: ■ Calculate Variance is TRUE. ■ Report Tables is ELIM. ■ ACC_GR1_COL and ACC_GR2_COL columns appear in the column restriction list, or the BASE_CALC_COL column appears in the column display list. For more information, see Valid ELIM columns.
Use Commas	Type one of the following: ■ TRUE — To display numbers in the report with a comma separating every group of three digits. ■ FALSE — To display numbers in the report without comma separators.

Parameter	Description
Precision	Specify the number of digits to display after the decimal in numbers in the report.
Show Missing Transactions	Type one of the following: ■ TRUE – To show missing matching transactions that do not appear in the klx_ic_transaction table. ■ FALSE – To show only the transactions recorded in the klx_ic_transaction table. Use this parameter only if Special Investment Report Flag is FALSE, Report Tables is ELIM, and both ACC_GR1_COL and ACC_GR2_COL are found in the column restriction block.
	If your text editor program does not automatically add an end-of-file marker, add an empty hard return as the last line of the file.

Valid ELIM columns

Column	Description	Parameter
REPORTER	Entities appear in all intercompany reports. Entities usually appear as the first column of a report, except for Elimination reports. Entities often have all three Group Flag settings set to TRUE.	symbol idx
CONTRA_REPORTER	Offset Entities appear in most intercompany reports, except for reports where the Special Investment Report Flag is TRUE. Offset Entities normally have all three group settings set to FALSE, except for One-sided Reconciliation reports, in which “Group Items” should be set to TRUE.	symbol idx
ACCOUNT	Accounts should normally appear in all types of reports, except for elimination reports on net offset accounts.	symbol idx
OFFSET_ACCOUNT	Offset accounts appear in most reports, except for two-sided reconciliation reports where offset accounts are represented by the account group column.	symbol idx
AMOUNT	The amount normally appears in most types of reports, with the exception of elimination reports on net offset accounts.	double
POSTED_ACC_AMT	This information is not used in typical intercompany reports.	double
POSTED_OFFSET_AMT	This information is usually used only in elimination reports on net offset accounts.	double
ELIM_ENTITY	Elimination entities usually appear only in elimination reports, often as the first column. Elimination entities represent the entities that have been identified as elimination entities for a Longview application. The data in elimination entities comes from automatic elimination journal entries created by the Longview servers.	symbol idx
TIMEPERIOD	The time period dimension	symbol idx

Column	Description	Parameter
FIXED_DIM1	These dimensions are not usually displayed as columns in reports, since they are often restricted to one symbol per dimension. Instead, the symbol name would be included in the header of the report.	symbol idx
FIXED_DIM2		symbol idx
FIXED_DIM3		symbol idx
FIXED_DIM4		symbol idx
FIXED_DIM5		symbol idx
TRANS_ID	A numbering scheme that is internal to the Longview servers.	Long
JE_ID	The Journal Entry ID number for the journal entry that corresponds to the data being displayed. In some cases, the displayed data has been posted directly by the identified journal entries. In other cases, the only data being displayed is schedule and/or base data (i.e. no journal entry data). In those cases, the JE Identifier that appears is the one for the journal entry that eliminates the data being displayed. If a row is showing data that has no related journal entry, the JE Identifier will appear as -1. This feature is rarely used since elimination journal entries are dynamically regenerated by the Longview servers each time the database is recalculated/restated	long
CREATED_TP_IDX	Displays the creation period of the journal entry that corresponds to the data being displayed.	symbol idx
REPORTER_CALC_COL	The Calculated Entities column displays the names of the entities that have invested in those subsidiaries (i.e. they have entered intercompany investment amounts in Longview schedules). This information is used in reports where the special investment report setting is TRUE. These reports display the eliminated stock amounts of subsidiaries.	calculated symbol idx
ACC_GR1_COL	This column is normally used only in two-sided reconciliation reports. Two-sided reconciliation reports are reports where both entities involved in I/C transactions have entered their versions of the transaction in a Longview schedule. A common example would be an I/C Receivables vs Payables Reconciliation report. The account group column in this type of report represents the equivalent of the offset account column in other reports.	calculated symbol idx
ACC_GR2_COL	This column is normally used only in two-sided reconciliation reports. Two-sided reconciliation reports are reports where both entities involved in I/C transactions have entered their versions of the transaction in a Longview schedule. A common example would be an I/C Receivables vs Payables Reconciliation report. The account group column in this type of report represents the equivalent of the offset account column in other reports.	calculated symbol idx

Column	Description	Parameter
AMOUNT_CALC_COL	This information usually represents the offset amount column in two-sided reconciliation reports, or the I/C investment amount column in reports where the Special Investment Report setting is TRUE.	calculated double
VAR_CALC_COL	Variances are normally displayed in Reconciliation reports.	calculated double
BASE_CALC_COL	This column is normally used only in one-sided reconciliation reports. One-sided reconciliation reports are reports in which only one of the two entities involved in an intercompany transaction enters its version of the events in a Longview schedule. The other entity provides only base data in the appropriate intercompany account. The unadjusted data column in this type of report retrieves the base data that represents the offset amount.	calculated double

Default Column Widths

Column Type	Default width (in number of characters)
symbol idx	10
string	20
double	15
long	8
calculated symbol idx	10
calculated double	13

Deactivate User

Use this command to deactivate a user. Deactivated users will not be able to sign on to Longview, however the user's information and settings remain in the database. When the user is Activated, the user can sign on to Longview again.

Syntax (regular)

```
DEACTIVATE USER [DomainName\]UserId
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the name of the user to which you want to activate.

Syntax example:

```
DEACTIVATE USER JSmith
```

Creating An ASCII Input File

To deactivate multiple users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId
```

Syntax (with an ASCII file):

```
DEACTIVATE USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
DEACTIVATE USER @users.txt
MAINTENANCE OFF
```

See also

- Activate User

Delete Attribute

Use this command to delete an Attribute.

An attribute is data used to describe the characteristics of an object in Longview. For example, SWFAdminEMail is an Attribute representing the email address of an Administrator.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Caution: This command cannot be undone. Use with caution.

Syntax regular:

```
DELETE ATTRIBUTE AttrClass AttrName
```

where:

- AttrClass is the Attribute class. Select one of the following:

AttrClass	Description
SYSTEM	Describes the entire system at the highest level. Attributes of this Attribute class specify system-wide characteristics. There is only one object in the SYSTEM Attribute class: DBDEFAULT.
USER	Describes the Attributes of a particular user. Each user is an object in the USER Attribute class.
SYMBOL	Describes the characteristics of individual symbols. Each symbol is an object in the SYMBOL Attribute class.

- AttrName is the name of Attribute to delete.

Syntax example:

```
MAINTENANCE ON
DELETE ATTRIBUTE SYSTEM TESTSYSTEM
MAINTENANCE OFF
```

Creating an ASCII input file

To delete a large number of Attributes, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
AttrClass{AttrName
```

Syntax (with an ASCII file):

```
DELETE ATTRIBUTE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
DELETE ATTRIBUTE @sysatr.asc
MAINTENANCE OFF
```

Delete Category

Use this command to delete a category.

Syntax regular:

```
DELETE CATEGORY CategoryName
```

where:

- CategoryName is the name of the category to delete.

Syntax example:

```
DELETE CATEGORY TestCategory
```

Delete Data

Use this command to clear data from the database. The area to be cleared of data must be locked.

Syntax:

```
DELETE DATA USING dataspec.lvdsp
```

where:

- dataspec.lvdsp is the name of a DataSpec file. For information on DataSpec documents, see Sample DataSpec document.

Syntax example:

```
DELETE DATA USING daMaster.lvdsp
```

See also

- Create Lock

Delete DataArea

Use this command to delete a DataArea from memory and allow the reuse of the same DataArea name within the current procedure.

Note: The Delete DataArea command also deletes any related DataAreas created by the corresponding Create DataArea command.

Syntax:

```
DELETE DATAAREA DataAreaName
```

where:

- DataAreaName is the name of the DataArea to delete.

Syntax example:

```
DELETE DATAAREA daMaster
```

See also

- Create DataArea

Delete DataRole

Use this command to delete a DataRole.

Syntax:

```
DELETE DATAROLE RoleName
```

where:

- RoleName is the name of the DataRole to delete.

Creating an ASCII input file

To delete a large number of DataRoles, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
RoleName
```

Syntax (with an ASCII file):

```
DELETE DATAROLE @ FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
DELETE DATAROLE NorthAmAccounts
```

Delete DataTable

Use this command to delete a DataTable from memory and allow the re-use of the same DataTable name within the current procedure. To delete only the rows in the DataTable, but maintain the DataTable object itself, see Delete DataTableRows.

Syntax:

```
DELETE DATATABLE DataTableName
```

where:

- DataTableName is the name of the DataTable object to delete.

Syntax example:

```
DELETE DATATABLE ExecutiveSalaries
```

See also

- Create DataTable
- Upload (for DataTables)
- Download (for DataTables)

Delete DataTableRows

Use this command to delete rows in a DataTable object. You can delete all rows in or only the rows that meet a specific condition.

By deleting all rows, you can use this to reset the DataTable object before you use the Download (for DataTables) command to refresh the data in the DataTable. You can also use this command, followed by the Upload (for DataTables) command, to clear out the data in the App table that was included in the DataTable object.

To delete the DataTable object, see Delete DataTable.

Syntax:

```
DELETE DATATABLEROWS FROM DataTableName [WHERE "[ConditionalColumn1]Operator
'ConditionValue1' [AND|OR [ConditionalColumn2] Operator'ConditionValue2]"]
```

where:

- DataTableName is the name of the DataTable object in which to delete rows.
- WHERE deletes only the rows that meet the specified condition. If you do not specify a WHERE clause, all rows in the DataTable are deleted.
- ConditionalColumn1 is the column that must contain a specified value for the row in order for the row to be deleted. If a column name contains spaces, you must enclose it in square brackets. For example [Employee Type]. You cannot specify a userlist column as a ConditionalColumn.
- Operator is the relational operator and can be one of the following:

Value	Description
EQ or ==	Equal to.
GE or >=	Greater than or equal to.
GT or >	Greater than.
LE or <=	Less than or equal to.
LT or <	Less than.
NE or !=	Not equal to.

- ConditionValue1 is the value that ConditionalColumn1 must contain in order for the row to be deleted. Values can be numbers or strings, or, in the case of boolean columns, 1 for TRUE and 0 for FALSE. You must enclose string values in single quotes. For example, 'contract'.

Note: You can specify two conditions separated by one And or Or clause.

Syntax example:

```
DELETE DATATABLEROWS FROM ExecutiveSalaries
```

Syntax example:

```
DELETE DATATABLEROWS FROM ExecutiveSalaries WHERE "[Office Location]=
'Sunnydale' OR [Office Location] ='Smallville'"
```

See also

- Create DataTable
- Upload (for DataTables)
- Download (for DataTables)
- Delete DataTable
- Update DataTableRows

Delete File

Use this command to delete a file. This command works with absolute and relative paths.

The allowed paths for the DELETE FILE command are specified by the PERFMITTED_FILEMAINTENANCE_PATH parameter in the lv_af.cfg file, located in the working directory of the lv_af.exe. Files that cannot be deleted are defined by the EXCLUDED_FILEMAINTENANCE_FILES parameter in the same file. These parameters only apply when they are set.

Syntax:

```
DELETE FILE "[Path]\FileName"
```

where:

- path is an optional parameter and can be specified relative to the executing procedure or can be an absolute path.
- FileName is the name of the file that will be deleted.

Syntax example:

```
DELETE FILE "logs\export\export.log"
```

See also

- Copy File
- FileExists
- GetFileProperties
- Move File
- Rename File

Delete Group

Use this command to delete a user group.

Syntax regular:

```
DELETE GROUP GroupName
```

where:

- GroupName is the name of the user group to delete.

Syntax example:

```
DELETE GROUP Administrators
```

Creating an ASCII input file

To delete multiple user groups, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user group:

```
GroupName
```

Syntax (with an ASCII file):

```
DELETE GROUP @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
DELETE GROUP @groups.txt
MAINTENANCE OFF
```

Delete Lock

A lock is a Data Server repository security feature preventing a data intersection from being changed by another user or process until an operation is completed and the data intersection is released.

Use this command to release a lock on data in the Data Server repository. Users with Locks authorization can delete any lock in the system, while users without Locks authorization can delete their own locks only.

If you use this command in conjunction with the UseInputRelatedSchedules command, the original DataArea lock is deleted along with the locks for the related DataAreas for line item details, comments, and attachments.

For information on how this command is affected by input-related schedules, see UseInputRelatedSchedules.

Syntax:

```
DELETE LOCK ALL|USER USING dataspec.lvdsp [method ]["comment"]
```

where:

- ALL deletes all locks within the DataSpec that meet the conditions specified by the method.
- USER deletes all locks owned by the current user that meet the conditions specified by the method.
- dataspec.lvdsp is the name (and extension) of the DataSpec file.
- method can be one of the following:

Value	Description
MATCH	Deletes only locks that match the area defined by the DataSpec exactly. If you do not specify a method, MATCH is used by default.
ALL	Deletes any locks that are contained by the area defined by the DataSpec. Any lock that is only partially contained by the area defined in the DataSpec is not deleted.

- comment specifies the comment to match when deleting the lock.

Syntax example:

```
DELETE LOCK USER USING CopyNI.lvdsp MATCH "Delete Net Income Locks"
```

See also

- Create Lock

Delete Rule

Use this command to delete a rule.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Caution: This command cannot be undone. Use with caution.

Syntax:

```
DELETE RULE RuleID
```

where:

- RuleID is the rule ID of the rule to delete.

Syntax example:

```
APPEND ATTRIBUTE SYSTEM SGPFloatingTimePeriods VALUE DBDEFAULT
ISRAB|ISRPFY|ISPNHR
```

Creating an ASCII input file

To delete a large number of rules, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each rule:

```
RuleName
```

Syntax example:

```
DELETE RULE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax:

```
MAINTENANCE ON
DELETE RULE @rules.txt
MAINTENANCE OFF
```

Delete Schedule

Use this command to delete a schedule.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users.

For more information, see Maintenance or Exclusive.

Caution: This command cannot be undone. Use with caution.

Syntax:

```
MAINTENANCE ON
DELETE SCHEDULE SchedName
MAINTENANCE OFF
```

where:

- SchedName is the name of the schedule to be deleted.

Creating an ASCII input file

To delete a large number of schedules, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each schedule:

```
SchedName
```

Syntax (with an ASCII file):

```
DELETE SCHEDULE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
DELETE SCHEDULE @Scheduleremove.txt
MAINTENANCE OFF
```

Delete Statistics

Use this command to delete the saved statistics and stop the auto-collection process for a specified Collect Statistics command. You must call the Delete Statistics command with same parameters as the ones used in the corresponding Collect Statistics command.

Syntax:

```
DELETE STATISTICS ON DataAreaName DimName
```

where:

- DataAreaName is the name of the DataArea for which to delete statistics.
- DimName is the calculation dimension.

Syntax example:

```
DELETE STATISTICS ON testarea accounts
DELETE STATISTICS ON testarea entities
```

See also

- Collect Statistics
- SkipNoData

Delete Symbol

Use this command to delete a symbol from the Data Server repository.

When you delete symbols, do so from the bottom up. In other words, delete or move each leaf symbol before you delete their parent symbol.

When you delete a symbol, you also need to delete its data. To do this, follow this process:

1. Export.
2. Import system data.
3. Import data.
4. Run restatement.

For more information, see the Longview Application Administrator Guide.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users.

For more information, see Maintenance or Exclusive.

Caution: This command cannot be undone. Use with caution.

Syntax regular:

```
DELETE SYMBOL DimName SymName
```

where:

- DimName is the dimension containing the symbol.
- SymName is the name of the symbol to delete.

Syntax example:

```
MAINTENANCE ON
DELETE SYMBOL ACCOUNTS NewSales
MAINTENANCE OFF
```

Creating an ASCII input file

To delete a large number of symbols from the Data Server repository, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each symbol:

```
SymName
```

Syntax (with an ASCII file):

```
DELETE SYMBOL DimName @FileName
```

where:

- DimName is the dimension containing the symbols.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
DELETE SYMBOL ACCOUNTS @C:\Temp\Test.asc
MAINTENANCE OFF
```

See also

- Create Symbol

Delete User

Use this command to delete a user.

Syntax regular:

```
DELETE USER [DomainName\]UserId
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the ID of the user you want to delete.

Syntax example:

```
DELETE USER JSmith
```

Creating an ASCII input file

To delete multiple users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId
```

Syntax (with an ASCII file):

```
DELETE USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Syntax example:

```
MAINTENANCE ON
DELETE USER @users.txt
MAINTENANCE OFF
```

Disconnect

Use this command to disconnect from the Data Server repository.

Syntax:

```
DISCONNECT
```

See also

- Connect

Download (for DataAreas)

Use this command to retrieve all data for the given DataArea from the database into local memory. A DataArea is created using the CREATE DATAAREA command, and initially consists of an empty structure only. The DOWNLOAD command is then used to populate the DataArea with data from the database.

Note: This command will download only DataAreas to which the user has access.

Syntax:

```
DOWNLOAD DataAreaName [StandardAll|StandardLeafOnly|CTA|LeafData|Validation]
```

where:

- DataAreaName is the name of the DataArea to download.
- StandardAll specifies that all standard data is downloaded. This option is the default for the DOWNLOAD command.
- StandardLeafOnly specifies that only standard leaf data is downloaded.
- CTA specifies that only CTA data is downloaded.
- LeafData specifies that only true leaf data is downloaded. This option differs from StandardLeafOnly in that it does not include calculated data on leaf symbols.
- Validation specifies that the discrepancy calculated by a validation rule is downloaded. You must use the Rule function to specify the server rule associated with this calculation.

Syntax example:

```
DOWNLOAD DataArea1 StandardLeafOnly
```

See also

- Create DataArea
- Rule
- Stream
- Upload (for DataAreas)

Download (for DataTables)

Use this command to retrieve all data for the given DataTable object from the database into local memory. You can create a DataTable using the CREATE DATATABLE command. Because New DataTables initially consists of an empty structure only, you should then use the DOWNLOAD command to populate the DataTable with data from the database.

Note: This command will download only DataTables to which the user has access.

This command does not apply to virtual DataTables.

Syntax:

```
DOWNLOAD DataTable
```

where:

- DataTable is the name of the DataTable object to download.

See also

- Upload (for DataTables)
- Create DataTable

End AdminService

Use this command to close the session to the Data Servers Administration Port.

Syntax:

```
END AdminService
```

See also

- Begin AdminService
- LVMGRL

Exclusive

Use this command to enable or disable data submission by any other user in the system.

When you issue the Exclusive command, other users in the system cannot submit data, but they can still connect and view data. It is particularly useful to issue this command just before deleting symbols.

> **Note:** Although you can use Maintenance mode alone to perform database maintenance in certain situations when other users are actively connected (such as when you are querying and submitting data), you should use Exclusive mode in conjunction with Maintenance mode. For more information, see Selecting Maintenance or Exclusive.

Syntax:

```
EXCLUSIVE ON|OFF
```

where:

Value	Description
ON	To prevent all users from submitting data to the Data Server repository.
OFF	To allow normal access to the Data Server repository.

Syntax Example

```
EXCLUSIVE ON
```

See also

- Maintenance

ExportCommands

Use this command to output many of the commands and functions in Longview Application Framework to a text file. It can be used as a quick way of viewing the available commands and functions and their corresponding syntax.

> **Note:** This command can only be used in a Procedure document. This command outputs only the functions used in a procedure.

Syntax example:

```
EXPORTCOMMANDS "FileName"
```

where:

- FileName is the name of a text file to which the commands will be exported, enclosed in double quotation marks.

Syntax example:

```
EXPORTCOMMANDS "AllSyntax.txt"
```

ExportHierarchy

Use this command to export a hierarchy in a format that can be used as an input file for the SyncHier command. You may use the ExportHierarchy command in conjunction with the SynchHier command to migrate hierarchy structures from one Longview environment to another Longview environment.

Note: You must have read access to the hierarchy you wish to export.

Syntax:

```
EXPORTHIERARCHY DimensionName Symbol FileName"
```

where:

- DimensionName is a dimension containing the symbols of the hierarchy to be exported.
- Symbol is the relative root symbol of the hierarchy to export.
- FileName is the name of the file the data is exported to. If FileName includes spaces, enclose it in double quotation marks; for example:

  "C:\My Documents\Hierarchy.txt"

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

Syntax example:

```
ExportHierarchy TimePeriods AYR LV_TimePeriods.txt
```

An example of this hierarchy export may look similar to the following:

Example:

```
{AYR{{Actuals Current Year{Standard{Auto{Manual{Neither
AYR{AYR_Q1{+{Actuals Q1{Standard{Auto{Manual{Neither{1
AYR_Q1{AYR_01{+{Actuals Jan{Standard{Auto{Manual{Neither{1
AYR_Q1{AYR_02{+{Actuals Feb{Standard{Auto{Manual{Neither{2
AYR_Q1{AYR_03{+{Actuals Mar{CarryForward{Auto{Manual{Neither{3
```

ExportProperty

Use this command to export a specified property of variables of object type to an external text file.

Syntax:

```
EXPORTPROPERTY ObjectVarName.Property Filename [APPEND]
```

where:

- ObjectVarName is the object.
- Property is the property of the object specified to export.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Data.txt"

  Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

- APPEND directs the command to place the variable's value in a new line of an existing text file.

If the named text file does not exist, it is automatically created, even if APPEND is part of the command.

Caution: If the named text file already exists and APPEND is not designated, the existing text file will be overwritten and any data stored in it will be lost.

Syntax example:

```
EXPORTPROPERTY TestObject.comments ExportObjectProperty.txt
```

Syntax example:

```
EXPORTPROPERTY TestObject.anyNumber ExportObjectProperty.txt APPEND
```

ExportVariable

Use this command to export the values of variables to an external text file.

Caution: The names of variables in Longview Application Framework to be used in data event triggers are limited to 10 characters. The convention is E_DIMNAME (where DIMNAME is the first 8 characters of the dimension name). In the case where the first 8 characters of the dimension names are the same, the variables are named E_DIMNAME_N (where DIMNAME is the first 6 characters of the dimension, and N is the dimension number).

Syntax:

```
EXPORTVARIABLE VariableName FileName [APPEND]
```

where:

- VariableName is the name of a particular variable.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks;

  for example: @"C:\My Documents\My Data.txt"

  Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

- APPEND directs the command to place the variable's value in a new line of an existing text file.

If the named text file does not exist, it is automatically created, even if APPEND is part of the command.

Caution: If the named text file already exists and APPEND is not designated, the existing text file will be overwritten and any data stored in it will be lost.

Syntax example:

```
CREATE VARIABLE X AS STRING
CREATE VARIABLE Y AS NUM
SET VARIABLE X = "YEAR"
SET VARIABLE Y = 2007
EXPORTVARIABLE X Variables.txt
EXPORTVARIABLE Y Variables.txt APPEND
```

See also

- Create Variable
- Set Variable

Fetch

Use this command, in conjunction with the Open Cursor and Close Cursor commands, to retrieve data from a DataTable object and store it in a variable.

This command can be useful when you want to move existing data from a DataTable object back to a hierarchical DataArea. If you want to move existing data from a hierarchical DataArea to a flat DataTable, use the Insert DataTableRow command.

You can use the LVS_FetchStatus system variable to verify the status of the Fetch command. The LVS_FetchStatus returns the following values based on the success of the Fetch command:

- 0 — The Fetch command was successful, and a string is stored in the relevant variable.
- -1 — The Fetch command was unsuccessful (the cursor moved past the result).
- -2 — The Fetch command was unsuccessful (the result is marked as deleted or ignored).

For more information, see Using the LVS_FETCHSTATUS system variable.

Syntax:

```
Fetch Row DATATABLEROW FROM MyCursor INTO MyString
```

where:

- Row is the row to fetch in relation to the cursor position and can be one of the following:

Value	Description
NEXT	Fetches the DataTable row immediately proceeding the cursor position.
PRIOR	Fetches the DataTable row immediately preceding the cursor position.
FIRST	Fetches the first DataTable row.
LAST	Fetches the last DataTable row.

- MyCursor is the name for the Cursor object.
- MyString is the string variable in which to store the retrieved data.

Syntax:

```
// Loops through rows of DataTable
 // If employee is Active status, then will add salary to nExemptTotal or
nNonExemptTotal


// Variables to store Totals for Exempt and NonExempt employees
Create Variable nExemptTotal AS NUM
Create Variable nNonExemptTotal AS NUM
```

```
Create VARIABLE sMsg AS STRING


// Create DataTable
Create DATATABLE dtSalary using "Salary.lvdtd"


OPEN Cursor salaryCursor Select "ID, Name, Position, Salary, Category,
Entity, Active" FROM dtSalary
Fetch FIRST DATATABLEROW from amounts into amountsRow


While $LVS_FetchStatus$ != -1
If $LVS_FetchStatus$ EQ 0
If ( ($MyTableRow[7]$) AND ("$MyTableRow[5]$" == "Non-Exempt"))
Set Variable nNonExemptTotal = $nNonExemptTotal$ + $MyTableRow[4]$
End If
If ( ($MyTableRow[7]$) AND ("$MyTableRow[5]$" == "Exempt"))
Set Variable nExemptTotal = $nExemptTotal$ + $MyTableRow[4]$
End If
End If


Fetch Next DataTableRow from salaryCursor Into MyTableRow
End While


Close Cursor salaryCursor
SET VARIABLE sMsg = "Total for Active Non-Exempt Employees: " + NUMTOSTR
($nNonExemptTotal$)
Show Message $sMsg$
SET VARIABLE sMsg = "Total for Active Exempt Employees: " + NUMTOSTR
($nExemptTotal$)
Show Message $sMsg$
//Transfer nonexempt and exempt totals to an existing dataArea
RUN MODEL transferTotals.lvmod on daTotSalary
```

See also

- Insert DataTableRow

For Each

Use this command to iterate through the values in a list variable.

Syntax:

```
FOR EACH VariableName1 IN VariableName2... NEXT
```

where:

- VariableName1 is the name of a variable that has not yet been declared and whose type is based on VariableName2.
- VariableName2 is the name of a list variable.

Syntax example:

```
CREATE VARIABLE ItemList[] AS STRING
SET VARIABLE ItemList = CreateList("Users")
FOR EACH Item IN ItemList
<commands to execute>
NEXT
```

FTP Put

Use this command to copy a file to a server. This command is used in conjunction with the SET FTP command.

Syntax:

```
FTP PUT "LocalFileName" "RemoteFileName"
```

where:

- LocalFileName is the name of the local file to be copied.
- RemoteFileName is the name to use on the remote server.

Syntax Example

```
FTP PUT "FTP1.txt" "FTP2.txt"
```

See also

- Set FTP

Get AppInfo

Use this command to retrieve group and category assignments for published apps.

Syntax:

```
GET APPINFO Group|All AppName|All @GroupAssignmentFile
@CategoryAssignmentFile
```

where:

- Group is the user group for which you want to retrieve published Longview Apps, or All for all user groups.
- AppName is the name of the Longview App for which you want to retrieve category assignments, or All for all Longview Apps. You don't have to include the .lvapp extension in the name.
- GroupAssignmentFile is the name for the ASCII file that is created by the system that contains Longview App user group assignments. If you specify a file name that includes spaces, enclose it in double quotation marks. The group assignment file is created with the following syntax:

  ```
  AppName{GROUP{Group
  ```

- CategoryAssignmentFile is the name for the ASCII file that is created by the system that contains Longview App category assignments. If you specify a file name that includes spaces, enclose it in double quotation marks. The category assignment file is created with the following syntax:

  ```
  AppName{CATEGORY{Category
  ```

Syntax example:

```
GET APPINFO Administrators All @groups.txt @categories.txt
GET APPINFO All All @"group assignments.txt" @"category assignments.txt"
```

Get Category

Use this command to create a file containing the contents of the KLX_CATEGORIES table. This command is useful during system migration.

Syntax:

```
Get CATEGORY @FileName
```

where:

- FileName is the name for the created ASCII file, with the extension .cat, containing category data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.cat"
  ```

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
Get TEMPLATE ALL ALL ALL
Get CATEGORY @categories.cat
```

See also

- Get Template
- Put Category

Get Data

Use this command to retrieve data from an outside data source using the Open Business Data Fabric. For more information on how to develop an end-to-end solution using this command, see the Longview Integration Guide.

> **Note:** This command is only available for customers on the ISW Platform and can only be used with a service-to-service account. This command also requires the system attribute SAccessToken to be set to an API Key for a user in your Organization's ISW platform.

Syntax:

```
Get DATA BusinessViewId ["Parameters"] TO FileName
```

where:

- BusinessViewId is the unique identifier of the business view from which you will retrieve data.
- Parameters is a list of parameters, such as a filter, you want to apply to the business view. For more information on the types of parameters that can be used and how to use them, see the Longview Integration Guide.
- FileName is the name of the file that will be created containing the data retrieved and saved in oData format.

Syntax example:

```
Get DATA s_64ccbd3793655z760rtn7fc3 "filter=accounting_period_name eq 'Feb
2019'" TO "s_64ccbd3793655z760rtn7fc3.txt"
```

Get DataAuditTrail

Use this command to retrieve data audit trail information to a file or an object list. You can retrieve specific data audit trail information by using a DataSpec document to filter the results retrieved.

Note: Using the Data Audit Trail functionality requires that the Data Audit Trail parameter be set to TRUE in Longview Server Manager (the default value is FALSE). For more information, see the Longview Server Manager Guide. If you do not have access to this functionality, contact your System Administrator.

Syntax:

```
Get DATAAUDITTRAIL TO FileName|ObjectListName [USING dataSpecName]
```

where:

- FileName is the name for the created ASCII file, containing the data audit trail information in comma delimited format. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\Info.csv"

- ObjectListName is the name of the object list variable, containing the data audit trail information.

- dataSpecName is the name of the DataSpec document to filter the result. This parameter is optional. If you do not specify a DataSpecName, all data audit trail data will be retrieved.

Note: There are specific DataSpec functions that can be used to filter the DataAuditTrail results: BatchID, DataChangedThreshold, DateFilter, Dimension, Schedule, and UserFilter. See Using DataSpec functions

Syntax example:

```
Get DATAAUDITTRAIL TO AuditData.csv USING MyDataSpec.lvdsp
```

Sample DataSpec file for Get DataAuditTrail:

```
ACCOUNTS SalesT#99
TIMEPERCP#99
SCHEDULE ICStandard
CONTRAS Boston

USERFILTER user888
```

```
DATEFILTER 2018-09-05, TO, 2018-09-30
DATACHANGEDTHRESHOLD BETWEEN, 10.1, 335.5
BATCHID 5, to, 9
```

Sample output file to CSV:

	A	B	C	D	R	S	T	U	V	W	X	Y	Z
1	Batch ID	ACCOUNT	TIMEPER	ENTITIES	Schedule	Contras	Old Value	New Valu	Delta	Date	User	Comments	
2	5	SALES9	P01YTD	DIM2SET	ICStandar	BOSTON	0	44	44	2018-09-05 16:21	USER888	Data submission	
3	5	SALES9	P02YTD	DIM2SET	ICStandar	BOSTON	0	55	55	2018-09-05 16:21	USER888	Data submission	
4	5	SALES9	P03YTD	DIM2SET	ICStandar	BOSTON	0	66	66	2018-09-05 16:21	USER888	Data submission	
5	6	SALES9	P01YTD	DIM2SET	ICStandar	BOSTON	44	55	11	2018-09-05 16:22	USER888	Data submission	
6	6	SALES9	P02YTD	DIM2SET	ICStandar	BOSTON	55	66	11	2018-09-05 16:22	USER888	Data submission	
7	6	SALES9	P03YTD	DIM2SET	ICStandar	BOSTON	66	77	11	2018-09-05 16:22	USER888	Data submission	
8	7	SALES9	P01YTD	DIM2SET	ICStandar	BOSTON	55	66	11	2018-09-05 16:22	USER888	Data submission	

v26.2 Get EffectiveAuthorization

Use this command to retrieve the effective authorizations assigned to users and groups in the system.

The command creates a CSV file containing the following columns: User Name, Group Name, Authorization Name, Authorization Description, and Restricted to Group. The output reflects the resolved authorizations for each user and group combination. Users who do not belong to a group or the group specified are not included in the output.

Syntax:

```
GET EFFECTIVEAUTHORIZATION UserName|All GroupName|All TO FileName
```

where:

- UserName is the user name or All for all users.
- Group is the group name or All for all groups.
- FileName is the name for the created file, containing the effective authorization information in comma delimited format (CSV). It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks. For example “C:\My Documents\Authorizations.csv”

Syntax example:

```
GET EFFECTIVEAUTHORIZATION "JohnDoe" "Analyst" TO "EffectiveResult.csv"
```

Get JEDetails

Use this command to retrieve journal entry details information to a file or an object list. You can retrieve specific journal entry details by using a JEQuerySpec document to filter the results retrieved.

Note: Whether you choose to use a JEQuerySpec document or not, only journal entry details that you have access to are retrieved. Therefore, only the details for shared journal entries and journal entries which you have created, will be retrieved.

Syntax:

```
Get JEDETAILS TO FileName1|ObjectListName1 FileName2|ObjectListName2 [USING
JEQuerySpecName]
```

where:

- FileName1 is the name for the created ASCII file, containing the journal entry details data for base dimensions in comma delimited format. It can include a complete or partial folder path in the format C:\...\FileName1. If FileName1 includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\JEDetails.csv"

- FileName2 is the name for the created ASCII file, containing the journal entry details data for schedule dimensions in comma delimited format. It can include a complete or partial folder path in the format C:\...\FileName2. If FileName2 includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\JESchedDetails.csv"

- ObjectListName1 is the name of the object list variable, containing the journal entry details data.

- ObjectListName2 is the name of the object list variable, containing the journal entry detailsdata for schedule dimensions .

- JEQuerySpecName is the name of the JEQuerySpec document to filter the result. This parameter is optional. If you do not specify a JEQuerySpecName, all journal entry headers, that you have access to, will be retrieved.

Syntax:

```
Get JEDETAILS TO JEDetails.csv JESchedDetails.csv USING MyJEQuerySpec.lvqje
```

Sample output file to CSV:

	A	B	C	D	E	F	G	H	L	M	V	W
1	TYPE	JE_ID	CREATION	CATEGORY	LINE_NU	ACCOUNTS	TIMEPER	ENTITIES	CURRENCY	CONTROLS	BALANCET	VALUE
2	LEAF	1	P03YTD	Elimination	1	APINTCO	P03YTD	CANELIM	DIM6SET	DIM7SET	CREDIT	-200
3	LEAF	1	P03YTD	Elimination	2	SUSPENSE1	P03YTD	CANELIM	DIM6SET	DIM7SET	DEBIT	-200
4	LEAF	1	P03YTD	Elimination	3	APINTCO	P03YTD	CORPWEL	DIM6SET	DIM7SET	CREDIT	-200
5	LEAF	1	P03YTD	Elimination	4	SUSPENSE1	P03YTD	CORPWEL	DIM6SET	DIM7SET	DEBIT	-200
6	LEAF	2	P03YTD	Elimination	1	APOTHER	P03YTD	CANELIM	DIM6SET	DIM7SET	CREDIT	-300

Get JEHeaders

Use this command to retrieve journal entry header information to a file or an object list. You can retrieve specific journal entry headers by using a JEQuerySpec document to filter the results retrieved.

> **Note:** Whether you choose to use a JEQuerySpec document or not, only journal entry headers that you have access to are retrieved. Therefore, only the headers for shared journal entries and journal entries which you have created, will be retrieved.

Syntax:

```
Get JEHEADERS TO FileName|ObjectListName [USING JEQuerySpecName]
```

where:

- FileName is the name for the created ASCII file, containing the journal entryheader data in comma delimited format. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\JEHeaders.csv"

- ObjectListName is the name of the object list variable, containing the journal entry header data.

- JEQuerySpecName is the name of the JEQuerySpec document to filter the result. This parameter is optional. If you do not specify a JEQuerySpecName, all journal entry headers, that you have access to, will be retrieved.

Syntax example:

```
Get JEHEADERS TO JEHeaders.csv USING MyJEQuerySpec.lvqje
```

Sample output file to CSV:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	JE_ID	CREATION	CATEGORY	SUB_CATE	POST_PERIOD	TYPE	FINANCIAL	CALCULATED	SHARED	RECURRING	STATUS	APP_ID	CREATOR	SUMMARY_E	SUMMARY	CREATED	LAST_MODIFIED
2	1	P03YTD	Elimination	1	P03YTD	CurrentPeriod	TRUE	FALSE	TRUE	FALSE	Posted		KARENDEFAU	Entity: EDMONTON	Offset Er	2018-08-13 13:39	2018-08-13 13:39
3	2	P03YTD	Elimination	4	P03YTD	CurrentPeriod	TRUE	FALSE	TRUE	FALSE	Posted		KARENDEFAU	Entity: EDMONTON	Offset Er	2018-08-13 13:39	2018-08-13 13:39

Get Log

Use this command to download log files, such as a file from the Data Server repository.

The Longview Data Server Log file, lv_dataserver.log, records a wide variety of activities in the Longview servers.

Syntax:

```
GET LOG MONITOR
```

See also

- AuditTrail

Get Mappings

Use this command to retrieve one or all mappings in the LV_SYM_MAPPINGS table to a comma delimited csv file. This command is useful during system migration.

Mappings are output in the following format:

```
SymbolName, MapMethod, Expression
```

where:

- MapMethod is Exact|Range|Wildcard
- Expression is enclosed in double quotes.

The naming convention of the mapping files that are output are in the following format: MapName>.csv

Syntax:

```
GET MAPPINGS Map|All [TargetFolder]
```

where:

- Map is the name of the map or All Maps.
- TargetFolder is the folder in which the specified Mapping or all Mappings will be stored.

The TargetFolder may include a path; if no path is specified, the current default working folder is used. If there are spaces in the file name or file path, enclose the path or file name in double quotation marks.

Syntax example:

```
GET MAPPINGS "MyAccountMapping"
GET MAPPINGS All "C:\temp\Mappings"
```

Get Maps

Use this command to create a file containing the contents of the LV_SYM_MAPS table. The LV_SYM_MAPS table contains Internal Maps of the system. This command is useful during system migration.

Syntax:

```
GET MAPS @FileName.ext
```

where:

- FileName.ext is the name for the created ASCII file, containing map metadata. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\My Maps.txt"

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
Get MAPS @Maps.txt
```

Get MetadataAuditTrail

Use this command to retrieve metadata audit trail information to a file or an object list. You can retrieve specific metadata audit trail information by using a AuditSpec document to filter the results retrieved.

Note: Using the Metadata Audit Trail functionality requires that the Metadata Audit Trail parameter be set to TRUE in Longview Server Manager (the default value is FALSE). For more information, see the Longview Server Manager Guide. If you do not have access to this functionality, contact your System Administrator.

Syntax:

```
Get METADATAAUDITTRAIL TO FileName|ObjectListName [USING auditspecName]
```

where:

- FileName is the name for the created ASCII file, containing the metadata audit trail information in comma delimited format. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  @"C:\My Documents\Info.csv"

- ObjectListName is the name of the object list variable, containing the metadata audit trail information.

- auditSpecName is the name of the AuditSpec document to filter the result. This parameter is optional. If you do not specify an auditSpecName, all metadata audit trail data will be retrieved.

Note: There are specific AuditSpec functions that can be used to filter the DataAuditTrail results. See Using DataSpec functions.

Note: Depending on your user and/or group authorizations, you may not be able to use this command, or you may only be able to retrieve non-security related metadata activity.

Syntax example:

```
Get METADATAAUDITTRAIL TO MetaDataAudit.csv USING MyAuditSpec.lvaud
```

Syntax Example

```
USERFILTER user888
DATEFILTER 2020-01-01, TO, 2020-01-31
CATEGORYFILTER USER|GROUP|GROUPMEMBERSHIP|SYMBOL
```

Get ProcessMapInfo

Use this command to retrieve group and category assignments for published process maps.

Syntax:

```
GET PROCESSMAPINFO Group|All ProcessID|All @GroupAssignmentFile
@CategoryAssignmentFile
```

where:

- Group is the user group for which you want to retrieve published Longview Process Maps, or All for all user groups.
- ProcessID is the ID of the Longview Process Map for which you want to retrieve category assignments, or All for all Longview Process Maps.
- GroupAssignmentFile is the name for the ASCII file that is created by the system that contains Longview Process Maps user group assignments. If you specify a file name that includes spaces, enclose it in double quotation marks. The group assignment file is created with the following syntax:

  ```
  ProcessID{GROUP{Group
  ```

- CategoryAssignmentFile is the name for the ASCII file that is created by the system that contains Longview Process Map category assignments. If you specify a file name that includes spaces, enclose it in double quotation marks. The category assignment file is created with the following syntax:

  ```
  ProcessID{CATEGORY{Category
  ```

Syntax:

```
GET PROCESSMAPINFO Administrators All @groups.txt @categories.txt
```

Syntax example:

```
GET PROCESSMAPINFO All All @"group assignments.txt" @"category
assignments.txt"
```

Get Template

Use this command to publish and unpublish templates without manual user intervention for each template.

You can use the Get Template commands (Get Template and Get TemplateInfo) in conjunction with the PublishTemplate command to migrate templates from one environment to another.

Publishing or unpublishing allocation patterns must be defined to all user groups.

Syntax:

```
GET TEMPLATE Group|All TemplateType|All FileName|All [TargetFolder]
[@GroupFile] [@CategoryFile]
```

where:

- Group is the user group name or All for all user groups.
- TemplateType can be one of the following:

Value	Description
LINK	To retrieve links.
REPORT	To retrieve report templates.
ALL	To retrieve all template types.

- FileName is the file name of the template (including the file extension) or All files.
- TargetFolder is the folder in which the template or templates will be stored. The TargetFolder may include a path; if no path is specified, the current default working folder is used. If there are spaces in the file name or file path, enclose the path or file name in double quotation marks. You cannot specify a TargetFolder and a GroupFile with different paths, or a GroupFile without a TargetFolder.

  Caution: Ensure that your TargetFolder is not the existing templates folder on the server. This folder is reserved for Longview internal use. Using this folder may result in undesired behavior.

- GroupFile is the file name for the ASCII file containing template user group assignments. If you don't specify a name for the group file, the default is template.idx. The group file has the following syntax:

  ```
  Group{TemplateType{FileName
  ```

- CategoryFile is the file name for the ASCII file containing template category assignments. If you don't specify a name for the category file, the default is template.cat. The category file has the

following syntax:

```
FileName{Category
```

If no TargetFolder, GroupFile, or CategoryFile is specified, all templates will be saved in the working folder as well as the GroupFile (default name template.idx) and CategoryFile (default name template.cat).

Syntax example:

```
GET TEMPLATE All Report All "C:\Program files\Longview\Servers\Reports" @
template.idx @categories.cat
```

Caution: Always use GET TEMPLATE ALL PATTERN ALL and GET TEMPLATE ALL LINK ALL when syncing templates to ensure all patterns and links are migrated properly.

See also

- Get TemplateInfo
- Publish Template

Get TemplateInfo

Use this command to retrieve user group assignments of templates to an ASCII file.

You can use the Get Template commands (Get Template and Get TemplateInfo) in conjunction with the PublishTemplate command to migrate templates from one environment to another.

Publishing or unpublishing allocation patterns must be defined to all user groups.

Syntax:

```
GET TEMPLATEINFO Group|All TemplateType|All FileName|All [@GroupFile ]
[@CategoryFile]
```

where:

- Group is the user group name or All for all user groups.
- TemplateType can be one of the following:

Value	Description
Report	To retrieve report templates.
All	To retrieve all template types.

- FileName is the file name of the template (including the file extension) or All for all templates.
- GroupFile is the file name for the ASCII file containing template user group assignments. If you don't specify a name for the category file, the default is template.idx. The group file has the following syntax:

```
Group{TemplateType{FileName
```

- CategoryFile is the file name for the ASCII file containing template category assignments. If you don't specify a name for the group file, the default is template.cat. The category file has the following syntax:

```
FileName{Category
```

- If no GroupFile is specified, the GroupFile is given the default name (template.idx) and created in the working folder.
- If no GroupFile or CategoryFile is specified, the file is saved in the working folder with the default name (template.idx for group files, template.cat for category files).

Syntax example:

```
GET TEMPLATEINFO All Report All
```

```
GET TEMPLATEINFO All Report All @groups.idx @categories.cat
```

See also

- Exclusive
- Get Template
- Publish Template

Get WebObject (image or icon)

Use this command to retrieve non-system Web images or icons to a folder.

You can use the Get WebObject commands in conjunction with the Put WebObject commands to migrate Web pages, panels, images, and icons from one environment to another. Page and panel information is retrieved to a file. Panel, icon, and image files are also retrieved to a folder. Only non-system pages, panels, images, and icons are retrieved.

Retrieving a page does not retrieve all related information (panels, images, and icons). Similarly, retrieving a panel does not retrieve all related information (images and icons). Web objects must be retrieved separately. You can use the Get WebObject Info command to retrieve user group Web page assignments. You may retrieve more information than you intend to “put” with the Put WebObject command.

When you retrieve panel information, associated image files are not retrieved and must be retrieved separately. You can also retrieve user group and page assignments separately with the Get WebObject Info command. You cannot retrieve a single panel and all its dependencies.

Ensure that the following subfolders exist in the folder for retrieving objects:

- Panels
- Images
- Icons\large
- Icons\small

If these folders do not exist, an error message will instruct you to create them. Unless specified elsewhere, panels, images, and icons are stored in subfolders in the working folder.

Syntax:

```
GET WEBOBJECT Image ObjectName|All [Folder]
GET WEBOBJECT Icon ObjectName|All [Folder]
```

where:

- ObjectName is the name of the image or icon. If you specify All, you will retrieve all files in the Images, Icons\large, and Icons\small folders.
- Folder is the folder in which the data will be stored. If not specified, the working folder is used.

Syntax example:

```
GET WEBOBJECT Image All C:\retrieved_objects
```

See also

- Put WebObject (image or icon)

Get WebObject (page or panel)

Use this command to retrieve non-system Web pages or panels as administered in Longview Dashboard Designer.

You can use the Get WebObject commands in conjunction with the Put WebObject commands to migrate Web pages, panels, images, and icons from one environment (server) to another. Page and panel information is retrieved to a file. Panel, icon, and image files are also retrieved to a folder. Only non-system pages, panels, images and icons are retrieved.

Retrieving a page does not retrieve all related information (panels, images, and icons). Similarly, retrieving a panel does not retrieve all related information (images and icons). Web objects must be retrieved separately. You can use the Get WebObject Info command to retrieve user group Web page assignments. You may retrieve more information than you intend to “put” with the Put WebObject command.

When you retrieve panel information, associated image files are not retrieved and must be retrieved separately. You can also retrieve user group and page assignments separately with the Get WebObject Info command. You cannot retrieve a single panel and all its dependencies.

Ensure that the following subfolders exist in the folder for retrieving objects:

- Panels
- Images
- Icons\large
- Icons\small

If these folders do not exist, an error message will instruct you to create them. Unless specified elsewhere, panels, images, and icons are stored in subfolders in the working folder.

Syntax:

```
GET WEBOBJECT All ObjectName|All @FileName
GET WEBOBJECT Page ObjectName|All @FileName
GET WEBOBJECT Panel ObjectName|All @FileName
```

where:

- ObjectName is the name of the page or panel as defined in Longview Dashboard Designer.
- FileName is the name of the output file to contain the retrieved data. The filename can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you don’t need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Caution: Ensure that the target folder is not the existing Web folder on the server. This folder is reserved for Longview internal use. Using this folder may yield undesired behavior.

Syntax example:

```
GET WEBOBJECT Page Group_1_Home @Grp1Pg.xml
```

See also

- Put WebObject (page or panel)

Get WebObjectInfo

Use this command to retrieve user group Web page assignments for Web objects.

You can use the Get commands (Get WebObject Page, Get WebObject Panel, Get WebObject Image, Get WebObject All, and Get WebObject Info) to retrieve Web pages, panels, images, and user group assignments.

Syntax:

```
GET WEBOBJECTINFO PageName|All @FileName
```

where:

- PageName is the name of the page assignment you want to retrieve. To retrieve user group Web page assignments for all pages, type All.
- FileName is the file in which the data will be stored. The file name can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
GET WEBOBJECTINFO Group_1_Home @Grp1PgAssn.xml
```

See also

- Put WebObjectInfo

Get Workflow

Use this command to retrieve workflow definitions (process and email) to a file. This command can be useful when migrating your system.

Note: When migrating a system that uses Certification, the Certification Definition needs to be redefined in Longview Workflow for the processes using Certification.

You can use the Get Workflow commands (Process, Email, All) in conjunction with the Put Workflow commands (Process, Email, All) to migrate workflow definitions from one environment to another. The Get Workflow commands do not retrieve workflow statuses. Processes that are under construction are not available for retrieval.

Since the Get Workflow Email command is not process-specific, this command retrieves all Workflow email definitions.

Syntax:

```
GET WORKFLOW ALL ALL @OutputFileName
GET WORKFLOW Email ALL @OutputFileName
GET WORKFLOW Process ALL|ProcessName @OutputFileName
```

where:

- OutputFileName is the name of the output file to contain the retrieved data. The file name can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.
- ProcessName is the name of the workflow process in the Longview server.

Syntax:

```
GET WORKFLOW ALL ALL @AllWorkStuff.xml
```

See also

- Put Workflow
- Set WorkflowStatus

Get WorkflowLog

Use this function to retrieve a history log for a specified Workflow process and step. The function will return an object with the following properties:

Property	Description
WFProcessName	String indicating the name of the Workflow Process
WFStepName	String indicating the Workflow step name
DimensionName	String indicating the Approval Dimension, if applicable
SymbolName	String indicating the Symbol Name, for hierarchical steps
History	An object list containing a history of the status changes for the workflow step. The history object contains the following properties: ■ UserID: Username of the user who last changed the state ■ UserDescription: User description of the user who last changed the state ■ Modified: Datetime stamp of the last state change ■ Comment: Comment for the last state change, if applicable ■ CurrentStatus: Current state of the workflow step ■ PreviousStatus: Previous state of the workflow step

The return values for the status properties CurrentStatus and PreviousStatus:

Property	Description
-1	Failed **Note:** Failed indicates the Longview Application Framework function did not execute properly or encountered an issue.
0	Not Started
999	Rejected
1000	In Progress
2000	Submitted for Approval
3000	Approved
9999	No Previous Status Available

Syntax:

```
Get WorkflowLog TO FileName|ObjectListName Process Step[ Symbol]
```

where:

- FileName is the name for the created ASCII file, containing the workflow log data. It can include a complete or partial folder path in the format C:\...\FileName1. If FileName1 includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\WF_Log.json"
  ```

- ObjectListName is the name of the object list variable, containing the workflow log data.
- Process is the name of an active Workflow process. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Symbol is optional for hierarchical steps, and is the symbol for the hierarchical step. Symbol is not used for simple steps.

Syntax example:

```
GET WORKFLOWLOG TO WFLog.json MonthEndProcess "ABC Canada"
```

History

Use this command to record commands executed by a procedure document to a history file.

Syntax:

```
HISTORY ON|DETAIL "FileName"
HISTORY ON|DETAIL
HISTORY TIMERRUN|TIMEROFF
HISTORY TIMERSTART|TIMERSTOP
HISTORY OFF
```

where:

Value	Description
ON	To record commands run from a Procedure document. Your computer may experience delays. The first time you use HISTORY ON or HISTORY DETAIL, you must specify a value for "FileName". For subsequent uses of HISTORY ON or HISTORY DETAIL, "FileName" is optional. If you use HISTORY ON without the "FileName" parameter after a HISTORY OFF command, HISTORY ON resumes recording commands in the history file specified by the last "FileName" parameter.
DETAIL	The DETAIL option can be used instead of the HISTORY ON command. It prompts the system to insert detailed messages useful for diagnostic purposes into the designated history file. DETAIL messages are identifiable by the ">>>" prefix. All DETAIL messages end with a statement of the amount of memory, in kilobytes, consumed by the operation. If HISTORY DETAIL is used in conjunction with the HISTORY TIMERUN command, DETAIL messages are headed by a time stamp indicating when the operation occurred. The first time you use HISTORY ON or HISTORY DETAIL, you must specify a value for "FileName". For subsequent uses of HISTORY ON or HISTORY DETAIL, "FileName" is optional. If you use HISTORY DETAIL without the "FileName" parameter after a HISTORY OFF command, HISTORY DETAIL resumes inserting detailed messages in the history file specified by the last "FileName" parameter.
"FileName"	An ASCII file in which the history details will be saved. If the file name includes spaces, enclose it in double quotation marks; for example: "C:\My Documents\My filename.txt" If the ASCII file does not exist, your system creates it. If the file exists, the history details are added to the end of the existing file.

Value	Description
TIMERRUN	To record the time taken by each command for the rest of the session or until TIMEROFF. It can be used only if preceded by HISTORY ON or HISTORY DETAIL. It writes a time stamp for each command that indicates the total time elapsed since HISTORY TIMERRUN was first called. Once it is called, the time it keeps remains in effect for the rest of the session.
TIMEROFF	To stop recording time taken by each command for the rest of the session.
TIMERSTART	To start recording the time elapsed for a certain series of Events. It can be used only if preceded by HISTORY ON or HISTORY DETAIL. If HISTORY TIMERSTART is called after HISTORY TIMERRUN, the time stamp displayed for each command thereafter is the time elapsed since TIMERSTART was called. Once HISTORY TIMERSTOP is called, all commands thereafter will display the total elapsed time.
TIMERSTOP	To stop recording the time elapsed for a certain series of Events.
OFF	Turns off the recording of commands run from a Procedure document. This is the default.

The following messages are output for the following commands when HISTORY DETAIL is used:

Command	Messages
UPLOAD	■ >>> Total number of values in the DataArea: ■ >>> Total number of values uploaded:
DOWNLOAD	■ ">>> DOWNLOAD peak memory usage (MemKb)" — gives peak memory usage during DOWNLOAD command ■ ">>> Populated DATAAREA with N values (MemKb)" — tells how many values were added to the DATAAREA
RUN MODEL	■ ">>> DATAAREA contains N values (before) (MemKb)" — tells size of DATAAREA before Model execution ■ ">>> Executing on N X N X N X N ... X N = N nodes (MemKb)" — for each CALCULATIONBLOCK, tells size of block (in dimension index order), and total number of nodes on which statements will be executed ■ ">>> (line N) statement (MemKb)" — echoes out each line of the Model prior to execution ■ ">>> ...processed N records... (MemKb)" — reports every 10,000,000 nodes of execution ■ ">>> DATAAREA contains N values (after) (MemKb)" — tells size of DATAAREA after Model execution

Syntax example:

```
HISTORY ON history.txt
```

If...

Use this command to carry out commands conditionally in a Procedure document.

Your system tests whether a particular condition exists or not, and issues instructions based on the result.

- If the value is not zero, it is true, and it carries out Instructions1.

  Note: In models, this command will carry out the instructions for all the intersections in the DataArea if there is data that matches the criteria stated in the command.

- Otherwise, it carries out Instructions2.
- If you omit Instructions2, nothing happens if the value is zero.

You can nest up to ten levels of If statements per Procedure document.

Syntax:

```
IF <Condition>
Instructions1
[ELSE
Instructions2]
END IF
```

where:

- Instructions1 is a set of instructions carried out by your system if the value in the test cell is true (not zero).
- Instructions2 is a optional set of instructions carried out by your system if the value in the test cell is false (zero). If you include Instructions2, use the ELSE statement.

If you omit Instructions2, the command carries out instructions only when the test cell is true.

Syntax example:

```
CREATE VARIABLE Test AS NUM
SET VARIABLE Test = VALUE (Master, A99999, AYR01, WDGTMFG)

If $Test$ GE 100
RUN MODEL DivCalc.mod ON MASTER
End If
```

Caution: The names of variables in Longview Application Framework to be used in data event triggers are limited to 10 characters. The convention is E_DIMNAME (where DIMNAME is the first 8 characters of the dimension name). In the case where the first 8 characters of the

dimension names are the same, the variables are named E_DIMNAME_N (where DIMNAME is the first 6 characters of the dimension, and N is the dimension number).

See also

- While... End While

ImportObject

Use this command to import object properties and values from a JSON-formatted document into a variable of type object.

Syntax example:

```
IMPORTOBJECT ObjectVarName DocName
```

where:

- ObjectVarName is the object type variable to import to.
- DocName is the name for the JSON-formatted document in the document cache and must be unique. Optionally, you can include a file path. All file paths are relative to the document cache. If the path includes spaces, you must enclose it in double quotation marks.

Syntax example:

```
Create Variable TestObject as Object
IMPORTOBJECT TestObject ObjectsExample.json
```

Insert DataTableRow

Once you have created an in-memory DataTable object, you can add a row to it dynamically using this command.

This command can be useful when you want to move existing data from a hierarchical DataArea to a flat DataTable.

You can use the CreateList function with the Populated data option to retrieve data from the DataArea and store it in a variable. You can then use the Insert DataTableRow command to insert the contents of the list variable into a row in a table.

Note: If you're using lists to extend the contents of the column name variable, including adding columns with the userlist data type, you must use the ListAppend function.

If you want to move existing data from a DataTable object back to a hierarchical DataArea, use the Fetch command.

Syntax:

```
Insert DataTableRow Into DataTableName using VariableName
```

where:

- DataTableName is the name of a DataTable object in which to insert the row.
- VariableName is the name of a string variable. The STRING list variable contains a value for each column in the DataTable. Each value in the STRING list variable can be translated to a correct data type according to the column in the DataTable (for example, string or number).

Note: The column order is determined by the ColumnID, in the LV_APPTABLE_COLUMNS table, in ascending order for the particular DataTable.

Syntax example:

```
// Populates DataTable dtSalary with data from dataArea daSalary
// Create and download the dataArea that we want to use as our source
Create DATAAREA daSalary using "Salary.lvdsp"
Create LOCK USER USING "Salary.lvdsp"
Download daSalary StandardAll

// Create the DataTable
Create DATATABLE dtSalary using "Salary.lvdtd"

// Variables to store fields of each record to insert into the DataTable
```

```
Create VARIABLE sField AS STRING
Create VARIABLE ID AS NUM
Create VARIABLE EmpNames[] AS STRING
Create VARIABLE Positions[] AS STRING
Create VARIABLE Salaries[] AS NUM
Create VARIABLE Entity AS STRING
Create VARIABLE Categories[] AS STRING

Set VARIABLE ID = 0
set Variable Entity = E11211

// Extract the data out of the dataArea into lists

// List of employee names from dataArea
Set VARIABLE sField = "Employee_Name"
Set VARIABLE EmpNames = CreateList("POPULATEDDATA", "daSalary", "STRING",
"VALUES", "SalaryFields.lvdsp")
//ExportVariable EmpNames "DataTable2.out" APPEND
// List of employee positions from dataArea
Set VARIABLE sField = "Employee_Position"
Set VARIABLE Positions = CreateList("POPULATEDDATA", "daSalary", "STRING",
"VALUES", "SalaryFields.lvdsp")
//ExportVariable Positions "DataTable2.out" APPEND
// List of salaries from dataArea
Set VARIABLE sField = "AnnualBase"
Set VARIABLE Salaries = CreateList("POPULATEDDATA", "daSalary", "NUM",
"VALUES", "SalaryFields.lvdsp")
//ExportVariable Salaries "DataTable2.out" APPEND

// List of categories from dataArea
Set VARIABLE sField = "Employee_Category"
Set VARIABLE Categories = CreateList("POPULATEDDATA", "daSalary", "STRING",
"VALUES", "SalaryFields.lvdsp")
//ExportVariable Categories "DataTable2.out" APPEND

// Variable to store row information to insert
Create VARIABLE MyTableRow[] AS STRING

// loop through each list and populate MyTableRow
```

```
For EACH sItem in EmpNames

// Clear MyTableRow
Set VARIABLE MyTableRow = ""
Set Variable ID = $ID$ + 1

Set VARIABLE MyTableRow = NumToStr("$ID$")
Set VARIABLE MyTableRow = ListAppend(MyTableRow, "$EmpNames[$ID$]$" )
Set VARIABLE MyTableRow = ListAppend(MyTableRow, "$Positions[$ID$]$" )
Set VARIABLE MyTableRow = ListAppend(MyTableRow, "$Categories[$ID$]$" )
Set VARIABLE MyTableRow = ListAppend(MyTableRow, "$Salaries[$ID$]$" )
Set VARIABLE MyTableRow = ListAppend(MyTableRow, "$Entity$" )

// Insert the row into the table
Insert DataTableRow Into dtSalary using MyTableRow

Next

Show DATATABLE dtSalary using TABLE "Salary.lvtvw"
```

See also

- Fetch

Launch App

Use this command to launch a Longview App from a toolbar icon or a context-sensitive cell action. You can embed the Launch App command in the Command parameter of the Action Data View function. For more information, see Action. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch App command in batch mode.

Syntax:

```
Launch App AppName ["Variable1=Value1[&Variable2=Value2...]"]
```

where:

- AppName is the name of the Longview App to launch without the .lvapp extension. If the Longview App name includes spaces, enclose it in double quotation marks.
- Variable is optional and is the variable to which to pass the Value. You can pass more than one variable and value pair as long as the variable is unique. For information on variables, see Using tokens, variables, and strings.
- Value is the value to pass to the related Variable.

Syntax example:

```
Launch App HelloWorldApp "account=$C_ACCOUNTS$"
```

Launch Component

Use this command to launch a Longview component from a toolbar icon or a context-sensitive cell action. For example, you can embed the Launch Component command in the Command parameter of the Action Data View. For more information, see Action. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch Component command in batch mode.

Syntax:

```
Launch Component ComponentName [“filename”]
```

where:

- Component is the name of the Longview component to launch and can be one of the following:

ComponentName	Component to Launch
AddinOffice	Longview Add-In for Office
ApplicationAdministrator	Longview Application Administrator
AnalysisReporting	Longview Analysis and Reporting
DashboardDesigner	Longview Dashboard Designer
JournalEntries	Longview Journal Entries
ServerManager	Longview Server Manager
WorkflowDesigner	Longview Workflow Designer

- Filename is optional and specifies the file to open when using the Launch Component AddInforOffice command.

Syntax example:

```
Launch Component AddinOffice "My File.xls"
```

Syntax example:

```
Launch Component JournalEntries
```

Launch File

Use this command to launch a file from a toolbar icon or a context-sensitive cell action. For example, you can embed the Launch File command in the Command parameter of the Action Data View function. For more information, see Action. The file opens in a new tab within the client. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch File command in batch mode.

Syntax:

```
Launch File "File Name"
```

where:

- FileName is the name of the file. FileName may include a path. The file will open using the default application set in the client operating system.

Syntax example:

```
Launch File "My File.docx"
Launch File "Z:\Personal\ExcelFiles\MyWorkbook.xlsx"
Launch File "\\ca123\Corp_Shared\QA\My Document.docx"
```

Launch ProcessMap

Use this command to launch a Longview Process Map from a toolbar icon or a context-sensitive cell action. For example, you can embed the Launch ProcessMap command in the Command parameter of the Action Data View function. For more information, see Action. The process map opens in a new tab within the client. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch ProcessMap command in batch mode.

Syntax:

```
Launch ProcessMap ProcessID
```

where:

- ProcessID is the ID of the Longview Process Map. If the Longview Process Map ID includes spaces, enclose it in double quotation marks. Longview Process Map IDs cannot exceed 100 characters.

Syntax example:

```
Launch ProcessMap "TopReports"
```

Launch ReportViewer

Use this command to launch a report in Longview Report Viewer from a toolbar icon or a context-sensitive cell action. For example, you can embed the Launch ReportViewer command in the Command parameter of the Action Data View function. For more information, see Action. The report opens in a new tab within the client. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch ReportViewer command in batch mode.

Syntax:

```
Launch ReportViewer ReportName ["DimName1=SymName1[&DimName2=SymName2...]"]
```

where:

- ReportName is the name of report created in the Longview Analysis and Reporting. If the name of report includes spaces, enclose it in double quotation marks.
- DimNameN is optional and is a dimension to which selected symbol(s) is to be specified for the report.
- SymNameN is optional and is the selected symbol(s) to be used for dimension N for the report. To include multiple selected symbols, separate symbols with a semicolon (;).

Syntax example:

```
Launch ReportViewer "My Report.rtp" TimePeriods=A17_Open;A18_Open
```

Syntax example:

```
Launch ReportViewer "My Accounts.rtp" Accounts=Cash&Currency=CCAD
```

Launch SystemApp

Use this command to launch a Longview System App or Editor from a toolbar icon or a context-sensitive cell action. For example, you can embed the Launch SystemApp command in the Command parameter of the Action Data View function. For more information, see Action. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch SystemApp command in batch mode.

Syntax:

```
Launch SystemApp SystemAppName
```

where:

- SystemAppName is the name of the Longview System App or Editor to launch and can be one of the following:

SystemAppName	System App to Launch	Applies to...
DataLocks	Data Locks	All systems
Events	Events	All systems
EventStatus	Event Status	All systems
Publisher	Apps Publisher	All systems
Users	Users	All systems
UserGroups	User Groups	All systems
UserSubmissions	User Submissions	All systems
BookTax	Book Tax Differences	Tax Provision systems
CurrenciesEditor	Currencies	Tax Provision systems
DeferredTaxSetup	Current and Deferred Tax	Tax Provision systems
InterimAccount	Interim	Tax Provision systems
LossCreditAndOtherAccounts	Loss, Tax Credit, and Other Accounts	Tax Provision systems
Mappings	Mappings	Tax Provision systems
ManageEntities	Entities	Tax Provision systems
NIBT	Net Income Before Tax	Tax Provision systems

SystemAppName	System App to Launch	Applies to...
TaxRollover	Rollover	Tax Provision systems
RolloverAndScenarioSetup	Rollover and Scenario Setup	Tax Provision systems
RunCalc	Tax Calculations	Tax Provision systems
SystemImport	Import	Tax Provision systems
SystemExport	Export	Tax Provision systems
SystemSettingsEditor	System Settings	Tax Provision systems
TARFSetup	Tax Account Rollforward Data Transfer	Tax Provision systems
TARFHierarchy	Tax Account Rollforward	Tax Provision systems
TaskJurisdictionsEditor	Jurisdictions	Tax Provision systems
Regions	Regions	Tax Provision systems
RegionalModifications	Regional Modifications	Tax Provision systems

Note: Some SystemAppName values apply to Longview Tax systems only.

Syntax example:

```
Launch SystemApp BookTax
```

Syntax example:

```
Launch SystemApp DataLocks
```

Launch URL

Use this command to launch a URL link from a toolbar icon or a context-sensitive cell action. You can embed the Launch URL command in the Command parameter of the Action function. Items launched using Launch commands are independent to the .lvapp from which the item is launched. This command is specific to Longview Apps.

Note: You cannot use the Launch URL command in batch mode.

You can use this command in conjunction with the LVS_HTTPPROTOCOL, LVS_IDENTIFIER, LVS_LANGCODE, LVS_WEBBRIDGE, LVS_WEBBRIDGEPATH, and LVS_WEBSERVER system variables to generate a valid URL to any Longview application. For more information, see the following:

- Using the LVS_HTTPPROTOCOL system variable
- Using the LVS_IDENTIFIER system variable
- Using the LVS_LANGCODE system variable
- Using the LVS_SESSIONID system variable
- Using the LVS_WEBBRIDGE system variable
- Using the LVS_WEBBRIDGEPATH system variable
- Using the LVS_WEBSERVER system variable

Syntax:

```
Launch URL "URL [?Variable1=Value1[&Variable2=Value2...]]" [LaunchOption]
```

where:

- URL is the URL link.
- Variable is optional and is the variable to which to pass the Value. You can pass more than one variable and value pair as long as the variable is unique. For information on variables, see Using tokens, variables, and strings.
- Value is the value to pass to the related Variable.

  Note: You can use the optional Variable and Value parameters to pass variables to other Longview applications or as query strings for the URL.

- LaunchOption is optional, specifies the way to open the URL, and is one of the following:

Value	Description
EXTERNAL [" " IE]	where: EXTERNAL specifies that the URL opens in a new browser window. This is the default if LaunchOption is not specified, and is the only applicable option for Longview Smart Client in standalone mode. IE is optional and forces the URL to open externally in Microsoft Internet Explorer. Use this parameter to open any Longview application. If you specify IE, you must precede IE with " ". If you do not specify IE, URLs open in the user's default browser.
INTERNAL ["Title"]	where: INTERNAL specifies that the URL opens in a new tab within the client. This is applicable only to Longview Tax and Longview Performance Manager, and is generally used in preconfigured code. The INTERNAL option should only be used to launch URLs that directly return HTML content, as opposed to launching another application via a URL. Title is optional and is the title for the tab of the launched URL.

Syntax example:

```
Launch URL "http://www.someurl.com" EXTERNAL
```

Syntax example:

```
Launch URL "http://www.someurl.com/search?q=$queryString$" INTERNAL
```

Syntax example:

```
Set Variable sFullURL = "$LVS_HttpProtocol$:/"+"/$LVS_WebServer$" + "$LVS_
WebBridgePath$" + "Longview.ReportViewer.application" +
"?LongviewWebSID=$LVS_SessionID$" + "&LongviewWebBridge=$LVS_WebBridge$" +
"&LongviewIdentifier=$LVS_IDENTIFIER$" + "&LongviewLangCode=$LVS_LANGCODE$"
+ "&LongviewReportName=Controllable Expenses.rtp" +
"&selectedSymbols=ENTITIES:$C_ENTITIES$"

Launch URL "$sFullURL$" EXTERNAL " " IE
```

Load

Use this command to load a document into the document cache. This command can be useful during performance tuning, since documents that are loaded into the document cache do not need to be retrieved from the server each time the document runs. You can load documents such as procedures, models, import specs, export specs, DataArea specs, data table specs, UI files, data views, table views, subroutines, images, and HTML. Any tokens used inside .html files are parsed only once the first time the .html file is used.

Note: During development and testing, use this command with caution, since the copies in the document cache are static and are not updated with any changes you make to the original documents. For more information, see Unload.

Syntax

```
Load [Path\]FileName
```

where:

- Path is the file path for the document. If the path includes spaces, you must enclose it in quotes. For Longview Apps, the default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For Longview Application Framework, the default path is the lvaf folder in the local working directory.
- FileName is the name of the document to load, including the relevant extension.

Syntax example:

```
Load Budgeting\SalaryPlan.lvpro

Load NetInterestIncome.lvdvw
```

See also

- Unload

LoadKar

Use this command to load and expand a .kar file into the document cache. This command stores each object in the .kar file in memory as though using the Load command individually for each object. This command can be useful during performance tuning, since documents that are loaded into the document cache do not need to be retrieved from the server each time the object runs.

You can include procedures, models, import specs, export specs, DataArea specs, data table specs, UI files, data views, table views, subroutines, images, and HTML in a .kar file. Any tokens used inside .html files are parsed only once the first time the .html file is used.

Note: During development and testing, use this command with caution, since the copies in the document cache are static and are not updated with any changes you make to the original documents. For more information, see Unload.

Syntax:

```
LoadKar KarName RootPath
```

where:

- KarName is the path and file name of a .kar file, including the relevant file extension, relative to the applications folder of your Data Server.
- RootPath is the path that is virtually prepended to all files in the .kar archive when the files are extracted.

Syntax example:

```
LoadKar NetInterestIncome.kar CPM\NII\karfiles
```

See also

- Load
- Resource Reset
- Unload

LVMGRL

Use this command to run Longview Server Manager console commands, such as restatements, stop/start, import/export, archive monitor, etc.… This command will allow up to 16 commands to be used in one LVMGRL command or multiple LVMGRL commands can be used.

This command must be used in conjunction with the BEGIN AdminService and END AdminService commands. The output of the LVMGRL commands will be saved in the document specified in the BEGIN AdminService command.

Syntax:

```
LVMGRL "Command1" ["Command2"]…["Command16"]
```

where:

- Command is the Longview Server Manager console command to be ran.

Syntax example:

```
BEGIN AdminService "OutputFile.txt" e3000 9700
LVMGRL "User Admin Admin1" "restate unadjusted" "archive monitor"
END AdminService
```

Syntax example for ISW platform configuration:

```
BEGIN AdminService "OutputFile.txt" https://localhost:8080/client/
LVMGRL "User 4c796c6b-21u5-463b-8ddd-c5839e5c41e7 qPa-pz0D8mgi4zDZRR2E AAD
AzureServer iswb2c" "restate unadjusted" "archive monitor"
END AdminService
```

See also

- Begin AdminService
- End AdminService

Maintenance

Use this command to enable or disable the ability to perform any database maintenance activity by any other user in Longview.

When you are ready to make Database Maintenance changes to the Data Server repository, enable the Maintenance mode. While it is enabled, users can still connect and perform read-only queries. They can also perform locks, journal entry posting, and data submissions. These activities are suspended (not stopped) for the duration it takes the servers to commit and refresh user and symbol changes.

The time required for the servers to commit and refresh can be several seconds to several minutes. The length of time depends on several factors:

- symbol and hierarchy sizes
- use of other features, such as NDD or data translation
- status of the Math queue. When user or symbol changes are committed, the submission queue is suspended and a suspend request is sent to the Math queue. All pending math jobs in the queue prior to this request must complete before the Math Server can be suspended and refreshed. When large math jobs are pending, long wait times may occur.

Selecting Maintenance or Exclusive

Do not confuse the Maintenance command with the Exclusive command:

Value	Description
Maintenance	■ allows users to connect and perform read-only queries ■ allows users to perform locks, journal entry posting, and submissions ■ may result in long wait times for maintenance activities, if many users are working in the application at the same time
Exclusive	■ used in conjunction with Maintenance ■ does not prevent users from connecting or performing read-only queries ■ prevents users from performing locks, journal entry posting, and submissions ■ can be invoked only if no pending submissions are taking place

Although you can use Maintenance alone to perform user and symbol maintenance when users are actively connected, querying, and submitting, in certain situations, you should use Exclusive in conjunction with Maintenance.

- If you make changes to symbols while submissions are in queue, and those symbols are implicated within active submissions, invalid parent symbol totals may occur. In particular, be careful with the following types of symbol changes:

- symbol weights
- parent switching
- virtual status
- symbol deletion
- symbol change from a parent symbol to a child symbol
- symbol change from a child symbol to a parent symbol

- If you are concerned that parent totals may calculate incorrectly, use Exclusive in conjunction with Maintenance, and then perform a full enterprise restatement.

- If you change a user's access from write to read-only while that user has an active submission, the submission may not proceed completely. In this situation, use Exclusive in conjunction with Maintenance.

Syntax:

```
MAINTENANCE StatusType
```

where:

- StatusType can be one of the following:

Value	Description
ON	To allow the use of maintenance commands.
OFF	To stop the use of maintenance commands and refresh the servers.

Syntax example:

```
MAINTENANCE ON

...
MAINTENANCE OFF
```

See also

- Exclusive

For more information on restatements, see the *Longview Application Administrator Guide*.

Move File

Use this command to move an existing file. This command works with absolute and relative paths.

The allowed paths for the MOVE FILE command are specified by the PERMITTED_ FILEMAINTENANCE_PATH parameter in the lv_af.cfg file, located in the working directory of the lv_ af.exe. Files that cannot be moved are defined by the EXCLUDED_FILEMAINTENANCE_FILES parameter in the same file. These parameters only apply when they are set.

Syntax:

```
MOVE FILE FROM "SourceFile" TO "[Destination]\[FileName]" [Overwrite]
```

where:

- SourceFile is the name of the file to be moved. The source file can be specified by its full path or by the name along if it is located in the working directory.
- Destination is optional and is used to specify the location where the file should be moved to. If it's not specified the SourceFile location will be used, unless the SourceFile location is not specified then it will be moved to the current working directory. If Destination is not specified, the FileName must be.
- FileName is optional and is used to name the moved file. If the FileName is not used, the file will be moved with the same name. If FileName is not specified, then the Destination must be.
- Overwrite is optional and specifies whether to overwrite a file that already exists with the same name.

Syntax example:

```
MOVE FILE FROM "logs\export\export.log" TO "logs\export\backup\" OVERWRITE
```

See also

- Copy File
- Create Directory
- Delete File
- FileExists
- GetFileProperties
- Rename File

OnClose

Use this command to specify the procedure to run after a user clicks the window close button (the x in the top right of the window's title bar) in the main window of a Longview Data Grid or HTML window of a Longview App. OnClose behavior does not apply to subwindows.

After the procedure runs, the application terminates unless a called procedure includes the CancelClose command. For more information, see CancelClose.

If you do not specify OnClose behavior, the window close button is disabled. The window close button is also disabled whenever Longview Application Framework is running. For example, the close button is disabled when any Longview Application Framework processes execute as a result of the following in a Data View:

- toolbar actions
- context-sensitive actions
- dynamic models
- the onBlur and onFocus parameters when a user switches tabs

If the OnClose procedure contains an error, and OnError behavior is set to RETURN, TERMINATE, or CONTINUE, the following actions occur:

- the error is written to an error file
- the procedure specified by the OnError command runs (if applicable)
- an error message appears
- the process terminates

If no OnError behavior is specified, an error message appears and then the process is terminated.

Syntax:

```
ONCLOSE "[FolderPath\]Proc.lvpro"
```

where:

- FolderPath is relative to the working directory. You do not need to specify a file path if the procedure resides in the lvaf directory.
- Proc is the name of the procedure to run.

Syntax example:

```
ONCLOSE "Proc.lvpro"
```

OnError

Use this command to specify how the process should proceed after encountering an error.

Syntax:

```
ONERROR Action [ErrorFile [Procedure]]
```

where:

- Action is the desired action to take when the system encounters an error. Action may be one of the following:

Value	Description
CONTINUE	To ignore any errors and continue processing the next item in a file. If you specify the ErrorFile parameter, any errors are reported and written to an error file and the process continues. If you specify both the ErrorFile and Proc parameters, any errors are reported and written to an error file and the specified error procedure runs before the process continues.
TERMINATE	To terminate the process when an error is encountered. If you specify the ErrorFile parameter, any errors are reported and written to an error file and the process terminates. If you specify both the ErrorFile and Proc parameters, any errors are reported and written to an error file and the specified error procedure runs before the process terminates. This is the default setting.
RETURN	To terminate the current procedure and return to the calling procedure. If you specify the ErrorFile parameter, any errors are reported and written to an error file before the current procedure terminates. If you specify both the ErrorFile and Proc parameters, any errors are reported and written to an error file and the specified error procedure runs before the current procedure terminates.

- ErrorFile is the file path of the desired location of the error file. The default error file is lv_afError.log and is written to the directory that contains Longview Application Framework binaries.
- Procedure is an optional parameter and specifies the procedure to execute when an error occurs. ErrorFile must be specified in order to use Procedure.

Note: The same error will not be processed by the same error procedure twice consecutively.

Syntax example:

```
ONERROR CONTINUE errorProcedure C:\LONGVIEW\ERRPROC.LVPRO
```

Open Cursor

Use this command, in conjunction with the Fetch and Close Cursor commands, to define a cursor in a DataTable object, which allows you to move through the various records and then fetch them to a string variable.

Syntax:

```
Open Cursor CursorName Select "Columns" From DataTableName
```

where:

- CursorName is the name for the cursor.
- Columns can be one of the following:
  - * — To navigate the cursor through all columns.
  - Column1, Column2, ... ColumnN — To navigate the cursor through the specified columns, where Column is the column name.
- DataTableName is the name of the DataTable object for which to open the cursor.

Syntax example:

```
Create Variable amountRow[] as String

Create DataTable dtAllSalaries Using "AllSalaries.lvdtd"

Download dtAllSalaries
Open Cursor amounts

Select "Employee ID, Entity, Amount" From dtAllSalaries

Fetch FIRST DATATABLEROW from amounts into amountsRow

While $LVS_FETCHSTATUS$ != -1

If $LVS_FETCHSTATUS$ == 0

//Insert data table row processing here

Run MODEL "TransferTableData.lvmod" ON daSalaries

END If

Fetch NEXT DATATABLEROW FROM amounts INTO amountsRow

END While

Close Cursor amounts
```

OpSys

Use this command to run an executable or a batch program issuing commands to an executable.

For example, OPSYS "\"C:\Program Files\Main files\run.exe\" parameter1 parameter2" resolves to running "C:\Program Files\Main files\run.exe" with parameter1 and parameter2 on the command line.

The error code for the procedure is available to the user by examining the value of OPSYSCODE, which is a variable returned whenever OPSYS is run (a value of 0 indicates that the command ran successfully).

The permitted program paths for the OPSYS command are specified in the lv_af.cfg file found in the working directory of the lv_af.exe. While events are running, the lv_af.exe will look for the lv_af.cfg in the working directory of the Data Server. For example, C:\Longview\LVCPM\DataServers\LVCPM.

The working directory of the Data Server can be specified in the lvsrvr.cfg file under the WORK_ DIRECTORY parameter.

To change the permitted program paths, open the lv_af.cfg file in your preferred text editor, and modify the PERMITTED_OPSYS_PATH statement. There are no path restrictions if the lv_af.cfg file does not exist or if the statement value is blank. Lines in the lv_af.cfg file cannot surpass 4094 characters. Separate multiple paths with a semicolon (;). For example:

```
PERMITTED_OPSYS_PATH = C:\AF\App_Logic;C:\tmp;C:\lv\source
```

Syntax:

```
OPSYS [NOWAIT] "Command"
```

where:

- NOWAIT is an optional parameter that indicates that the procedure should continue during the execution of the OPSYS command. The default is for the procedure to wait until the OPSYS command has completed execution.
- "Command" is the name of the command to execute, including any parameters and, if necessary, the path, enclosed in double quotation marks. For security reasons, the command cannot list a path that contains directory traversal characters (. .). For example, OPSYS "\"C:\temp\..\run.exe\" returns a security violation error.

Note: To include double quotation marks in the character string, precede the interior double quotation marks with a backslash (\ ).

Note: Windows batch file names, and the paths to them, referenced in the command cannot contain spaces.

Syntax example:

```
OPSYS mybatch.bat
```

Profile

Use this command to summarize all Longview Application Framework processes and Smart Client interaction with Application Framework that are run from the time the command is set to ON until it is set to OFF. The summary is output to a text file, delimited with commas. For each command, function, procedure, model, import, export, DataSpec, data query, or Smart Client interaction with Application Framework, the output file indicates how many times it was called and how long it took to process, in seconds.

Note: The PROFILE command does not output values for RUN PROCEDURE commands, however, it does summarize the commands executed by the RUN PROCEDURE commands.

Syntax:

```
PROFILE ON

PROFILE OFF FileName [APPEND]
```

where:

- FileName is the name of a text file that will contain the output data. If the text file does not exist, it is automatically created. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  "C:\My Documents\My Data.txt"

  Note: If the text file is in the same location as lv_af.exe, you do not need to specify the drive or path.

- APPEND is optional and adds the output to the existing text file.

Caution: If the text file already exists and APPEND is not specified, the existing text file is overwritten and any stored data is lost.

Syntax example:

```
PROFILE ON
 PROFILE OFF "C:\My Documents\My Data.txt" APPEND
```

Output syntax:

```
Type,Name,TimesCalled,Duration
...
```

```
ELAPSED,,,TotalDuration
```

where:

- Type can be COMMAND, FUNCTION, PROCEDURE, MODEL, IMPORT, EXPORT, DATASPEC, DATAQUERY, or API.

  Note: API indicates Smart Client interaction with Application Framework.

- Name is the name of the Longview Application Framework process or item as specified by Type.
- TimesCalled is the number of times the process or item was called in all procedures and models executed.
- Duration is the amount of time, in seconds, required to execute the process or item in all procedures and models.
- TotalDuration is the total amount of time, in seconds, between the PROFILE ON and PROFILE OFF commands. This amount of time is approximately equal to the sum of the times for all commands listed in the text file that contains the output data. Times for other item types (such as PROCEDURE or MODEL) are not included in this approximate total, since the Duration time for these items includes the time of the command items that define the item type.

Output example:

```
COMMAND,AUDITTRAIL,25,0.013
...
PROCEDURE,APPFRMWRK\Shared\UpdateStatus\Code\LogStatus.lvpro,25,0.059
...
MODEL,APPFRMWRK\Shared\UpdateEntityStatus\Code\SetInProgress.lvmod,1,0.003
...
DATASPEC,APPFRMWRK\Shared\LoadParameters\Code\LoadParameters.lvdsp,1,0.001
...
API,GetDataAreaUI,5,0.000
...
ELAPSED,,,9.691
```

Publish App

Use this command to publish a Longview App to a user group. You must first create the .lvapp file and place it in the applications folder on the data server. For more information, see Deploying Longview Apps. Use separate lines to publish a Longview App to multiple user groups. You can publish apps to existing user groups only.

For information on creating user groups, see Create Group.

Syntax:

```
PUBLISH APP AppName GROUP Group
```

where:

- AppName is the name of the Longview App to be published to the group. If the Longview App name includes spaces, enclose it in double quotation marks. You don't have to include the .lvapp extension in the name. Longview App names cannot exceed 100 characters.
- Group is the name of the user group to which the Longview App is published.

Syntax example:

```
PUBLISH APP SampleApp GROUP Administrators
```

Syntax (with an ASCII file):

```
PUBLISH APP @FileName
```

where:

- FileName is an existing ASCII file containing the user groups to which the Longview App is published, with the following syntax:

  ```
  AppName{GROUP{Group
  ```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
PUBLISH APP @"Sample App Groups.txt"
```

See also

- Create Group
- Unpublish App

Publish ProcessMap

Use this command to publish a Longview Process Map to a user group. You must first create the process map. Use separate lines to publish a Longview Process Map to multiple user groups. You can publish process maps to existing user groups only. For information on creating user groups, see Create Group.

Syntax:

```
PUBLISH PROCESSMAP ProcessID GROUP Group
```

where:

- ProcessID is the ID of the Longview Process Map to be published to the group. If ProcessID includes spaces, enclose it in double quotation marks. Longview Process Map IDs cannot exceed 100 characters.
- Group is the name of the user group to which the Longview Process Map is published.

Syntax example:

```
PUBLISH PROCESSMAP SampleProcess GROUP Administrators
```

Syntax (with an ASCII file):

```
PUBLISH PROCESSMAP @FileName
```

where:

- FileName is an existing ASCII file containing the user groups to which the Longview Process Map is published, with the following syntax:

  ```
  ProcessID{GROUP{Group
  ```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
PUBLISH PROCESSMAP @"Sample ProcessMap Groups.txt"
```

See also

- Create Group
- Unpublish ProcessMap

Publish Template

Use this command to publish templates to a specific user group or all user groups.

You can use the Publish Template command in conjunction with the Get Template commands (Get Template and Get Template Info) to migrate templates from one environment to another. You can also use this command to publish templates that already exist in the FromFolder.

Publishing or unpublishing allocation patterns must be defined to all user groups.

When you use a specified GroupFile parameter, invalid entries will be reported, but execution of the command will continue.

Syntax:

```
PUBLISH TEMPLATE Group|All TemplateType|All FileName|All [FromFolder]
```

where:

- Group is the user group name or All for all user groups. If you specify ALL for the user group, you must specify Pattern for the TemplateType.
- TemplateType can be one of the following:

Value	Description
LINK	To publish links.
REPORT	To publish report templates.
ALL	To publish all template types.

- FileName is the file name of the template (including the file extension) or All for all files.
- FromFolder is the folder or folders containing the templates and template information, including the CategoryFile and GroupFile. FromFolder can include a complete or partial folder path in the format C:\...\FileName. If the templates and pattern allocation files are in the working folder, you don't need to specify the drive or path. If the folder name includes spaces, enclose it in double quotation marks.

Syntax example:

```
PUBLISH TEMPLATE All Report All "C:\Program files\Longview\Servers\Reports"
```

Caution: Always use PUBLISH TEMPLATE ALL PATTERN ALL and PUBLISH TEMPLATE ALL LINK ALL when synching templates to ensure all patterns and links are migrated properly.

Syntax (with an ASCII file):

```
PUBLISH TEMPLATE @GroupFile @CategoryFile
```

where:

- GroupFile is an existing ASCII file containing template user group assignments, with the following syntax:

```
Group{TemplateType{FileName
```

- CategoryFile is the file name of an existing ASCII file containing template category assignments, with the following syntax:

```
FileName{Category
```

The group file and category file may have been created using the Get TEMPLATE or Get TEMPLATEINFO command.

Syntax example:

```
PUBLISH TEMPLATE @template.idx
```

See also

- Get Template
- Get TemplateInfo
- Set Attribute

Pushdown

Use this command to copy data from a parent symbol to its leaf descendants. Using the Pushdown command, data is processed by looping through a list of accounts and executing a model to transfer data for each account in the list.

Syntax:

```
Calculate PUSHDOWN SourceParents Dimension DataArea [RestrictDimension
Sources Targets]
```

where:

- SourceParents is a range variable, consisting of a list of parent symbols to transfer the data from.
- Dimension is the name of the dimension to perform the calculation on.
- DataArea is the name of the DataArea to perform the calculation on.

  Note: The DataArea must contain only the symbols required for the calculation.

- Sources is a range variable, consisting of a list of source symbols for the restrict dimension.
- Targets is a range variable, consisting of a list of target symbols for the restrict dimension.

The number of symbols in the Targets range variable must match the number of symbols in the Sources range variable.

The following code is an example of how to set up the range variables for Sources and Targets (in the example, there are two restrict dimensions).

Example - how to set up the range variables for Sources and Targets (in the example, there are two restrict dimensions).

Example:

```
Set Variable rSources =

"GrsEOY_ASC|GrsEOYIP_ASC|NetNBOY_ASC|NetNBRBOY_ASC|NetNBOYA_ASC|NetNEOY_
ASC|NetNEOYIP_ASC|NetNEOYCLSSIP_ASC"


Set Variable rTargets =
"tGrsEOY_ASC|tGrsEOYIP_ASC|tNetNBOY_ASC|tNetNBRBOY_ASC|tNetNBOYA_
ASC|tNetNEOY_ASC|tNetNEOYIP_ASC|tNetNEOYCLSSIP_ASC"

For each sEntity in E_ENTITIES


Set Variable sNATCUR = [[SYMBOL, ZGPNativeCurrency, $sEntity$]]
```

Syntax example:

```
Calculate PUSHDOWN rSatCNCP ACCOUNTS dataArea CURRENCY sNATCUR sNATCUR
ELEMENTS rSources rTargets
```

See also

- Set Variable
- UserExists

Put Category

Use this command to update the contents of the KLX_CATEGORIES table using an existing category file. This command can be useful during system migration. You can also use this command to create categories and modify existing categories for Longview Apps.

Syntax:

```
Put CATEGORY @FileName
```

where:

- FileName is the name for the existing ASCII file containing the categories to be created or modified, with the following syntax:

  ```
  CategoryName{CategoryDescription
  ```

It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks.

Syntax Example

```
Put CATEGORY @categories.txt
```

See also

- Get Category
- Publish Template
- Assign App

Put Mappings

Use this command to update the contents of the LV_SYM_MAPPINGS table using an existing mappings comma delimited csv file. This command is useful during system migration.

Note:

- If a target Map is locked in the target system, the process will fail.
- If the Mapping already exists, the Mapping will be replaced by the Mapping in the csv file.
- If there is an invalid Symbol or a syntax error in the mapping file when importing a Mapping, the entire Mapping file will not be uploaded to the database.

Syntax:

```
Put MAPPINGS Map|All [FromFolder]
```

where:

- Map is the name of the map or All Maps
- FromFolder is the folder containing the mappings information.
- FromFolder can include a complete or partial folder path in the format C:\...\Folder. If the mappings files are in the working folder, you don't need to specify the drive or path. If the folder name includes spaces, enclose it in double quotation marks.

Syntax example:

```
Put MAPPINGS ALL "c:\temp\Mappings"
```

Put Maps

Use this command to update the contents of the LV_SYM_MAPS table using an existing maps file. This command can be useful during system migration.

> **Note:** If a target Map is locked in the target system, the process will fail. This command will only upload Maps that do not already exist, it does not overwrite existing Maps.

Syntax:

```
Put MAPS @FileName.ext
```

where:

- FileName.ext is the name for the existing ASCII file containing the maps to be created or modified, with the following syntax:

  ```
  MapName{Schedule{DimName{Tag
  ```

It can include a complete or partial folder path in the format C:\...\FileName.ext. If FileName.ext includes spaces, enclose it in double quotation marks.

```
@"C:\My Documents\My Maps.txt"
```

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
Put MAPS @FileName.ext
```

Put WebObject (image or icon)

Use this command to upload Web image or icon information to the server.

You can use the Put WebObject commands to put Longview Web objects (pages, panels, images, icons) retrieved with the Get WebObject commands from a folder or file to the corresponding tables or folders on another system. When you put pages, all associated panels must exist on the server first. You may put fewer items than are contained in the source file, but you cannot use a Put command with information not contained in the specified input file.

You can use a Put command to replace existing pages, panels, images, or icons.

When you use the Put commands to upload Web objects, do so in the following order:

1. Images and icons
2. Panels
3. Pages
4. Web object information

Syntax:

```
PUT WEBOBJECT IMAGE|ICON ObjectName|All [Folder]
```

where:

- ObjectName is the name of the image or icon. If you use All, you will send all files in the Images, Icons\large, and Icons\small folders. If an object with that name already exists, the command will overwrite it with the new object.
- Folder is the folder containing the data to be sent. If not specified, the working folder is used. If the folder path includes spaces, enclose it in double quotation marks. Always specify the folder that has all the required subfolders created within it.

Syntax example:

```
PUT WEBOBJECT ICON klx.jpg "C:\Program Files\Longview\Web"
```

See also

- Get WebObject (page or panel)

Put WebObject (page or panel)

Use this command to upload Web page or panel information, as administered in Longview Dashboard Designer, to the server.

You can use the Put WebObject commands to put Longview Web objects (pages, panels, images, icons) retrieved with the Get WebObject commands from a folder or file to the corresponding tables or folders on another system. When you put pages, all associated panels must exist on the server first. You may put fewer items than are contained in the source file, but you cannot use a Put command with information not contained in the specified input file.

You can use a Put command to replace existing pages, panels, images, or icons.

When you use the Put commands to upload Web objects, do so in the following order:

1. Images and icons
2. Panels
3. Pages
4. Web object information

Syntax example:

```
PUT WEBOBJECT All ObjectName|All @FileName

PUT WEBOBJECT Page ObjectName|All @FileName

PUT WEBOBJECT Panel ObjectName|All @FileName
```

where:

- ObjectName is the name of the page or panel as defined in Longview Dashboard Designer. If an object with that name already exists, the command will overwrite it with the new object.
- FileName is the name of the source file retrieved previously with the Get WebObject command. The file name can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
PUT WEBOBJECT Page Group_1_Home @Group1pg.xml
```

See also

- Get WebObject (page or panel)

Put WebObjectInfo

Use this command to upload user group Web page assignments to the server.

You can use the Put WebObject Info command to put Longview Web page assignments retrieved with the Get WebObject Info command to the server. When you use Put WebObject Info, all associated pages must exist on the server first.

When you use the Put commands to upload Web objects, do so in the following order:

1. Images and icons
2. Panels
3. Pages
4. Web object information

Syntax:

```
PUT WEBOBJECTINFO PageName|All @FileName
```

where:

- PageName is the name of the Longview Web page whose group assignment you want to upload. To upload page assignments for all groups, type All.
- FileName is the file in which the data is stored. The file name can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
PUT WEBOBJECTINFO Group_1_Home @Grp1PgAssn.xml
```

See also

- Get WebObjectInfo

Put Workflow

Use this command to upload workflow definitions (process and email) contained in an input file to the server. This command can be useful when migrating your system.

Note: When migrating a system that uses Certification, the Certification Definition needs to be redefined in Longview Workflow for the processes using Certification.

You can use the Put Workflow commands (Process, Email, All) in conjunction with the Get Workflow commands (Process, Email, All) to upload previously retrieved workflow definitions from an input file. You may put fewer items than are contained in the source file, but you cannot use a Put Workflow command with information not contained in the specified input file. You cannot upload a workflow process definition to an existing active workflow definition.

Since workflow email is not process specific, this command puts all workflow email definitions.

Syntax:

```
PUT WORKFLOW ALL ALL @InputFileName

PUT WORKFLOW Process ProcessName|ALL @InputFileName

PUT WORKFLOW Email ALL @InputFileName
```

where:

- InputFileName is the name of the input file containing the data retrieved by the Get Workflow command. The file name can include a complete or partial folder path in the format C:\...\FileName. If the document is in the working folder, you do not need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.
- ProcessName is the name of the workflow process in the server. If the process name includes the ampersand (&), left angle bracket (<), or right angle bracket (>) special characters, you must escape the special characters in the following way:

Special character	Escape text to use
left angle bracket <	<
right angle bracket >	>
ampersand &	&

Syntax:

```
PUT WORKFLOW ALL ALL @AllWorkStuff.xml
```

Syntax example:

```
PUT WORKFLOW Process MonthEnd @MonthEndWork.xml
```

Syntax example:

```
PUT WORKFLOW Email ALL @AllWorkEmail.xml
```

See also

- Get Workflow
- Set WorkflowStatus

QuickFormat

Use this command to perform quick, simple changes to ASCII files. The command reads in an ASCII file that the user specifies and generates a new file (or overwrites the original file), filtering out the following characters (as indicated by the command parameters): blanks, double quotation marks, or any control characters (any non-printable character). You can also use the QuickFormat command to replace all occurrences of a string in the ASCII file with another string.

Syntax:

```
QUICKFORMAT FileName [OutputFileName] [Blank] [Quotes] [Control] [Replace
"String1" "String2"]
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

  > **Note:** If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

- OutputFileName is optional and is the name of the output file. If not specified, the original file is overwritten.
- Blank indicates that all spaces in the named file are removed.
- Quotes indicates that all double quotation marks in the named file are removed.
- Control indicates that all control characters in the file named in FileName are removed.
- Replace indicates that all instances of the string specified by "String1" in the file named in FileName are replaced by the value specified by "String2".
- "String1" is a character string to replace, enclosed in double quotation marks.
- "String2" is a character string to replace String1, enclosed in double quotation marks.

Syntax example:

```
QUICKFORMAT "May data.txt" "June data.txt" REPLACE "May" "June"
```

Reconcile Copy

Use this command to copy non-zero data in the Data Server repository from one symbol to another, and to specify that the copy reconciles the target symbol to the source symbol.

Use this command to directly manipulate data in the Data Server repository, based on an intersection of data specified. Reconcile Copy does not overwrite existing non-zero data in the target area unless there is non-zero data in the source area for that data. When copying data from a parent symbol to a child symbol, the data is copied even if the parent symbol is a virtual parent.

If a user has any symbol access that is read-only in the area defined by the lock to which the command pertains, the command is allowed to continue and all read-only symbols are ignored (not copied). If you try to copy to an area that is read-only, an error message appears.

Reconcile Copy differs from the Copy Data command in that it performs the following actions:

Action	Description
If the data does not exist in database...	it submits the data
If the data does exist in the database, and the value is the same...	no submission occurs
If the data does exist in the database, and the value is not the same...	it submits the new value
If the database has a data intersection not referenced in the file...	it submits a zero

Syntax:

```
RECONCILE COPY FROM DimName SourceSymName TO TargetSymName [DataType] USING
dataspec.lvdsp
```

where:

- DimName is a dimension containing the symbols.
- SourceSymName is a symbol whose data you want to copy.
- TargetSymName is a symbol that will contain the copied data.

- DataType can be one of the following:

Value	Description
PARENT	To copy data from (UP + AP) to UL table. The PARENT option does not copy the rollup value of a carryforward parent. If SourceSymName is a virtual parent, the value of the source symbol is recalculated before copying it.
LEAF	To copy data from UL to UL table.
CALCLEAF	To copy data from (UC + UL + AC + AL) to UL table.
CTA	To copy data from (UT, AT) to (UT, AT) table.
ADJUSTEDLEAF	To copy data from (UL + UA) to UL table.

- dataspec.lvdsp is the name of a DataSpec file.

The following table explains the abbreviations in the DataType parameters:

Parameter	Description
U	unadjusted
A	adjusted
C	calculated
L	leaf
P	parent
T	CTA
+	add value for the identical coordinate
,	the corresponding table (for example, unadjusted to unadjusted)

Syntax example:

```
CREATE LOCK USER USING CopyNI.lvdsp "Copy plan income"
RECONCILE COPY FROM TIMEPER A0312 TO P0401 CALCLEAF USING CopyNI.lvdsp
DELETE LOCK USER USING CopyNI MATCH
```

Reconcile Upload

Use this command to submit changed data, including schedule data, to the Longview database.

Business processes often involve submitting files from a single source multiple times within a period. For example, month-end actuals data extracted from the general ledger (potentially large files) could be loaded into the application in several iterations to account for adjusting entries, with the differences between iterations usually very small.

The Reconcile Upload command is almost identical to the Upload command, except that DataArea contents are compared with current database contents, to submit changes only:

Value	Description
If the data does not exist in database...	it submits the data
If the data does exist in the database, and the value is the same...	no submission occurs
If the data does exist in the database, and the value is not the same...	it submits the new value
If the database has a data intersection not referenced in the file...	it submits a zero

Note: The RECONCILE UPLOAD command will only upload data to which the user has write access.

The Reconcile Upload command depends on the existence and status of the lock ID for the DataArea being uploaded as follows:

Value	Description
DataArea has a valid lock ID	Delta values will be uploaded.
DataArea does not have a valid lock ID, and...	one or more locks exist that covers the entire DataArea: All writable data will be uploaded by reconciling the data in the DataArea with the data in the database.
	no lock exists that covers any part of the DataArea: The UPLOAD will fail.

Note: The DataArea must be fully locked for the Reconcile Upload command to run successfully.

Syntax:

```
RECONCILE UPLOAD DataAreaName ["Comment"] [BatchWait DisableServerRollup]
[UploadBatchSize]
```

where:

- DataAreaName is the name of the DataArea.
- Comment is an optional setting equivalent to having a BatchComment command before the UPLOAD command, enclosed in double quotation marks. If not specified, the batch has no comment.
- BatchWait is equivalent to having a BatchWait command before the UPLOAD command and can be one of the following:
  - FALSE — Use this value if you want to exit without waiting for the batch to complete.
  - TRUE — Use this value if you want to wait for the batch to complete.
- DisableServerRollup turns off the server rollup when set to TRUE, and has a default value of FALSE which leaves the server rollup on. This setting is the same as the ServerMath command.
- UploadBatchSize specifies how the upload batch is broken down and transferred to the Data Server. This has no effect on the submission itself, only on performance. The default value is 150000.

Note: If either BatchWait or DisableServerRollup is specified, the other must also be specified.

Syntax example:

```
CREATE DATAAREA daMaster USING Master.lvdsp

CREATE LOCK daMaster USING Master.lvdsp
RECONCILE UPLOAD daMaster "Upload Batch"
```

See also

- BatchComment
- BatchWait
- Upload (for DataAreas)

Refresh Attribute

Use this command to refresh the Attribute values available to the system.

An attribute is data used to describe the characteristics of an object in Longview. For example, SWFAdminEMail is an Attribute representing the email address of an Administrator.

If you have made changes to Attributes, use this command to update your system.

Syntax:

```
REFRESH ATTRIBUTE
```

Syntax example:

```
REFRESH ATTRIBUTE
```

Refresh Dimension

Use this command to refresh all symbol information in a particular dimension.

When you connect to the Data Server repository, the most recent symbol information is automatically downloaded. However, before making changes to the Data Server repository, such as adding a symbol or editing a hierarchy, you probably want to make sure you have the most up-to-date symbol information in the database (if, for example, another user has made hierarchy changes). You can use this command to retrieve the most recent symbol and hierarchy information in a particular dimension.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax:

```
REFRESH DIMENSION DimName
```

where:

- DimName is a dimension.

Syntax example:

```
MAINTENANCE ON
REFRESH DIMENSION ACCOUNTS
MAINTENANCE OFF
```

Remove App

Use this command to remove a Longview App from a category. Use separate lines to remove a Longview App from multiple categories.

Syntax:

```
REMOVE APP AppName CATEGORY [Category]
```

where:

- AppName is the name of the Longview App to be removed from the category. If the Longview App name includes spaces, enclose it in double quotation marks. You don't have to include the .lvapp extension in the name. Longview App names cannot exceed 100 characters.
- Category is optional and is the name of the category from which the Longview App is removed. If no category is specified, the Longview App is removed from all categories. If the category name includes spaces, enclose it in double quotation marks.

Syntax example:

```
REMOVE APP SampleApp CATEGORY Administrators
```

Syntax (with an ASCII file):

```
REMOVE APP @FileName
```

where:

- FileName is an existing ASCII file containing the categories to which the Longview App is assigned, with the following syntax:

  ```
  AppName{CATEGORY{Category
  ```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
REMOVE APP @"Sample App Categories.txt"
```

Remove DataRole

Use this command to remove the database privileges of a DataRole created by a CREATE DATAROLE command.

You can use the following role types in the Remove DatatRole command:

- RESTRICTED — defines access as a subset of the dimension spanning more than one level.
- FIXED — defines access to a single symbol in the dimension.
- FULL — enables write access to all symbols in the dimension.

The parameters to include vary depending on the specified role type.

Syntax regular:

```
REMOVE DATAROLE RoleName ACCESS DimName RESTRICTED SymName [AccessType
NumLevels Priority]

REMOVE DATAROLE RoleName ACCESS DimName FIXED SymName

REMOVE DATAROLE RoleName ACCESS DimName FULL
```

where:

- RoleName is the name of a symbol access role.
- DimName is the dimension containing to which the DataRole is to be given access.
- SymName is the top symbol in the hierarchy from which you want to remove the role access.
- AccessType is a setting that determines whether the role's users have read/write access. This parameter is optional; however, if you specify AccessType you must also specify NumLevels and Priority.
  Select one of the following for the AccessType:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- NumLevels is a number representing the number of hierarchy levels below SymName. This parameter is optional; however, if you specify NumLevels you must also specify AccessType and Priority.
- Priority is a number that designates the symbol's position in the hierarchy relative to its parent. Symbols are listed in order of ascending priority, with zeros falling at the bottom of the list. This parameter is optional; however, if you specify Priority you must also specify AccessType and NumLevels.

Creating an ASCII input file

To remove a large number of DataRoles, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
RoleName{ACCESS{DimName{RESTRICTED{SymName{AccessType[{NumLevels]
```

Syntax (with an ASCII file):

```
REMOVE DATAROLE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
REMOVE DATAROLE NorthAmAccounts
```

Remove Group

Use this command to remove symbol access, authorizations, and users from a user group.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users.

For more information, see Maintenance or Exclusive.

Note: v26.2 For systems on the ISW Platform, when a user belongs to a group with authorizations configured, the user's individual authorizations are ignored and only the group authorizations apply. If the group has no authorizations configured, the user's individual authorizations are used instead. Any group authorizations that exceed what the user's license permits are also ignored.

Syntax:

```
REMOVE GROUP GroupName ACCESS Dimension RoleName [SymbolName [AccessType]]
REMOVE GROUP GroupName ACCESS
REMOVE GROUP GroupName AUTHORIZATION AuthorizationName [AuthorizationGroup]
REMOVE GROUP GroupName AUTHORIZATION
REMOVE GROUP GroupName USER [[DomainName\]UserId]
REMOVE GROUP GroupName USER
```

where:

- GroupName is the name of the group from which you want to remove access, authorizations, or users.
- Dimension is the dimension containing the symbol you want to remove access to.
- RoleName is name of a symbol access role.
- SymbolName is a symbol from which the specified group needs access removed.
- AccessType is a setting that determines whether the user has read/write access. Select one of the following:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- AuthorizationName is a name that designates the type of authorization removed from the group.

Value	Description	Platform License
ADDINFOROFFICE	Access the Longview Add-In for Office. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Longview Add-In for Office.	Designer Power User Viewer
ADDINFOROFFICE SUBMITDATA	Submit data in the Longview Add-In for Office.	Designer Power User
ANALYSISREPORTING AUTHOR	Access Longview Analysis and Reporting as a Report Author.	Power user
ANALYSISREPORTING PUBLISHER	Access Longview Analysis and Reporting as a Report Publisher.	Designer
ANALYSISREPORTING USER	Access Longview Analysis and Reporting as a Report User.	Viewer
APPLICATION ADMINISTRATOR	Access Longview Application Administrator.	Admin Designer Service
ATTRIBUTE	Manage attributes.	Designer
BATCH	Manage batches.	Admin Designer
CONNECTVIA APPLICATION FRAMEWORK	Connect to the server using Longview Application Framework. **Note:** You must assign this authorization to allow users or user groups to access Longview Apps, Longview Designer, Longview tools and editors, the Longview Add-In for Office, and Longview Tax.	Admin Designer Power User Viewer Service
DASHBOARDDESIGNER	Access Longview Dashboard Designer.	Designer

Value	Description	Platform License
DELETECOMMENTS	Delete any existing comments in the Data Server. Users without Delete Comments authorization can delete only their own comments before they are submitted to the database. **Note:** Delete existing comments functionality is available in Data Grids only.	Designer
DESIGNER	Access Longview Designer. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSCREATE	Create data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSDELETE	Delete data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSMODIFY	Edit data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSPUBLISH	Publish data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer

Value	Description	Platform License
DESIGNERLONGVIEWAPPSPUBLISH	Publish Longview Apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
FOREIGNEXCHANGESETTINGS	Manage foreign exchange settings.	Admin Designer
GROUPADMINISTRATION	Perform group administration.	Does not apply
GROUPSYMBOLACCESS	Manage symbol access for groups.	Does not apply
INTERCOMPANYSETTINGS	Manage intercompany settings.	Designer
JOURNALENTRIES	Access Longview Journal Entries.	Designer Power User
JOURNALENTRY	Manage journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODCREATE	Create current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODPERMPOST	Permanently post current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENTPERIODREVIEWPOST	Review post current period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYDELETE	Delete non-shared journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODCREATE	Create future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODPERMPOST	Permanently post future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODREVIEWPOST	Review post future period journal entries.	Designer Power User Service
JOURNALENTRYOWN PERMPOST	Permanently post own journal entries.	Designer Power User Service
JOURNALENTRYOWN REVIEWPOST	Review post own journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJCREATE	Create prior period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYPRIOR PERIODADJPERMPOST	Permanently post prior period journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJREVIEWPOST	Review post prior period journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTCREATE	Create restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTPERMPOST	Permanently post restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTREVIEWPOST	Review post restatement journal entries.	Designer Power User Service
LOCK	Manage locks.	Designer Service
MAPPINGSEDITOR	Access the Mappings editor. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service

Value	Description	Platform License
MAPPINGSMANAGE	Create, modify, delete mappings. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MAPSMANAGE	Create, modify, delete maps. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MODIFYDATA	Submit data.	Designer Power User Service
NDDSETTINGS	Manage NDD settings.	Designer
ROLE	Manage symbol access roles.	Designer Service
RULE	Manage rules.	Designer Service
SCHEDULE	Manage schedules.	Designer Service
SERVERMANAGER	Access Longview Server Manager.	Admin Service Designer
SERVERMANAGERSTART	Start/stop the Longview server.	Admin Service Designer

Value	Description	Platform License
SERVICEACCOUNTUSERADMINISTRATOR	Sets the user to be a User Administrator.	Service
SERVICEACCOUNTRESTAPI	Sets user to be a Service Account User. **Note:** Can only be set using PUSER or a user who has User Administrator Authorization.	Admin Designer Service
SYMBOL	Manage symbols.	Designer Service
SYMBOLASSIGN	Assign symbols to a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLATTRIBUTECREATE	Create symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEDELETE	Delete symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEMODIFY	Modify symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLCANCREATEROOT	Create root symbols. Must be used with SYMBOL, CREATESYMBOL.	Designer Service
SYMBOLCANDELETEROOT	Delete root symbols. Must be used with SYMBOL, SYMBOLDELETE.	Designer Service

Value	Description	Platform License
SYMBOLCREATE	Create symbols. Must be used with SYMBOL.	Designer Service
SYMBOLDELETE	Delete symbols. Must be used with SYMBOL.	Designer Service
SYMBOLREMOVE	Remove symbols from a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLSET	Modify symbols. Must be used with SYMBOL.	Designer Service
SYMBOLSWITCH	Switch symbols in a hierarchy. Must be used with SYMBOL.	Designer Service
SYSTEMATTRIBUTECREATE	Create system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEDELETE	Delete system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEMODIFY	Modify system attributes. Must be used with ATTRIBUTE.	Designer Service
USERADMINISTRATION	Perform user administration	Does not apply
USERATTRIBUTECREATE	Create user attributes. Must be used with ATTRIBUTE.	Designer Service

Value	Description	Platform License
USERATTRIBUTEDELETE	Delete user attributes. Must be used with ATTRIBUTE.	Designer Service
USERATTRIBUTEMODIFY	Modify user attributes. Must be used with ATTRIBUTE.	Designer Service
USERRESETPASSWORD	Reset passwords.	Does not apply
USERSYMBOLACCESS	Manage symbol access for users.	Does not apply
VIEWDATA	View data.	Designer Power User Viewer Service
WORKFLOWDESIGNER	Access Longview Workflow Designer.	Designer

- AuthorizationGroup is required only when AuthorizationName is USERADMINISTRATION or USERRESTPASSWORD, and is the group from which you want the indicated group's authorization removed.

For ACCESS, if the optional AccessType parameter is not included in the syntax, the command removes all access to the specified symbol from the indicated group. If the optional SymName and AccessType parameters are not included in the syntax, the command removes all symbol access for the dimension and role combination from the indicated group. If the syntax contains only "REMOVE GROUP GroupName ACCESS", the command removes all access from the indicated group.

For AUTHORIZATION, if the syntax contains only "REMOVE GROUP GroupName AUTHORIZATION", the command removes all authorizations from the indicated group.

For USER, if the optional DomainName\UserId is not included in the syntax, the command removes all users from the indicated group.

Syntax example:

```
MAINTENANCE ON
REMOVE GROUP NorthAm ACCESS ACCOUNTS AnalystRole Cash W
MAINTENANCE OFF
```

Creating an ASCII input file

To remove the access privileges of a large number of groups, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each group:

```
GroupName{ACCESS{Dimension{RoleName[{SymbolName][{AccessType]
GroupName{ACCESS{Dimension{RoleName
GroupName{AUTHORIZATION{AuthorizationName[{AuthorizationGroup]
GroupName{AUTHORIZATION{AuthorizationName
GroupName{USER{[DomainName\]UserId
GroupName{USER
```

Use the QUICKFORMAT command, if necessary, to make formatting changes to the ASCII file.

Syntax (with an ASCII file):

```
REMOVE GROUP @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON
REMOVE GROUP @Access.txt
MAINTENANCE OFF
```

See also

- QuickFormat

Remove ProcessMap

Use this command to remove a Longview Process Map from a category. Use separate lines to remove a Longview Process Map from multiple categories.

Syntax:

```
REMOVE PROCESSMAP ProcessID CATEGORY [Category]
```

where:

- ProcessID is the ID of the Longview Process Map to be removed from the category. If the Longview Process Map name includes spaces, enclose it in double quotation marks. Longview Process Map IDs cannot exceed 100 characters.
- Category is optional and is the name of the category from which the Longview Process Map is removed. If no category is specified, the Longview Process Map is removed from all categories. If the category name includes spaces, enclose it in double quotation marks.

Syntax example:

```
REMOVE PROCESSMAP SampleProcessMap CATEGORY Administrators
```

Syntax (with an ASCII file):

```
REMOVE PROCESSMAP @FileName
```

where:

- FileName is an existing ASCII file containing the categories to which the Longview Process Map is assigned, with the following syntax:

  ```
  ProcessID{CATEGORY{Category
  ```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
REMOVE PROCESSMAP @"Sample Process Categories.txt"
```

Remove Symbol

Use this command to detach a symbol from its parent symbol in the Data Server repository. You must turn Maintenance on to use this command. When you are finished, turn Maintenance off to allow access by other users. For more information, see Maintenance or Exclusive.

Syntax:

```
REMOVE SYMBOL DimName SymName PARENT OldParentSymName
```

where:

- DimName is a dimension containing the symbols.
- SymName is a symbol to be detached.
- OldParentSymName is the current parent symbol of SymName.

Creating an ASCII input file

Create an ASCII file containing the following information for each symbol:

```
SymName{PARENT{OldParentSymName
```

Syntax (with an ASCII file):

```
REMOVE SYMBOL DimName @FileName
```

where:

- DimName is a dimension containing the symbols.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

REMOVE SYMBOL ACCOUNTS @C:\Temp\Test.asc
```

```
MAINTENANCE OFF
```

See also

- Assign Symbol
- Set Variable
- Switch Symbol

Remove Template

Use this command to remove a template from a category. Use separate lines to remove a template from multiple categories.

Syntax:

```
REMOVE TEMPLATE FileName|All Category|All
```

where:

- FileName is the name of the template to be removed from the category or All for all templates. If the template name includes spaces, enclose it in double quotation marks.
- Category is the category from which the template is removed or All to remove the template from all categories.

Syntax example:

```
REMOVE TEMPLATE SampleTemplate Administrators
```

Syntax (with an ASCII file):

```
REMOVE TEMPLATE @CategoryFile
```

where:

- CategoryFile is an existing ASCII file containing the categories to which the template is assigned, with the following syntax:

  ```
  FileName{Category
  ```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
REMOVE TEMPLATE @"Sample Template Categories.txt"
```

Remove User

Use this command to remove symbol access, authorizations, and group membership from a user.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax regular:

```
REMOVE USER [DomainName\]UserId ACCESS Dimension RoleName [SymbolName
[AccessType]]

REMOVE USER [DomainName\]UserId ACCESS

REMOVE USER [DomainName\]UserId AUTHORIZATION AuthorizationName
[AuthorizationGroup]

REMOVE USER [DomainName\]UserId AUTHORIZATION

REMOVE USER [DomainName\]UserId GROUP [GroupName]
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the ID of the user from which you want to remove access, authorizations, or users.
- Dimension is the dimension containing the symbol you want to remove access to.
- RoleName is name of a symbol access role.
- SymbolName is a symbol from which the specified user needs access removed.
- AccessType is a setting that determines whether the user has read/write access. Select one of the following:

Value	Description
W	Write access, to allow the user to change data.
R	Read-only access, to allow the user to read the data but not change it. This is the default.

- AuthorizationName is a name that designates the type of authorization removed from the group.

Value	Description	Platform License
ADDINFOROFFICE	Access the Longview Add-In for Office. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Longview Add-In for Office.	Designer Power User Viewer
ADDINFOROFFICE SUBMITDATA	Submit data in the Longview Add-In for Office.	Designer Power User
ANALYSISREPORTING AUTHOR	Access Longview Analysis and Reporting as a Report Author.	Power user
ANALYSISREPORTING PUBLISHER	Access Longview Analysis and Reporting as a Report Publisher.	Designer
ANALYSISREPORTING USER	Access Longview Analysis and Reporting as a Report User.	Viewer
APPLICATION ADMINISTRATOR	Access Longview Application Administrator.	Admin Designer Service
ATTRIBUTE	Manage attributes.	Designer
BATCH	Manage batches.	Admin Designer
CONNECTVIA APPLICATION FRAMEWORK	Connect to the server using Longview Application Framework. **Note:** You must assign this authorization to allow users or user groups to access Longview Apps, Longview Designer, Longview tools and editors, the Longview Add-In for Office, and Longview Tax.	Admin Designer Power User Viewer Service
DASHBOARDDESIGNER	Access Longview Dashboard Designer.	Designer

Value	Description	Platform License
DELETECOMMENTS	Delete any existing comments in the Data Server. Users without Delete Comments authorization can delete only their own comments before they are submitted to the database. **Note:** Delete existing comments functionality is available in Data Grids only.	Designer
DESIGNER	Access Longview Designer. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSCREATE	Create data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSDELETE	Delete data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSMODIFY	Edit data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
DESIGNERDATA IMPORTSPUBLISH	Publish data import apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer

Value	Description	Platform License
DESIGNERLONGVIEW APPSPUBLISH	Publish Longview Apps in Longview Designer. **Note:** You must also assign the DESIGNER authorization to allow users or user groups to access Longview Designer.	Designer
FOREIGNEXCHANGE SETTINGS	Manage foreign exchange settings.	Admin Designer
GROUPADMINISTRATION	Perform group administration.	Does not apply
GROUPSYMBOLACCESS	Manage symbol access for groups.	Does not apply
INTERCOMPANYSETTINGS	Manage intercompany settings.	Designer
JOURNALENTRIES	Access Longview Journal Entries.	Designer Power User
JOURNALENTRY	Manage journal entries.	Designer Power User Service
JOURNALENTRYCURRENT PERIODCREATE	Create current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENT PERIODPERMPOST	Permanently post current period journal entries.	Designer Power User Service
JOURNALENTRYCURRENT PERIODREVIEWPOST	Review post current period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYDELETE	Delete non-shared journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODCREATE	Create future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODPERMPOST	Permanently post future period journal entries.	Designer Power User Service
JOURNALENTRYFUTURE PERIODREVIEWPOST	Review post future period journal entries.	Designer Power User Service
JOURNALENTRYOWN PERMPOST	Permanently post own journal entries.	Designer Power User Service
JOURNALENTRYOWN REVIEWPOST	Review post own journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJCREATE	Create prior period journal entries.	Designer Power User Service

Value	Description	Platform License
JOURNALENTRYPRIOR PERIODADJPERMPOST	Permanently post prior period journal entries.	Designer Power User Service
JOURNALENTRYPRIOR PERIODADJREVIEWPOST	Review post prior period journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTCREATE	Create restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTPERMPOST	Permanently post restatement journal entries.	Designer Power User Service
JOURNALENTRYRE STATEMENTREVIEWPOST	Review post restatement journal entries.	Designer Power User Service
LOCK	Manage locks.	Designer Service
MAPPINGSEDITOR	Access the Mappings editor. **Note:** You must also assign the CONNECTVIAAPPLICATIONFRAMEWORK authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service

Value	Description	Platform License
MAPPINGSMANAGE	Create, modify, delete mappings. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MAPSMANAGE	Create, modify, delete maps. **Note:** You must also assign the MAPPINGSEDITOR authorization to allow users or user groups to access the Mappings editor.	Designer Power User Service
MODIFYDATA	Submit data.	Designer Power User Service
NDDSETTINGS	Manage NDD settings.	Designer
ROLE	Manage symbol access roles.	Designer Service
RULE	Manage rules.	Designer Service
SCHEDULE	Manage schedules.	Designer Service
SERVERMANAGER	Access Longview Server Manager.	Admin Service Designer
SERVERMANAGERSTART	Start/stop the Longview server.	Admin Service Designer

Value	Description	Platform License
SERVICEACCOUNTUSERADMINISTRATOR	Sets the user to be a User Administrator.	Service
SERVICEACCOUNTRESTAPI	Sets user to be a Service Account User. **Note:** Can only be set using PUSER or a user who has User Administrator Authorization.	Admin Designer Service
SYMBOL	Manage symbols.	Designer Service
SYMBOLASSIGN	Assign symbols to a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLATTRIBUTECREATE	Create symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEDELETE	Delete symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLATTRIBUTEMODIFY	Modify symbol attributes. Must be used with ATTRIBUTE.	Designer Service
SYMBOLCANCREATEROOT	Create root symbols. Must be used with SYMBOL, CREATESYMBOL.	Designer Service
SYMBOLCANDELETEROOT	Delete root symbols. Must be used with SYMBOL, SYMBOLDELETE.	Designer Service

Value	Description	Platform License
SYMBOLCREATE	Create symbols. Must be used with SYMBOL.	Designer Service
SYMBOLDELETE	Delete symbols. Must be used with SYMBOL.	Designer Service
SYMBOLREMOVE	Remove symbols from a hierarchy. Must be used with SYMBOL.	Designer Service
SYMBOLSET	Modify symbols. Must be used with SYMBOL.	Designer Service
SYMBOLSWITCH	Switch symbols in a hierarchy. Must be used with SYMBOL.	Designer Service
SYSTEMATTRIBUTECREATE	Create system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEDELETE	Delete system attributes. Must be used with ATTRIBUTE.	Designer Service
SYSTEMATTRIBUTEMODIFY	Modify system attributes. Must be used with ATTRIBUTE.	Designer Service
USERADMINISTRATION	Perform user administration	Does not apply
USERATTRIBUTECREATE	Create user attributes. Must be used with ATTRIBUTE.	Designer Service

Value	Description	Platform License
USERATTRIBUTEDELETE	Delete user attributes. Must be used with ATTRIBUTE.	Designer Service
USERATTRIBUTEMODIFY	Modify user attributes. Must be used with ATTRIBUTE.	Designer Service
USERRESETPASSWORD	Reset passwords.	Does not apply
USERSYMBOLACCESS	Manage symbol access for users.	Does not apply
VIEWDATA	View data.	Designer Power User Viewer Service
WORKFLOWDESIGNER	Access Longview Workflow Designer.	Designer

- AuthorizationGroup is required only when AuthorizationName is USERADMINISTRATION or USERRESETPASSWORD, and is the group from which you want the indicated user's authorization removed.

- GroupName is the name of a group. For AUTHORIZATION, this parameter is optional and required only when AuthorizationName is USERADMINISTRATION or USERRESETPASSWORD.

  For ACCESS, if the optional AccessType parameter is not included in the syntax, the command removes all access to the specified symbol from the indicated user. If the optional SymbolName and AccessType parameters are not included in the syntax, the command removes all symbol access for the dimension and role combination from the indicated user. If the syntax contains only "REMOVE USER [DomainName\]UserId ACCESS", the command removes all access from the indicated user.

  For AUTHORIZATION, if the syntax contains only "REMOVE USER [DomainName\]UserId AUTHORIZATION", the command removes all authorizations from the indicated user.

  For GROUP, if the optional GroupName parameter is not included in the syntax, the command removes the indicated user from all groups.

Syntax example:

```
MAINTENANCE ON
```

```
REMOVE USER Gilda ACCESS ACCOUNTS AnalystRole Cash W
MAINTENANCE OFF
```

Creating an ASCII input file

To remove the access privileges, authorizations, or group membership for a large number of users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId{ACCESS{Dimemsion{RoleName[{SymbolName][{AccessType]

[DomainName\]UserId{ACCESS{Dimension{RoleName
[DomainName\]UserId{AUTHORIZATION{AuthorizationName[{AuthorizationGroup]
[DomainName\]UserId{GROUP{GroupName
[DomainName\]UserId{GROUP
```

Use the QUICKFORMAT command, if necessary, to make formatting changes to the ASCII file.

Syntax (with an ASCII file):

```
REMOVE USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

REMOVE USER @Access.txt
MAINTENANCE OFF
```

See also

- QuickFormat

Rename File

Use this command to change the name of an existing file. This command works with absolute and relative paths.

The allowed paths for the RENAME FILE command are specified by the PERMITTED_FILEMAINTENANCE_PATH parameter in the lv_af.cfg file, located in the working directory of the lv_af.exe. Files that cannot be renamed are defined by the EXCLUDED_FILEMAINTENANCE_FILES parameter in the same file. These parameters only apply when they are set.

Syntax:

```
RENAME FILE FROM "Source" TO "NewName"
```

where:

- Source is the name of the file that will be renamed.
- NewName is the name that the file will be renamed to.

Syntax example:

```
RENAME FILE FROM "logs\export\export.log" TO "oldexport.log"
```

See also

- Copy File
- Create Directory
- Delete File
- FileExists
- GetFileProperties
- Move File

Reset MetadataAuditTrail

Use this command to reset metadata audit trail information. You can reset specific metadata audit trail information by using a AuditSpec document to filter the results retrieved.

Note:

- Using the Metadata Audit Trail functionality requires that the Metadata Audit Trail parameter be set to TRUE in Longview Server Manager (the default value is FALSE). For more information, see the Longview Server Manager Guide. If you do not have access to this functionality, contact your System Administrator.
- If the MetaData Audit Trail Log Files Location parameter is set in Longview Server Manager, the metatdata audit trail records that have been reset will be logged to the specified file.

Syntax:

```
Reset METADATAAUDITTRAIL [USING auditspecName] [comment]
```

where:

- auditSpecName is the name of the AuditSpec document to filter the result. This parameter is optional. If you do not specify an auditSpecName, all metadata audit trail data will be reset.
- comment is an optional character string to be used as a comment.
  - Enclose in double quotation marks.
  - To include double quotation marks in the character string, precede the interior double quotation marks with a backslash (\ ).

Note:

- There are specific AuditSpec functions that can be used to filter the DataAuditTrail results. See “Using DataSpec functions”
- Depending on your user and/or group authorizations, you may not be able to use this command, or you may only be able to reset non-security related metadata activity.

Syntax example:

```
Reset METADATAAUDITTRAIL USING MyResetAuditSpec.lvaud "Remove entires
created during testing"
```

Resolve

Use this command to resolve tokens within a file. The tokens can resolve to variables, attributes, or values in a data area. The command reads in a specified file and generates a new file, resolving any tokens encountered.

Syntax:

```
RESOLVE sourceDocName targetDocName [APPEND]
```

where:

- SourceDocName is the name of the file containing the tokens. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Variable File.txt"
  ```

- targetDocName is the name of the output document.
- APPEND is optional and adds the output to an existing text file.

If the named text file already exists and APPEND is not designated, the existing text file will be overwritten and any data stored in it will be lost.

Syntax example:

```
RESOLVE sourceDocName.xml targetDocName1.xml
```

Syntax example:

```
RESOLVE sourceDocName2.html targetDocName2.html APPEND
```

Within the source document, tokens are specified by enclosing the token between two square brackets [[ ]]. The following tokens are supported:

Token Type	Syntax and Description
Variables	[[VARIABLE,VariableName]] Resolves to the specified variable. where ▪ VariableName is the name of the variable to resolve

Token Type	Syntax and Description
Attributes	[[ATTRIBUTE, AttrClass, AttrName, AttrObject]] Resolves to the specified attribute. where ■ AttrClass is the Attribute class of the attribute that you wish to resolve. ■ AttrName is the Attribute name of the attribute that you wish to resolve. ■ AttrObject is the Attribute object of the attribute that you wish to resolve.
Base Data	[[DATA, DataAreaName,SymName1,SymName2,.., [Format]]] Resolves to the specified data value. where ■ DataAreaName is the name of the DataArea in which the symbols are found ■ SymName is the name of a symbol for each dimension in the DataArea. ■ Format is an optional parameter to specify the data format of numeric data that is resolved. For example, 10.2 means 10 characters wide, 2 decimal places. The default format, if not specified, is to return the value as stored in the Longview Data Server.
Schedule Data	[[SCHEDDATA,DataAreaName,SymName1,SymName2,...,SchedName, ExtraDimSymName1,ExtraDimSymName2,... [Format]]] Resolves to the specified data value. where ■ DataAreaName is the name of the DataArea in which the symbols are found. ■ SymName is the name of a symbol for each dimension in the DataArea. ■ ExtraDimSymName is the name of a symbol for each schedule dimension in the DataArea. ■ Format is an optional parameter to specify the data format of numeric data that is resolved. For example, 10.2 means 10 characters wide, 2 decimal places. The default format, if not specified, is to return the value as stored in the Longview Data Server.
Include	[[INCLUDE, filename]] Resolves the specified file and inserts the resolved content into the current file. where ■ filename is the name of another file to be resolved within the parent document.

Note: *You may nest most tokens but variables cannot be nested within another variable.

Source File Example:

The Resolve command can be used to generate xml reports that reference Longview data, attributes, and variables within a procedure.

The following snippet is an example of an xml file that references the supported tokens for the Resolve command:

CountryByCountryReporting.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<CBC_OECD version="">
<MessageSpec>
<SendingEntityIN>ABC Inc</SendingEntityIN>
<TransmittingCountry>CA</TransmittingCountry>
[[INCLUDE, XML_Header_RecCountry_Res.xml]]
<MessageType>CBC</MessageType> <!--MessageType_EnumType-->
<Language>EN</Language> <!--LanguageCode_Type*-->
<Warning>[[VARIABLE,warningText]]</Warning>
<Contact>Free text containing contact information</Contact>
<MessageRefId>SendersUniqueIdentfierForThisMessage</MessageRefId>
<MessageTypeIndic>[[VARIABLE,MessageTypeIndic]]</MessageTypeIndic>
<CorrMessageRefId>
[[DATA,da,MsgRefID,Dim1Set,Dim2Set,Dim3Set,Dim4Set,Dim5Set]]
</CorrMessageRefId>
<ReportingPeriod>
[[ATTRIBUTE, SYSTEM, ASReportingPer, DBDEFAULT]]
</ReportingPeriod>
</MessageSpec>
<CbcBody> <!--CbcBody_Type (unbounded)-->
```

Resource Reset

Use this command to clear all resources in the memory. The resource bundle will be loaded again the next time it is accessed.

Syntax

```
RESOURCE RESET
```

Syntax example

```
LoadKar NetInterestIncome.kar CPM\NII\karfiles
Set PROPERTY TST.Message1 = "[[Resource,Issue\Messages,Text1]]"
Show message "$TST.Message1$"
LoadKar NetInterestIncome.kar CPM\NII\karfiles
RESOURCE RESET
Set PROPERTY TST.Message1 = "[[Resource,Issue\Messages,Text1]]"
Show message "$TST.Message1$"
```

See also

- Load
- LoadKar
- Unload

Rest DeleteRequestHeader

Use this command to delete a header and its value that was previously set using REST SETREQUESTHEADER. This command can be used to reset previously defined headers or remove headers that do not apply to the current REST EXECUTEGET, REST EXECUTEHEAD, REST EXECUTEPOST, or REST EXECUTEPUT request.

Syntax:

```
REST DELETEREQUESTHEADER [HeaderName]
```

where:

- HeaderName is the name of the header to delete. If a HeaderName is not provided, all request headers will be deleted from memory.

Syntax example:

```
REST DELETEREQUESTHEADER "Content-Type"
```

Rest ExecuteDelete

Use this command to delete a resource identitifed by a URI.

Syntax:

```
REST EXECUTEDELETE url [TO Target ] [Timeout]
```

where:

- url identifies the resource to delete.
- Target is the variable or document name that will contain the response.
- Timeout is the time to wait, in seconds, for the total request.

Syntax example:

```
REST EXECUTEDELETE "http://abc.someurl.com/subscription/123" TO TestResult1
```

Rest ExecuteGet

Use this command to retrieve information returned from a REST HTTP GET method. You may use this command in conjunction with REST GETRESPONSEHEADER to retrieve additional information about the response. You may also use this command in conjunction with the REST GETRESPONSESTATUS command to retrieve additional information regarding the status of the request.

Syntax:

```
REST EXECUTEGET url TO Target [Timeout]
```

where:

- url is the resource to retrieve the information from.
- Target is the variable or document name that will contain the response.
- Timeout is the time to wait, in seconds, for the total request.

Syntax example:

```
// Example with Target as a variable
CREATE VARIABLE TestResult1 AS String
REST EXECUTEGET "http://api.fixer.io/latest" TO TestResult1
EXPORTVARIABLE TestResult1 TestResult1.json

// Example with Target as a document
CREATE DOCUMENT "TestResult2.json"
REST EXECUTEGET "http://api.fixer.io/latest" TO "TestResult2.json"
SAVE "TestResult2.json"
```

Rest ExecuteHead

Use this command to retrieve headers that would be returned if the URL were requested with the Rest ExecuteGet command. The command can be used for obtaining metainformation about the entity specified by the request without transferring the entity-body itself. In conjunction with REST GERESPONSEHEADER and REST GETRESPONSESTATUS commands, this command can be used for testing hypertext links for validity, accessibility, and recent modification.

Syntax:

```
REST EXECUTEHEAD url [Timeout]
```

where:

- url is the resource to retrieve the information from.
- Timeout is the time to wait, in seconds, for the total request.

Syntax example:

```
REST EXECUTEHEAD "http://api.fixer.io/latest"
```

Rest ExecutePost

Use this command to post to a resource using a restful HTTP POST method. You may also want to use this command in conjunction with the REST GETRESPONSESTATUS command to retrieve additional information regarding the status of the request.

Syntax:

```
REST EXECUTEPOST url USING Source TO Target [Timeout]
```

where:

- url is the resource where the data will be posted.
- Source is the data to post. Source can be a variable or a document name.
- Target is the variable or document that will contain the response.
- Timeout is the time to wait, in seconds, for the total request.

Syntax example:

```
CREATE VARIABLE TestPost AS String
CREATE VARIABLE TestPostResponse AS String
SET VARIABLE TestPost = "$MyRequest$"
CREATE DOCUMENT "TestPost.json"
WRITE "TestPost.json" VARIABLE TestPost APPEND
REST SETREQUESTHEADER "Content-Type" "application/json"
REST EXECUTEPOST "https://httpbin.org/post" USING "TestPost.json" TO
TestPostResponse
EXPORTVARIABLE TestPostResponse "TestPostResponse.json"
```

Rest ExecutePut

Use this command to put or update data to a resource using a restful HTTP PUT method. You may also use this command in conjunction with the REST GETRESPONSESTATUS command to retrieve additional information regarding the status of the request.

Syntax:

```
REST EXECUTEPUT url USING Source TO Target [Timeout]
```

where:

- url is the resource where the data will be put.
- Source is the data to put. Source can be a variable or a document name.
- Target is the variable or document that will contain the response.
- Timeout is the time to wait, in seconds, for the total request.

Syntax example:

```
CREATE VARIABLE TestPutResponse as String

CREATE VARIABLE TestPut as String

SET VARIABLE TestPut = "$MyRequest$"

REST EXECUTEPUT "https://jsonplaceholder.typicode.com/posts/1" USING TestPut
TO TestPutResponse

EXPORTVARIABLE TestPutResponse TestPutResponse.xml
```

Rest GetResponseHeader

Use this command to retrieve a specified response header for the REST EXECUTEGET, REST EXECUTEHEAD, REST EXECUTEPOST, or REST EXECUTEPUT command that was previously invoked. The response header is reset after any REST EXECUTEGET, REST EXECUTEHEAD, REST EXECUTEPOST, or REST EXECUTEPUT command.

Syntax:

```
REST GETRESPONSEHEADER ResponseHeaderName VariableName
```

where:

- ResponseHeaderName is the name of the header to retrieve
- VariableName is the name of a string variable that will contain the value of the specified response header.

Syntax example:

```
CREATE VARIABLE TestResponseHeader as String
REST GETRESPONSEHEADER “Content-Type” TestResponseHeader
EXPORTVARIABLE TestResponseHeader TestResponseHeader.txt
```

Rest GetResponseStatus

Use this command to retrieve the HTTP response status for the REST EXECUTEGET, REST EXECUTEHEAD, REST EXECUTEPOST, or REST EXECUTEPUT command that was previously invoked. The response status is reset after any REST EXECUTEGET, REST EXECUTEHEAD, REST EXECUTEPOST, or REST EXECUTEPUT command.

Syntax:

```
REST GETRESPONSESTATUS VariableName
```

where:

- VariableName is the name of a numeric variable that will contain the value of the response status. The default value is 0

Syntax example:

```
CREATE VARIABLE TestResponseStatus as Num

REST GETRESPONSESTATUS TestResponseStatus
EXPORTVARIABLE TestResponseStatus TestResponseStatus.txt
```

Rest SetRequestEncoded

Use this command to url encode the JSON data you wish to send via a REST EXCECUTE call such as REST EXECUTEPOST or REST EXECUTEPUT call. The encoding takes the JSON string and replaces each of the following characters with a % hex encoding for the following characters:

- { is encoded as %7B
- }is encoded as %7D
- : is encoded as %3A
- , is encoded as %2C
- = is encoded as %3D
- \ is encoded as %5C

Syntax:

```
REST SETREQUESTENCODED Value [ParameterName]
```

where:

- value is TRUE or FALSE and indicates whether to url encode the data. The default is FALSE.
- ParameterName is the name of the optional parameter that will contain the url encoded data. If this parameter is not specified, the encoded data will be contained in a parameter named "jsonData".

Syntax example:

```
REST SETREQUESTENCODED TRUE jsonData
```

In the example, the data being sent will be encoded and sent as a value for the parameter jsonData.

Rest SetRequestHeader

Use this command to set the header of the REST call that you are about to execute. For example, you may want to set a header that sets the content type of the information that you will be sending via a REST EXECUTEPOST or REST EXECUTEPUT call.

> **Note:** You may set several headers before making the REST EXECUTEPOST, REST EXECUTEPUT, REST EXECUTEHEAD, or REST EXECUTEGET calls.

Syntax:

```
REST SETREQUESTHEADER HeaderName Value
```

where:

- HeaderName is the name of the header response to retrieve.
- Value is the value associated with the specified header.

Syntax example:

```
REST SETREQUESTHEADER “Content-Type” “application/json”
```

RestAPI GetRequestHeader

Use this command to retrieve a specified request header in the Solution-level REST APIs. This command is useful when creating Solution-level REST APIs that may be called by third-party applications. For more information see "Providing REST APIs" in the Longview Integration Guide.

Syntax:

```
RESTAPI GETREQUESTHEADER RequestHeaderName VariableName
```

where:

- RequestHeaderName is the name of the header to retrieve
- VariableName is the name of a string variable that will contain the value of the specified request header.

Syntax example:

```
CREATE VARIABLE TestRequestHeader as String

RESTAPI GETREQUESTHEADER “Content-Type” TestRequestHeader
EXPORTVARIABLE TestRequestHeader TestRequestHeader.txt
```

RestAPI SetResponseHeader

Use this command to set custom HTTP headers for a response. This command is useful when creating Solution-level REST APIs that may be called by third-party applications. For more information see "Providing REST APIs" in the Longview Integration Guide.

Syntax

```
RESTAPI SETRESPONSEHEADER HeaderName Value
```

where:

- HeaderName identifies the header to set.
- Value is the value associated with the specified header.

Syntax example:

```
RESTAPI SetResponseHeader content-type text/plain
```

Run (for ExportSpecs)

Use this command to write information to a file.

Syntax:

```
RUN EXPORT ExportSpecFile ON ObjectName
```

where:

- ExportSpecFile is the .lvexp file that contains the definition of the export to be performed. It can include a complete or partial folder path in the format C:\...\ExportSpecFile. If ExportSpecFile includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Export Spec.lvexp"
  ```

  If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

- ObjectName is the name of a DataArea or a DataTable object.

Syntax example:

```
RUN EXPORT TEST\EXPORT\EXPORT.lvexp ON DA1
```

See also

- Creating ExportSpecs
- Creating Model documents

Run (for ImportSpecs)

Use this command to read a text file containing a series of commands and execute them in sequence.

Syntax:

```
RUN IMPORT ImportSpecFile TO ObjectName
```

where:

- ImportSpecFile is the .lvimp file that contains the definition of the import to be performed. It can include a complete or partial folder path in the format C:\...\ImportSpecFile. If ImportSpecFile includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Import Spec.lvimp"
  ```

  > **Note:** If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

- ObjectName is the name of a DataArea or a DataTable object.

Syntax example:

```
RUN IMPORT TEST\EXPORT\EXPORT.lvimp TO DA1
```

See also

- Creating ImportSpecs
- Creating Model documents

Run (for Models)

Use this command to read a text file containing a series of commands and execute them in sequence.

Syntax:

```
RUN MODEL ModelFile ON DataAreaName [ProtectedMode]
```

where:

- ModelFile is the name of a Model file.
- DataAreaName is the name of a DataArea.
- ProtectedMode is optional and is either TRUE or FALSE. If TRUE, unlocked cells are not affected. If FALSE, all cells are affected.

Note: When ProtectedMode is TRUE, the model runs only on cells covered by the data lock.

Syntax example:

```
RUN MODEL TEST\EXPORT\EXPORT.LMOD ON DA1
```

See also

- Creating ImportSpecs
- Creating Model documents

Run (for Procedures)

The RUN PROCEDURE command reads a text file containing a series of commands and executes them in sequence. Typically, the Procedure document contains a RUN MODEL command to perform a series of calculations on a DataArea.

Syntax:

```
RUN PROCEDURE ProcFile
```

where:

- ProcFile is the name of a procedure file.

Syntax example:

```
RUN PROCEDURE PROCTEST\EXPORT\EXPORT.LPRO
```

See also

- Creating ImportSpecs
- Creating Model documents

Save

Use this command to write the contents of a document in the document cache to a physical file. This can be useful when debugging your code during development.

Syntax:

```
Save DocName
```

where:

- DocName is the name of the document in the document cache to save to a physical file, and should match the name specified in the related Create Document command. Documents that did not include a file path in the Create Document statement are saved to MyDocuments\Longview. If the document name included a file path, the document is saved relative to the MyDocuments\Longview with the same file path.

Syntax example without a file path:

```
Save MyDataView.lvdvw
```

Syntax example with a file path:

```
Save "C:\Testing\Application Framework\MyDataView.lvdvw"
```

See also

- Create Document
- Write
- Unload

Send Email

Use this command to send an email from Longview Application Framework.

> **Note:** You must set up the email environment before you can use the Send Email command. For more information, see Set Email.

Syntax:

```
SEND EMAIL "Subject" Document
```

where:

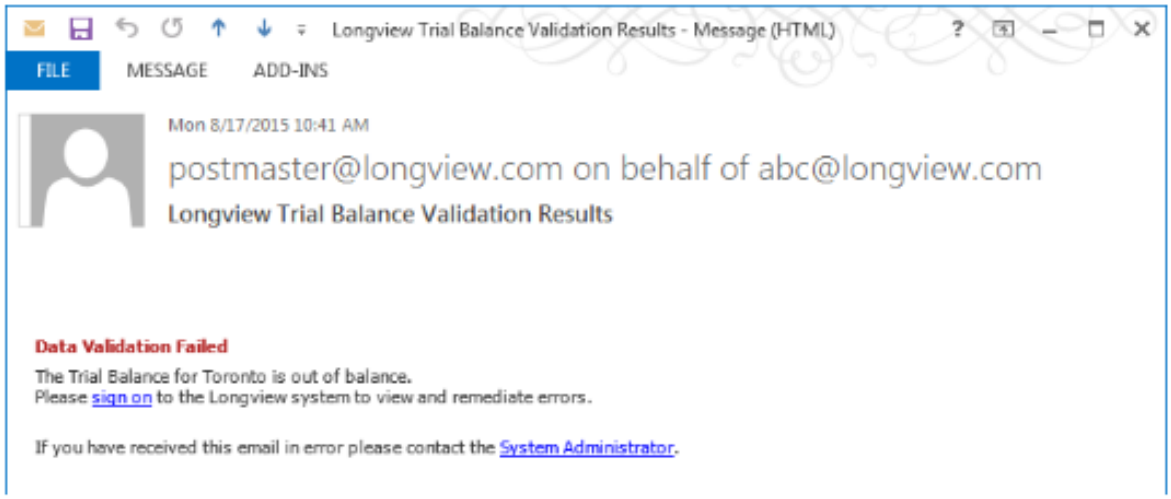
- Subject is the subject to display in the email.
- Document is the email content file. The file can be in memory or on disk. To send an email as HTML, the file suffix must be htm or html. If you're using Application Framework to generate the contents of an HTML file, you may need to use HTML character entities for special characters, including double quotation marks (") and the appropriate tags for formatting elements such as carriage returns (
).

Syntax example:

```
SEND EMAIL "Hello World" c:\Users\Lview\Documents\Sample_content_file

SEND EMAIL "Notice to Employees" c:\DJO\Documents\Notices\Employee_
notice.html
```

HTML file example

You can use .html to format your email with links and custom fonts, colors, and spacing. For example, you can format your email like this:



Sample email

To format the above email, you could use .html similar to the following in your email document:

```
<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=windows-1250">

<style type="text/css">

body {color:#333333; font-family: Tahoma, Geneva, sans-serif; font-size:
11px; margin-top: 10px; overflow:auto;}

h1 {color:#3d6e9f; font-family: Tahoma, Geneva, sans-serif; font-size: 12px;
font-weight:bold; margin-top: 0px; margin-bottom: 0px;}

h2 {color:#333333; font-family: Tahoma, Geneva, sans-serif; font-size: 11px;
font-weight:bold; margin-top: 12px; margin-bottom: 0px;}
.pass {color:#008000; font-family: Tahoma, Geneva, sans-serif; font-size:
11px; font-weight: bold; margin-top: 10px; margin-bottom: 0px;}
.fail {color:#B22222; font-family: Tahoma, Geneva, sans-serif; font-size:
11px; font-weight: bold; margin-top: 10px; margin-bottom: 0px;}
p {margin-top: 6px; margin-bottom: 0px;}

</style>

</head>

<body>

<h2 class="fail">Data Validation Failed</h2>

<p>The Trial Balance for Toronto is out of balance. <br>

Please <a href="http://webserver/LVCPMDEV" target="_blank">sign on</a> to
the Longview system to view and remediate errors.</p>

<br>

<p>If you have received this email in error please contact the <a
href="mailto:LVAdmin@abc.com?Subject=Data Validation Email Notification"
target="_top">System Administrator</a>.</p>

</body>

</html>
```

See also

- Set Email

ServerMath

Use this command to turn all server calculations (rollup, server rules, translations, etc.) on or off. For example, when you change the value of a symbol and submit the changed value, your system calculates the values of the ancestors of the symbol, up to the root symbol in each hierarchy.

Since this operation may slow down performance, use the ServerMath command to turn off the server calculation when you run processes that change many cells (for example, a period-end rollover). You will need to perform an enterprise restatement once you have finished submitting all the data.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax:

```
SERVERMATH StatusType [CURRENTSESSION]
```

where:

- StatusType can be one of the following:

Value	Description
ON	To perform data server calculations automatically as cells are submitted to the Data Server repository.
OFF	To stop calculating as cells are submitted to the Data Server repository.
OFF CURRENTSESSION	To turn off the ServerMath flag for the current user session only (does not require Maintenance mode).
DEFAULT CURRENTSESSION	To restore the ServerMath flag to the default after using the OFF CURRENTSESSION command (does not require Maintenance mode). Users are not allowed to turn on the ServerMath flag for the session, but they can turn it off. Once the user disconnects or signs off, the changed setting is lost.

Syntax example:

```
MAINTENANCE ON
SERVERMATH ON
SERVERMATH DEFAULT CURRENTSESSION
SERVERMATH OFF
CURRENTSESSION
MAINTENANCE OFF
```

Set Attribute

Use this command to modify or set the access, default value, description, name, and value of an Attribute.

An Attribute is data used to describe the characteristics of an object. For example, SWFAdminEmail is an Attribute representing the email address of an Administrator. Attributes may be referenced as variables, and their value retrieved for one or a series of symbols.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax regular:

```
SET ATTRIBUTE AttrClass AttrName DEFAULT TO "DefaultValue"

SET ATTRIBUTE AttrClass AttrName DESCRIPTION LanguageCode TO "String"

SET ATTRIBUTE AttrClass OldAttrName NAME TO NewAttrName

SET ATTRIBUTE AttrClass AttrName VALUE AttrObject "AttrValue"

SET ATTRIBUTE AttrClass AttrName ACCESS TO AccessType
```

where:

- AttrClass is the Attribute class and AttrObject is the Attribute object. Select from the following:

AttrClass	Description
SYSTEM	Describes the entire system at the highest level. Attributes of this Attribute class specify system-wide characteristics. There is only one object in the SYSTEM Attribute class: DBDEFAULT.
USER	Describes the Attributes of a particular user. Each user is an object in the USER Attribute class.
SYMBOL	Describes the characteristics of individual symbols. Each symbol is an object in the SYMBOL Attribute class.

- AttrName is the Attribute name.
- "DefaultValue" is the default value for the Attribute, enclosed in double quotation marks.
- LanguageCode is the two letter language code for the Attribute description.
- "String" is a character string to be used as a description of the Attribute, enclosed in double quotation marks.
- OldAttrName is the existing name of the Attribute.

- NewAttrName is the new name for the Attribute. Because Attribute names can be confusing, Longview uses strict naming conventions for creating Attributes. For more information, see the Longview Application Administrator Guide.
- AttrObject is the Attribute object.
- "AttrValue" is the value of the Attribute, enclosed in double quotation marks.
- AccessType sets the user's read/write access. Select one of the following:

Value	Description
READ	Read-only access, to allow the user to read the data but not change it. This is the default.
WRITE	Write access, to allow the user to change data. This can be set only for User Attributes.

Note: For USER Attributes, the user can be authorized to both view and change an Attribute. For SYMBOL and SYSTEM Attributes, the user can only be authorized to view an Attribute.

Syntax example:

```
MAINTENANCE ON

SET ATTRIBUTE SYMBOL AZCREDACCT DEFAULT TO "TrialBal"
MAINTENANCE OFF
```

Creating an ASCII input file

To set a large number of Attributes, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
AttrClass{AttrName{DEFAULT{TO{DefaultValue

AttrClass{AttrName{DESCRIPTION{LanguageCode{TO{DefaultValue
AttrClass{AttrName{NAME{TO{AttrNewName
AttrClass{AttrName{VALUE{AttrObject{AttrValue
AttrClass{AttrName{ACCESS{TO{AccessType
```

Syntax (with an ASCII file):

```
SET ATTRIBUTE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

```
@"C:\My Documents\My Data.txt"
```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
SET ATTRIBUTE @Attributename.txt
```

Set Category

Use this command to change the name or description of a category.

Syntax:

```
SET CATEGORY CategoryName NAME TO "NewName"
                SET CATEGORY CategoryName DESCRIPTION Language TO "Description"
```

where:

- CategoryName is the name previously given to, or currently being given to, the category.
- NewName is the new name of the category. The following characters are invalid: backslash (\ ), forward slash (/), apostrophe (’), double quotation marks (“), comma (,), left brace ({), and double brackets ([[ ) or ( ]]). The category name can have a maximum length of 40 characters.
- Language is the language in which the description is displayed, and can be EN or FR.
- Description is the string describing the category(e.g., the category’s function). If the description includes any spaces, it must be enclosed in double quotation marks. The following characters are invalid: left brace ({). The category description can have a maximum length of 100 characters.

Syntax example:

```
SET CATEGORY ABC DESCRIPTION EN TO "ABC Category"
```

Sample example:

```
SET CATEGORY ABC NAME TO "XYZ"
```

Set DataArea

Use this command to reset the flag for a specific DataArea that has been modified since the initial download.

This command is useful if, after the initial download of the DataArea, you require modeling before displaying the DataArea to the user and you want to track only user-initiated changes to the DataArea.

Syntax:

```
SET DATAAREA Name DATACHANGED TO FALSE
```

where:

- Name is the name of the DataArea for which you want to reset the flag.

Syntax example:

```
SET DATAAREA da1 DATACHANGED TO FALSE
```

Set DataRole

Use this command to change the name of a DataRole created by the CREATE DATAROLE command.

Syntax regular:

```
SET DATAROLE OldRoleName TO NewRoleName
```

where:

- OldRoleName is the original name of a symbol access role.
- NewRoleName is the new name to be applied to a symbol access role.

Syntax example:

```
SET DATAROLE NorthAmAccounts TO AccountsNorthAmerica
```

Creating an ASCII input file

To set a change the names of a large number of DataRoles, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information:

```
OldRoleName{TO{NewRoleName
```

Syntax (with an ASCII file):

```
SET DATAROLE @FileName
```

Set DataTable

Use this command to reset the flag for a specific Data Table that has been modified since the initial download.

This command is useful if, after the initial download of the Data Table, you require modification of some of the data before displaying the Data Table to the user and you want to track only user-initiated changes to the Data Table.

Syntax:

```
SET DATATABLE Name DATACHANGED TO FALSE
```

where:

- Name is the name of the Data Table for which you want to reset the flag.

Syntax example:

```
SET DATATABLE dt1 DATACHANGED TO FALSE
```

Set Email

Use this command to set up the email environment for an email sent from Longview Application Framework, including the email address to send the message to. You must set the email properties before you can use the Send Email command.

> **Note:** SMTP authorization and encryption cannot be set using the Set Email command. Your third-party email administrator configures these settings.

You can also use this command to specify an attachment to send with the email. The permitted paths for attachments are determined by the lv_af.cfg file found in the working directory of the lv_af.exe. While events are running, the lv_af.exe will look for the lv_af.cfg in the working directory of the Data Server. For example, `C:\Longview\LVCPM\DataServers\LVCPM`.

> **Note:** The working directory of the Data Server can be specified in the lvsrvr.cfg file under the WORK_DIRECTORY parameter.

To change the permitted file paths, open the lv_af.cfg file in your preferred text editor, and modify the PERMITTED_ATTACHMENT_PATH statement. There are no path restrictions if the lv_af.cfg file does not exist or if the statement value is blank. Lines in the lv_af.cfg file cannot surpass 4094 characters. Separate multiple paths with a semicolon (;).

For example:

```
PERMITTED_ATTACHMENT_PATH =

C:\tmp;C:\lv\source\ApplicationFramework\bin\email;\\jsmith39203\c$\lv\sourc
e\ApplicationFramework\bin\49681
```

By default, emails are sent unecrypted and unauthenticated. To send an ecrypted and authenticated email, use the SET EMAIL CREDENTIALS command. Alternatively, if you wish to separate the credentials from application-defined scripts, you can define the host, port, username and password credentials in the lv_af.cfg file found in the working directory of the lv_af.exe. To specify the credentials, open the lv_af.cfg file in your preferred text editor, and add the SMTP_SERVERHOST, SMTP_SERVERPORT (does not need to be specified if the default port is 25), SMTP_USERNAME and SMTP_PASSWORD statements. For example:

```
SMTP_SERVERHOST=host

SMTP_SERVERPORT=port

SMTP_USERNAME=username

SMTP_PASSWORD=password
```

When SMTP_SERVERHOST, SMTP_SERVERPORT, SMTP_USERNAME and SMTP_PASSWORD are specified in the lv_af.cfg, it is equivalent to calling SET EMAIL SERVER and SET EMAIL CREDENTIALS with these parameters. However, if the lv_af.cfg parameters exist and the SET EMAIL SERVER and SET EMAIL CREDENTIALS are also set in the Application Framework procedure document, the credentials specified in SET EMAIL SERVER and SET EMAIL CREDENTIALS take precedence.

Syntax:

```
SET EMAIL SERVER "Host" [Port]
SET EMAIL SENDER "Email" ["Name"]
SET EMAIL CREDENTIALS "UserName" "Password"
SET EMAIL TO "ToEmail1|ToEmail2|...ToEmail N" ["ToName1|ToName2|...ToNameN"]
SET EMAIL CC "ccEmail1"|ccEmail2|...ccEmailN" ["ccName1|ccName2|...ccNameN"]

SET EMAIL ATTACHMENT "FileName1|FileName2|...FileNameN"
```

where:

- Host is the SMTP server host name.
- Port is optional and is the port for the SMTP server. The default for this parameter is 25.
- Email is the sender's email address. You must specify the sender's email address.
- Name is the sender's name.
- UserName is the username to use as credentials when SET EMAIL CREDNETIALS is used. SET EMAIL CREDENTIALS is optional and when specified, indicates that encrytption and authentication are required.
- Password is the password to use as credentials when SET EMAIL CREDENTIALS is used. SET EMAIL CREDENTIALS is optional and when specified, indicates that encrytption and authentication are required.

  Note: To clear the username and password, use empty strings. For example: SET EMAIL CREDENTIALS "" "".

- ToEmail11ToEmail2|...ToEmail N are the email addresses of the recipients. You must specify at least one recipient.
- ToName11ToName2|...ToNameN are optional and are the names of the recipients.
- ccEmail1|ccEmail2|...ccEmailN are optional and are the email addresses to be cc'd.
- FileName1|FileName2|...FileNameN are optional and are the files to attach to the email, including the file path, in the format C:\...\FileName. If a FileName includes spaces, enclose it in double quotation marks.
  For example, "C:\My Documents\EmployeeList.txt". If the path contains a dollar sign ($), you must use a variable to store the path.
  For example,

```
SET VARIABLE sAttachments =
"\\ourserver123\c$"+"\LongviewCPMAF\Attachments\Attachment1.TXT|\\our
server123\c$\LongviewCPMAF\Attachements\Attachment1.TXT"

SET EMAIL ATTACHMENT $sAttachments$
```

Separate multiple FileNames with a pipe (|).

Note: If the document is in the same location as lv_af.exe, you do not need to specify the path.

Syntax example:

```
SET EMAIL SERVER "email.server.net" 1300

SET EMAIL SENDER "bsummers@longview.com" "Buffy Summers"

SET EMAIL CREDENTIALS "bsummers" "Password123"

SET EMAIL TO "rgiles@example.com|cchase@example.com|wrosenberg@example.com"
"Rupert Giles|Cordelia Chase|Willow Rosenberg"

SET EMAIL CC "xharris@example.com" "Xander Harris"

SET EMAIL ATTACHMENT "weekly_schedule.pdf"

SET EMAIL ATTACHMENT
"S:\CompanyShared\employeelist.doc|S:\CompanyShared\StaffPhotos\employeephot
o.jpg"
```

See also

- Send Email

Set FTP

Use this command to set FTP parameters to be used in conjunction with the FTP PUT command. The FTP PUT command copies files to a remote server using Application Framework.

The SET FTP command can set any of the following parameters:

Value	Description
Server	Specify the server and port to copy to, as well as the protocol (FTP or SFTP).
Credentials	Specify the FTP username and password credentials.
PreCommand	Specify commands to run before the copy occurs. This allows for operations to run before transferring the file. The actual commands are server dependent. If more than one FTP PRECOMMAND is specified, the commands are queued up and executed in the order that they are encountered. You may find it useful to use this command in conjunction with FTP CLEARCOMMANDS to clear the PreCommand queue once they are no longer required.
PostCommand	Specify commands to run after the copy occurs. This allows for operations such as renaming a file after transferring it (i.e. transfer abc.tmp, then rename to abc.final, so that a file watcher can be triggered after completion). The actual commands are server dependent. If more than one FTP POSTCOMMAND is specified, the commands are queued up and executed in the order that they are encountered. You may find it useful to use this command in conjunction with FTP CLEARCOMMANDS to clear the PostCommand queue once they are no longer required.
ClearCommands	The FTP CLEARCOMMANDS command is used in conjunction with FTP PRECOMMAND and FTP POSTCOMMAND and will clear the PreCommand and PostCommand queues.

Syntax:

```
SET FTP SERVER "Host" [Protocol [Port]]

SET FTP CREDENTIALS "UserName" "Password"
SET FTP PRECOMMAND "Command"
SET FTP POSTCOMMAND "Command"
SET FTP CLEARCOMMANDS
```

where:

- Host is the server hostname.
- Protocol is the transfer protocol and can be either FTP or SFTP.

> **Note:** SFTP is only supported for Application Framework running on Linux OS. SFTP is not supported for Windows OS.

- Port is optional and is the port for the Host specified.
- UserName is the username to use in the FTP credentials.
- Password is the password to use in the FTP credentials.
- Command is the command to run before or after the copy occurs. The commands are server dependent.

See also

- FTP Put

Set Group

Use this command to change the name or description of a user group.

Syntax:

```
SET GROUP GroupName NAME TO "NewName"

SET GROUP GroupName DESCRIPTION Language TO "Description"
```

where:

- GroupName is the name previously given to, or currently being given to, the group.
- NewName is the new name of the user group.
- Language is the language in which the description is displayed, and can be EN or FR.
- Description is the string describing the group (e.g., the group's function). If the description includes any spaces, it must be enclosed in double quotation marks.

Syntax example:

```
SET GROUP ABC NAME TO "ABC Administrators"
```

Syntax example:

```
SET GROUP ABC DESCRIPTION EN TO "ABC Administrators"
```

Creating an ASCII input file

To make changes for a large number of User Groups, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each symbol:

```
GroupName{NAME{TO{NewName
GroupName{DESCRIPTION{Language{TO[Description
```

Syntax (with an ASCII file):

```
SET GROUP @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks;

for example:

```
@"C:\My Documents\My Data.txt"
```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

SET GROUP @C:\Temp\Groupdescs.txt
MAINTENANCE OFF
```

Set Property

Use this command to set a value on a pre-defined property of an object type variable.

Syntax:

```
SET PROPERTY ObjectVarName.Property = Value
```

where:

- ObjectVarName is the name of tHe object variable.
- Property is the property that has been created on the specified object variable to be set.
- Value is the value to set.

Syntax example:

```
// Create a variable of type object

Create Variable TestObject as Object

// Create properties of the object
CREATE PROPERTY TestObject.comments as String
CREATE PROPERTY TestObject.anyNumber as Num

// Set properties of an object
SET PROPERTY TestObject.comments = "My comment"
SET PROPERTY TestObject.anyNumber = 777
```

Set Rule

Use this command to change the description of a rule in the database, or set the description in alternate languages.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax:

```
SET RULE RuleID DESCRIPTION LanguageCode TO "RuleDesc"
```

where:

- RuleID is the ID of the rule to be set.
- RuleDesc is the new description you want to give the rule.
- LanguageCode is the language in which to change or set the description for.

Syntax example:

```
MAINTENANCE ON

SET RULE 101 DESCRIPTION EN TO "ABC Rule"
MAINTENANCE OFF
```

Creating an ASCII input file

To make changes for a large number of rules, specify the parameters for this command in an ASCII file.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users.

For more information, see Maintenance or Exclusive.

Create an ASCII file containing the following information for each symbol:

```
RuleName{DESCRIPTION[LanguageCode{TO[RuleDesc
```

Syntax (with an ASCII file):

```
SET RULE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

```
@"C:\My Documents\My Data.txt"
```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

SET RULE @C:\Temp\Ruledescs.asc
MAINTENANCE OFF
```

Set Schedule

Use this command to modify a schedule, schedule dimensions, or schedule symbol description.

Syntax:

```
SET SCHEDULE SchedName DESCRIPTION LanguageCode TO "SchedDesc"

SET SCHEDULE SchedName [DIMENSION DimName] DESCRIPTION LanguageCode TO
"SchedDesc"
```

where:

- SchedName is the name of the schedule.
- DimName is the name of a schedule dimension.
- SymName is the name of symbol in the schedule dimension.
- SchedDesc is the string describing the schedule. If the description includes any spaces, it must be enclosed in double quotation marks.
- LanguageCode is the language code for which you want to change or set the description.

Syntax example:

```
MAINTENANCE ON

SET SCHEDULE Schedule1 DESCRIPTION EN TO "Schedule 1 Description"

MAINTENANCE OFF
```

Syntax example:

```
MAINTENANCE On

SET SCHEDULE Schedule1 DIMENSION ExtraDim1 DESCRIPTION EN TO "Extra
Dimension 1 Description for Schedule 1"

MAINTENANCE OFF
```

Creating ASCII input files

To create a large number of symbols in the Data Server repository, specify the parameters for this command in two ASCII files.

Create an ASCII file containing the following information:

```
SchedName [DIMENSION DimName] DESCRIPTION LanguageCode TO "SchedDesc"
```

Syntax (with an ASCII file):

```
SET SCHEDULE @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Set Symbol

Use this command to modify a symbol in the Data Server repository.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax regular:

```
SET SYMBOL DimName SymName BALANCETYPE TO BalanceType
SET SYMBOL DimName SymName DESCRIPTION LanguageCode TO "SymDesc"
SET SYMBOL DimName SymName NAME TO NewSymName
SET SYMBOL DimName SymName PRIORITY Parent Priority
SET SYMBOL DimName SymName[#NumLevels] ROLLUPSTATUS TO StatusType
SET SYMBOL DimName SymName ROLLUPSTATUS TO StatusType
SET SYMBOL DimName SymName SORTOPTION TO ChildSortOption
SET SYMBOL DimName SymName TYPE TO SymType
SET SYMBOL DimName SymName VIRTUALSTATUS TO StatusType
SET SYMBOL DimName SymName WEIGHT Parent Weight
```

where:

- DimName is a dimension containing the symbol.
- SymName is the name of a symbol.
- BalanceType is one of: Credit|Debit|Neither. Credit and Debit are only used for symbols in the ACCOUNTS dimension. All other dimensions use Neither.
- LanguageCode is the two letter language code for the description.
- "SymDesc" is a character string to be used as a description of SymName enclosed in double quotation marks. For guidelines, see Symbol descriptions. If you do not want to change this value, use " " instead.
- NewSymName is the new name for the symbol. For guidelines, see Symbol names.
- NumLevels is optional and is the pound sign followed a number representing the number of hierarchy levels below SymName to calculate rollup status.
- Priority is a numeric value between 0 and 99 (0 representing the lowest priority) representing the position of the symbol with respect to its parent.

- StatusType can be one of the following:

Value	Description
ON	Parent data will be calculated dynamically (automatically).
OFF	Parent data will not be calculated dynamically (automatically), but only during restatements. **Note:** StatusType cannot be set to OFF if VirtualStatus is ON.

- ChildSortOption is one of the following:

Value	Description
Ascending	Sorts the child symbols in ascending order.
Descending	Sorts the child symbols in descending order.
Manual	Allows manual sorting of child symbols.

- SymType specifies the way SymName rolls up to its parent symbol, and can be STANDARD, CARRYFORWARD, or STATIC.

Value	Description
Standard	To add the symbol into the parent (for example, monthly expenses added into annual expenses).
Carryforward	When the parent value equals the value of the cell immediately preceding it (for example, cash balance as a running year-to-date, where the year-end amount equals the amount for December).
Static	When the child value has no effect on the parent value – for example, for price symbols. (The parents of such symbols are used only to group symbols, not to aggregate their values.) Static symbols do not roll up in any direction, and override any assigned weights. **Note:** If the symbol is static then its type cannot be changed.

- "Weight" is one of the following:

Value	Description
+ or 1	To add the symbol's value to the specified parent value.
- or -1	To subtract the symbol's value from the specified parent value.
0	To ensure the symbol has no mathematical effect on the specified parent value.

Note: For the Accounts dimension, weight is determined by the credit/debit balance relationship of the parent and child symbols.

Syntax example to change the alternate description of the symbol (a character string value):

Syntax example:

```
MAINTENANCE ON

SET SYMBOL ACCOUNTS NewSales DESCRIPTION FR TO "Nouvelles ventes"
MAINTENANCE OFF
```

Creating an ASCII input file

To change the characteristics of a large number of symbols, specify the parameters for this Database command in an ASCII file.

Create an ASCII file containing the following information for each symbol:

```
SymName{BALANCETYPE{TO{BalanceType

SymName{DESCRIPTION{LanguageCode{TO{SymDesc
SymName{NAME{TO{NewSymName
SymName{PRIORITY{Parent{Priority
SymName[#NumLevels]{ROLLUPSTATUS{TO{StatusType
SymName{SORTOPTION{TO{ChildSortOption
SymName{TYPE{TO{SymType
SymName{VIRTUALSTATUS{TO{StatusType
SymName{WEIGHT{Parent{Weight
```

Syntax (with an ASCII file):

```
SET SYMBOL DimName @FileName
```

where:

- DimName is a dimension containing the symbols.
- FileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

SET SYMBOL ACCOUNTS @C:\Temp\Test.asc
MAINTENANCE OFF
```

See also

- Assign Symbol
- Create Symbol
- Delete Symbol

Set User

Use this command to change the properties of a user.

Syntax regular:

```
SET USER [DomainName\]UserId DESCRIPTION TO "Description"

SET USER [DomainName\]UserId EMAIL TO "Email"
SET USER [DomainName\]UserId FIRSTNAME TO "FirstName"
SET USER [DomainName\]UserId LASTNAME TO "LastName"
SET USER [DomainName\]UserId HOMEPHONE TO "HomePhone"
SET USER [DomainName\]UserId OFFICEPHONE TO "OfficePhone"

SET USER [DomainName\]UserId PASSWORD TO "Password"
```

where:

- DomainName is optional and is the network domain of a Windows-authenticated user.
- UserId is the ID of the user whose properties you want to set.
- Description is a description based on the user's role in the system, enclosed in double quotation marks.
- Email is the user's email address, enclosed in double quotation marks.
- FirstName is the user's first name, enclosed in double quotation marks.
- LastName is the user's last name, enclosed in double quotation marks.
- HomePhone is the user's home phone number, enclosed in double quotation marks.
- OfficePhone is the user's office phone number, enclosed in double quotation marks.
- Password is the user's password. Passwords cannot contain any of the following characters: double quotation marks ("), pipes (|), dollar sign ($), square brackets ([ ), ( ]) or spaces ( ).

Note: For more information on the guidelines for user settings, see the Longview Application Administrator Guide.

Syntax example:

```
SET USER JSmith OFFICEPHONE TO "555-555-5555"
```

Creating an ASCII input file

To make changes for a large number of users, specify the parameters for this command in an ASCII file.

Create an ASCII file containing the following information for each user:

```
[DomainName\]UserId{DESCRIPTION{TO{Description
[DomainName\]UserId{EMAIL{TO{Email
[DomainName\]UserId{FIRSTNAME{TO{FirstName
[DomainName\]UserId{LASTNAME{TO{LastName
[DomainName\]UserId{HOMEPHONE{TO{HomePhone
[DomainName\]UserId{OFFICEPHONE{TO{OfficePhone
[DomainName\]UserId{PASSWORD{TO{Password
```

Syntax (with an ASCII file):

```
SET USER @FileName
```

where:

- FileName is an ASCII file containing the data. It can include a complete or a partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

SET USER @userinfo.txt
MAINTENANCE OFF
```

Set Variable

Use this command to set a variable in Longview Application Framework.

You can also use the SET VARIABLE command in conjunction with the CREATE VARIABLE or CREATE GLOBALVARIABLE command to retrieve the balance type or description for a symbol. For more information, see the Syntax examples.

A variable should be set to a value of the same type it was created as. This can be a string value, a numeric value, or an object instance. For list or range variables the value can be a single item, a list or array of items separated by pipes (|), or a value returned from the CreateList function. The values specified for the Range variables should all be valid symbol names from the same dimension.

If the value assigned to the variable is not compatible with the variable type, an error message displays.

Syntax:

```
SET VARIABLE VariableName[] = Value
```

where:

- VariableName is the name of the variable to be set.
- [] designates the variable is to be created as a list.
- Value is the value that the variable is set to. The value parameter can be a constant string, number, object instance or list provided by the user, or the result of an operation or a function call. The variable name or value can contain references to other variables as shown in the examples below.

Syntax example:

```
CREATE VARIABLE IsUsers[] AS STRING

SET VARIABLE lsUsers = "User1 | User2 | User3 | User4 | User5"

CREATE VARIABLE nCount as NUM
SET VARIABLE nCount = Count ( lsUsers )//counts the elements in IsUsers

CREATE VARIABLE lsUsers[] AS NUM
SET VARIABLE lsUsers = 1|2|3|4|5

CREATE VARIABLE nIndex AS NUM
SET VARIABLE nIndex = 1

CREATE VARIABLE sTemp$nIndex$ AS STRING //creates variable sTemp1
SET VARIABLE sTemp1 = "sTemp1 variable"
```

```
CREATE VARIABLE sTemp AS STRING

SET VARIABLE sTemp = "This is a test string"

SET VARIABLE sTemp = "$sTemp$" + " - " + "Index: $nIndex$"


CREATE VARIABLE rSymbols AS RANGE

SET VARIABLE rSymbols = "SYMBOL1 | SYMBOL2 | SYMBOL3 | SYMBOL4"

SET VARIABLE rSymbols = $rSymbols$ + CreateList ( "SYMBOLS", "DATABASE",
"ACCOUNTS", "CASHT#99" )//appends to the list

For retrieving a symbol's balance type:

CREATE VARIABLE BALTYPE AS STRING

SET VARIABLE BALTYPE = [[SYMBOL, BALANCETYPE, A10021]]
For retrieving a symbol's description:
CREATE VARIABLE DESC AS STRING

SET VARIABLE DESC = [[SYMBOL, DESCRIPTION, A10021]]
```

Note: For the purposes of easily identifying a variable's type (string, number, or range), Longview recommends that the names of string variables be prefixed with s, and that numeric variables have their names prefixed with n, and ranges with r. Furthermore, variables in a list or array should also have their names prefixed with the lowercase letter l (for example, a variable for a list of a string values might be named lsTemp1). This convention also prevents potential conflicts caused by the overlapping of variable names and keywords that might be implemented in Longview Application Framework in future releases. Note that this is only a recommendation and not an actual requirement of the command.

See also

- Create Variable
- Create GlobalVariable

Set WorkflowStatus

Use this command to set the status for Longview Workflow.

Syntax:

```
SET WORKFLOWSTATUS "status" "comment" USING dataspec.lvdsp
```

where:

- status is the workflow status name and can be one of the following values:
  - Not Started
  - Rejected
  - In Progress
  - Submitted for Approval
  - Approved
- comment is an optional comment for the workflow status to attach to the status change. It appears on the Web workflow, in the last comment field, and in the status history log.
- dataspec.lvdsp is the name of a DataSpec file.

Syntax example:

```
SET WORKFLOWSTATUS "Approved" "This is the comment to use" USING
workflow.lvdsp
```

Show DataArea

Use this command to display a DataArea as a Data Grid in a Web browser. This command is specific to Longview Apps.

Note: You cannot use the Show DataArea command in batch mode.

Syntax:

```
SHOW DATAAREA DataArea [USING DataView] [Width Height]
```

where:

- DataArea is the name of the DataArea to display. For information on creating DataAreas, see Create DataArea.
- DataView is optional and specifies the Data View definition (.lvdvw) file to use. You can use a Data View definition to define the layout and functionality in a Data Grid. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. If you do not specify a Data View definition file, the grid displays the first symbol for the fixed dimensions.
- Width is optional and specifies the width (in pixels or as a percent of the window) of the Data Grid. The default window height is 800 pixels. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify WIDTH, you must also specify HEIGHT.
- Height is optional and specifies the height (in pixels or as a percent of the window) of the Data Grid. The default window height is 600 pixels. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify HEIGHT, you must also specify WIDTH.

Syntax example:

```
SHOW DATAAREA SalaryPlanning USING "SalaryPlan\SalaryPlanning.lvdvw" 1000
800
```

Syntax example:

```
SHOW DATAAREA NetInterestIncome USING "NII\NII.lvdvw" 60% 40%
```

Show DataReport

Use this command to display the output of an intercompany report as generated by the DataReport command.

Syntax:

```
SHOW DataReport FileName [Width Height] [Orientation] [PaperSize]
```

where:

- FileName is the name of the ASCII file generated by the DataReport command to display. The path for this file is relative to My Documents\Longview.
- Width is optional and specifies the width (in pixels or as a percent of the window) of the report. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify Width, you must also specify Height.
- Height is optional and specifies the height (in pixels or as a percent of the window) of the report. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify Height, you must also specify Width.

  Note: Width and Height apply only in standalone mode.

- Orientation is optional, is either Landscape or Portrait and must match the orientation specified in the configuration file used in the DataReport command. The default if you do not specify this parameter is Landscape.
- PaperSize is optional, must match the paper size specified in the configuration file used in the DataReport command, and is one of the following:

Letter (default)	Tabloid	DSize	A4	B4
Legal	BSize	ESize	A5	B5
Executive	CSize	A3		

Syntax example:

```
SHOW DataReport DataReportOut.txt
```

Syntax example:

```
SHOW DataReport DataReportOut.txt 80% 80% Portrait Legal
```

Show DataTable

Use this command to display a DataTable object as a Table or a Calendar. This command is specific to Longview Apps.

Note: You cannot use the Show DataTable command in batch mode.

Syntax:

```
SHOW DATATABLE DataTableName USING ViewType ViewDefinition [Width Height]
```

where:

- DataTableName is the name of the DataTable object to display. For information on creating DataTable objects, see Create DataTable.
- ViewType is the view type in which to render the DataTable object, and can be one of the following:
  - TABLE — Renders the specified DataTable object as a Table.
  - HIERARCHY — Renders the specified DataTable object as a Hierarchy.
  - CALENDAR — Renders the specified DataTable object as a Calendar.
- ViewDefinition specifies the Table View definition (.lvtvw), Hierarchy View definition (.lvhvw) or Calendar View definition (.lvcvw) to use to define the layout and functionality in the Table, Hierarchy or Calendar, respectively. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory.
- WIDTH applies only to objects rendered as a Table or Hierarchy. It specifies the width (in pixels or as a percent of the window) of the Table or Hierarchy. The default window height is 800 pixels. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify a WIDTH, you must also use HEIGHT.
- HEIGHT applies only to objects rendered as a Table or Hierarchy. It specifies the height (in pixels or as a percent of the window) of the Table or Hierarchy. The default window height is 600 pixels. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify a HEIGHT, you must also use WIDTH.

Syntax example:

```
SHOW DATATABLE TaskList USING TABLE "TaskPlanning\Tasks.lvtvw" 1000 800
```

Syntax example:

```
SHOW DATATABLE TaskList USING CALENDAR "TaskPlanning\Tasks.lvcvw"
```

Syntax example:

```
SHOW DATATABLE SymbolTree USING HIERARCHY "Symbols.lvhvw
```

See also

- Create DataTable
- Defining DataTable objects
- Creating Table View definition files
- Creating Hierarchy View definition files
- Creating Calendar View definition files

Show Error

Use this command to display the error that the system encountered. The error message is presented in a message prompt.

Syntax:

```
SHOW ERROR [DEBUG]
```

where:

- DEBUG is an optional parameter that indicates that the message prompt should display additional error details.

Syntax example:

```
SHOW ERROR
```

Syntax example:

```
SHOW ERROR DEBU
```

See also

- ThrowError
- OnError

Show Form

Use this command to display a Longview Form. This command is specific to Longview Apps.

Note: You cannot use the Show Form command in batch mode.

Syntax:

```
SHOW FORM USING File
```

where:

- File is the .lvfrm file that defines the form. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For information on creating an .lvfrm file, see Designing Longview Forms.

Syntax example:

```
SHOW FORM USING "NewEmployee.lvfrm"
```

Show HTML

Use this command to display an .html page. This command is specific to Longview Apps.

Note:

- Show HTML does not support HTML5.
- You cannot use the Show HTML command in batch mode.

Syntax:

```
SHOW HTML FileName [[CONTINUE]|[Width Height]]
```

where:

- FileName is the name of the .html file to display. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory.
- CONTINUE is optional. It displays the .html page and then continues with the procedure. This parameter is useful for items that do not require user input such as status indicators. If you do not specify this parameter, the Longview App displays the .html page and halts the procedure until the user initiates an interaction with the page.
- Width can be specified only if you do not specify CONTINUE. It specifies the width (in pixels or as a percent of the window) of any content opened in a window. Indicate a percentage with a percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. The default is 800 pixels. If you specify WIDTH, you must also specify HEIGHT.
- Height can be specified only if you do not specify CONTINUE. It specifies the height (in pixels or as a percent of the window) of any content opened in a window. Indicate a percentage with a percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. The default is 600 pixels. If you specify HEIGHT, you must also specify WIDTH.

Syntax example:

```
SHOW HTML "SalaryPlanning/SalaryPlanningMain.html" 1647 1000
```

Syntax example:

```
SHOW HTML "SalaryPlanning/SalaryPlanningMain.html" CONTINUE
```

Show Message

Use this command to display a message. This command is specific to Longview Apps. The window title of messages and prompts is specified in your .lvapp file with the <description> parameter. The look and feel of messages and prompts is specified in your .lvapp file with the <theme> parameter.

For information, see Configuring Longview Apps.

Note: You cannot use the Show Message command in batch mode.

Syntax example:

```
SHOW MESSAGE "Message" [CONTINUE]
```

where:

- Message is the message to display. To add multiple lines of text, separate lines with a backslash followed by the letter n (\n). For example, "This is line 1\nThis is line 2\nThis is line 3". At this time, you cannot use double quotation marks as part of your message text.
- CONTINUE is optional and displays the message as part of the throbber. If you do not specify this parameter, the message displays in a standard Windows alert box.

Syntax example:

```
SHOW MESSAGE "Data is being submitted to the database" CONTINUE

SHOW MESSAGE "This is line 1\nThis is line 2\nThis is line 3" CONTINUE
```

Show Return

Use this command to signal that the user has finished with the current modal user interface. If not using the SystemAction for Close, Data Grids or Tables in a Longview App must have at least one toolbar button that ultimately executes the SHOW RETURN command, otherwise the user cannot exit the Data Grid. This command is specific to Longview Apps.

Note: You cannot use the Show Return command in batch mode.

Syntax:

```
SHOW RETURN
```

Syntax example:

```
SHOW RETURN
```

Show UI

Use this command to display a tabbed Data Grid in a Web browser. This command is specific to Longview Apps.

Note: You cannot use the Show UI command in batch mode.

Syntax:

```
SHOW UI USING File [Width Height]
```

where:

- File is the .lvui file that defines the elements of the tabbed Data Grid.
- Width is optional and specifies the width, in pixels or as a percentage, of the Data Grid window. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify a Width, you must also specify a Height. If you do not specify both parameters, the window width is 800 pixels.
- Height is optional and specifies the width, in pixels or as a percentage, of the Data Grid window. Indicate a percentage with the percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you specify a Height, you must also specify a Width. If you do not specify both parameters, the window height is 600 pixels.

Syntax example:

```
SHOW UI USING "NetInterestIncome.lvui" 400 300
```

Sleep

Use this command to pause the execution of a procedure for a specific number of seconds.

Syntax:

```
SLEEP Time
```

where:

- Time is the amount of time the procedure will be paused, in seconds.

Syntax example:

```
SLEEP 1200
```

Sort DataTable

Use this command to sort data in a DataTable by a particular column. Sorting does not actually alter the order of items in the DataTable object; it creates a temporary sort index that you can use to improve performance when using the Fetch command with Open Cursor.

If you specify multiple columns, they are sorted in the order in which they appear in the SORT DATATABLE statement.

Note: The Insert DataTableRow and Delete DataTableRows commands inherently use the Sort feature. If you sort prior to using these commands, your sort will be overwritten.

Syntax:

```
SORT DATATABLE DataTable "ColumnName [ASC|DESC], ..."
```

where:

- DataTable is the name of the DataTable object to sort.
- ColumnName is the column to sort by. If the column name contains spaces, you must enclose it in square brackets. For example, [Employee Type].
- ASC sorts the column by ascending alphabetic or numeric order.
- DESC sorts the column by descending alphabetic or numeric order.

Syntax example:

```
SORT DATATABLE EmpSalaries "[Office Location] ASC, [Employee Type] DESC"
```

Syntax example:

```
SORT DATATABLE TopAccounts "State ASC, [Sales Amount] DESC"
```

See also

- Update DataTableRows
- Delete DataTableRows

Stream

Use this command to retrieve all data for the given DataArea. This command should be used in place of the DOWNLOAD command when working with very large amounts of data. A DataArea is created using the Create DataArea command, and initially consists of an empty Structure only. The Stream command is then used in conjunction with the Run (for ExportSpecs) command to populate the DataArea with data from the database. For more information, see Run (for ExportSpecs).

Note: The STREAM command can be used for ODS exports only.

The STREAM command opens a stream for the data. When the Stream command is used in conjunction with the Run (for ExportSpecs) command, the data is downloaded and exported one block at a time. In comparison, the Download (for DataAreas) command downloads all data and builds the data tree before export.

Note: When the STREAM command is used, no data is saved in the created DataArea. This command will download only DataAreas to which the user has access.

Syntax:

```
STREAM DataAreaName [StandardAll|StandardLeafOnly|CTA|LeafData|Validation]
```

where:

- DataAreaName is the name of the DataArea created by the command.
- StandardAll specifies that all standard data will be downloaded. This option is the default for the DOWNLOAD command.
- StandardLeafOnly specifies that only standard leaf data will be downloaded.
- CTA specifies that only CTA data will be downloaded.
- LeafData specifies that only true leaf data will be downloaded. This option differs from StandardLeafOnly in that it does not include calculated data on leaf symbols.
- Validation specifies that the discrepancy calculated by a validation rule is downloaded. You must use the Rule function to specify the server rule associated with this calculation.

Syntax example:

```
STREAM DataArea1 StandardLeafOnly
```

See also

- Create DataArea
- Download (for DataAreas)

- Upload (for DataAreas)

Switch Symbol

Use this command to move a symbol in the database from one parent symbol to another.

When you switch a parent symbol, you switch all its child symbols as well. The new parent symbol can reside in another hierarchy or the same hierarchy as the original parent. You can have a maximum of 47 levels in a hierarchy in the Data Server repository.

This command must be used in conjunction with Maintenance. While it is enabled, no other users can perform maintenance related activities in the Data Server repository. When you are finished, you must remember to disable the Maintenance mode, and thereby allow normal access by other users. For more information, see Maintenance or Exclusive.

Syntax regular:

```
SWITCH SYMBOL DimName SymName PARENT FROM OldParent TO NewParent "Weight"
```

where:

- DimName is a dimension that contains the symbol.
- SymName is the name of the symbol.
- OldParent is the name of the old parent symbol.
- NewParent is the name of the new parent symbol.
- "Weight" is one of the following:

Value	Description
+ or 1	To add the symbol's value to the specified parent value.
- or -1	To subtract the symbol's value from the specified parent value.
0	To ensure the symbol has no mathematical effect on the specified parent value.

Note: For the Accounts dimension, weight is determined by the credit/debit balance relationship of the parent and child symbols.

Syntax example:

```
MAINTENANCE ON
SWITCH SYMBOL ACCOUNTS NewSales PARENT FROM Regional TO National +
MAINTENANCE OFF
```

Creating ASCII input files

To switch a large number of symbols in the database, create a text file specifying the following information for each symbol:

```
SymName{PARENT{FROM{OldParent[TO{NewParent{Weight
```

Syntax (with an ASCII file):

```
SWITCH SYMBOL DimName @SymFileName
```

where:

- DimName is a dimension that contains the symbol.
- @SymFileName is an ASCII file containing the data. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  @"C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path.

Syntax example:

```
MAINTENANCE ON

SWITCH SYMBOL ACCOUNTS @C:\Temp\SymSwitch.asc

MAINTENANCE OFF
```

SyncHier

Use this command to synchronize symbol hierarchies between the Data Server repository and those of other systems. This command generates scripts that when applied, can modify a Longview hierarchy to mirror that of a third-party system or vice-versa.

Note: Symbol maintenance must be turned off before executing this command.

Syntax:

```
SYNCHIER DimName InFile Direction ExecMode DeleteToParent
```

where:

- DimName is a dimension containing the symbols of the hierarchy to be synchronized.
- InFile is the name of an ASCII input file containing symbol information and parent/child relationships. Any line beginning with an asterisk (*) is ignored as a comment. Assuming that the symbol already exists in the target hierarchy, any parameter left unassigned is left unchanged; however, if the symbol does not exist in the target hierarchy (and hence will need to be created), all fields must be included.
- One entry must exist representing the root symbol whereby the hierarchical structure below this root symbol is synchronized between the source and target systems. The root symbol must already exist in the source and target hierarchies.

Each line in the file has the following structure:

```
Parent{SymName{Weight{Desc{Type{RollUps{ChildSortOption{BalanceType
[{Priority]
```

where:

- Parent is the parent symbol for the symbol being defined. There must exist exactly one entry that defines the root symbol. For this entry, this parameter will be left blank.
- SymName is the name of the symbol being defined.
- Weight is the mathematical effect of the child symbol on its parent. If the value of the child symbol will be added, specify +. If the value of the child symbol will be subtracted, specify -. If there is no mathematical effect, specify 0.
- Desc is the description of the symbol in the default language of the server.
- Type specifies the way the symbol rolls up to its parent symbol, and must be one of the following:

Value	Description
Standard	To add the symbol into the parent (for example, monthly expenses added into annual expenses).
Carryforward	When the parent value equals the value of the cell immediately preceding it (for example, cash balance as a running year-to-date, where the year-end amount equals the amount for December).
Static	When the child value has no effect on the parent value – for example, for price symbols. (The parents of such symbols are used only to group symbols, not to aggregate their values.) Static symbols do not roll up in any direction, and override any assigned weights.

- RollUps designates the rollup type and is one of: None|Auto|Dynamic, with the following combinations:

Rollups	Virtual parent	Receive Rollups	Description
NONE	FALSE	FALSE	This setting turns Rollups OFF.
AUTO	FALSE	TRUE	This is the typical configuration for Standard parents.
DYNAMIC	TRUE	TRUE	This is the typical configuration for Virtual parents.

Note: Virtual = TRUE while Rollups = FALSE is an invalid combination and therefore is not available.

- ChildSortOption is one of the following:

Value	Description
Ascending	Sorts the child symbols in ascending order.
Descending	Sorts the child symbols in descending order.
Manual	Allows manual sorting of child symbols.

- BalanceType is one of: Credit|Debit|Neither. Credit and Debit are only used for the Accounts dimension and indicate whether the account is a credit account or a debit account. All other dimensions use Neither.

- Priority is a number that designates a symbol's position in the hierarchy relative to its parent. Symbols are listed under their parent in order of ascending priority, with zeros falling at the bottom of the list.

- Direction specifies where the destination hierarchy resides and can be one of the following:

Value	Description
In	Designates that your Longview system is the destination hierarchy to be synchronized with another system.

Value	Description
Out	Designates that the other system's hierarchy is the target to be synchronized with the Longview hierarchy.

- ExecMode specifies whether or not the process should be simulated or actually implemented when run and can be one of the following:

Value	Description
Execute	Implements the instructions and commits the designated changes to the destination hierarchy. Note that the changes are only committed when the Direction is set to In (for example, when a Longview hierarchy is the destination). Longview does not have the ability to initiate or implement changes on a third-party system, but only to generate scripts which a third-party vendor could use to accomplish the synchronization.
Simulate	Generates an ASCII output file for each type of command and will be given a name in the form: InFile_commandType.ext where InFile is input file name parameter (not including the file extension), .ext is the file extension of InFile, and commandType is one of Add, AddPC, Edit, Delete, Detatch, Assign, or Switch; for example: "infile_Add.txt"

- DeleteToParent specifies how deleted symbols are to be handled. The following may be specified for the DeleteToParent parameter.
  - The word DELETE to signify that the symbol should be completely removed from the database.
  - The name of a symbol where deleted symbols should be attached. The specified symbol must already exist, already be a parent symbol, and not be part of the hierarchy being synchronized.

Syntax example - Command:

```
SYNCHIER Accounts Symbols.txt In Execute Delete
```

Syntax example - Input file entry (root):

```
Input file entry (root):
{AYR11{{Actuals 2011{CarryForward{Auto{Manual{Neither>
```

Syntax example - Input file entry (non-root):

```
Input file entry (non-root):
AYR11{A11Q1{+{Actuals 2011 Q1{ Standard {Auto{Manual{Neither{1
A11Q1{A1101{+{Actuals Jan 2011{Standard{Auto{Manual{Neither{1
A11Q2{A1102{+{Actuals Feb 2011{Standard{Auto{Manual{Neither{2
```

```
A11Q3{A1103{+{Actuals Mar 2011{CarryForward {Auto{Manual{Neither{3
```

See also

- Create Symbol
- Delete Symbol
- Set Symbol

TaxDTNRRec

Use this command to perform the allocation of recognisable deferred tax to a specific rollforward hierarchy.

Note: Before you use the TAXDTNRREC command, you must create a hierarchy of symbols to contain the recognizable portion of deferred tax. This hierarchy must mirror the deferred tax rollforward hierarchy, with the exception that the prefix specified in the TargetPrefix parameter replaces the existing prefix for each symbol. For example, “Rec” replaces “Net”.

Syntax:

```
Calculate TAXDTNRREC SourceRoot TargetPrefix DataArea Method [Rate]
```

where:

- SourceRoot is the root symbol of the source hierarchy to be calculated in the Elements dimension.
- TargetPrefix is the prefix of the target hierarchy in the tax details dimension. TargetPrefix must be the same number of characters in length as the existing prefix in the source hierarchy.
- DataArea is the name of the DataArea containing the source and target for the calculation.

  Note: The DataArea must contain only the symbols required for the calculation.

- Method is the calculation method to be used. The calculation method can be one of the following:

Value	Description
LIFO	“Last In First Out”. If this method is used, the last deferred tax details are the first to be considered unrecognisable. The last deferred tax details are the details at the bottom of the list of leaf symbols in the target hierarchy.
FIFO	“First In First Out”. If this method is used, the first deferred tax details are the first to be considered unrecognisable. The first deferred tax details are the details at the top of the list of leaf symbols in the target hierarchy.
PROPORTIONAL	If this method is used, all deferred tax details are recognised proportionally.

- Rate is the symbol in the Elements dimension that contains the rate to be used for the proportional calculation method. This parameter is used only if the calculation method is specified as PROPORTIONAL.

Syntax example:

```
Calculate TaxDTNRRec NetNEOY_IAS Rec dataArea FIFO
```

TaxFX

Use this command to perform the tax foreign exchange calculation. The TaxFX command replaces the TaxFX_Calc procedure and greatly improves performance.

The following attributes are part of the tax foreign exchange calculation:

- AZTAXFXOVERRIDE
- ZGPNATIVECURRENCY
- AZTAXCTAELEMENT
- AZTAXCTASOURCEELEMENT
- ZFXSOURCECURRENCIES
- ZFXTRANSLATIONS

For more information on setting attributes for Longview Tax, see the *Longview Tax Administrator's Guide*.

Syntax:

```
Calculate TAXFX Currencies ctaElements SourceDataArea RateSourceDataArea
RateMethodDataArea RateOverrideDataArea
```

where:

- Currencies is a range variable, consisting of a list of target currency symbols.
- ctaElements is a range variable, consisting of a list of cumulative translation adjustment (CTA) element symbols.
- SourceDataArea is the name of the DataArea containing the source data for the calculation.
- RateSourceDataArea is the name of the DataArea containing the rate source data.
- RateMethodDataArea is the name of the DataArea containing the rate method data.
- RateOverrideDataArea is the name of the DataArea containing the rate override data.

The following code is an example of how to set up the range variables for translationCurrencies and ctaElements

Example:

```
Create variable currencies as Range
Create variable translationCurrencies as Range
Create variable elements as Range
```

```
Create variable ctaElements as Range
Set variable currencies = createlist("Symbols", "DATABASE", "CURRENCY",
"CURRENCIES###")
Set variable translationCurrencies = ""
Create variable translation as String
For Each currency in currencies
Set variable translation = [[ SYMBOL, ZFXTranslations, $currency$ ]]
If "$translation$" NE ""
Set variable translationCurrencies = $translationCurrencies$ + $currency$
End If
Next
Set variable elements = createlist("Symbols", "DATABASE", "ELEMENTS",
"TRNELE###")
Set variable ctaElements = ""
Create variable cta as String
For Each element in elements
Set variable cta = [[ SYMBOL, AZTaxCTAElement, $element$ ]]
If "$cta$" EQ "TRUE"
Set variable ctaElements = $ctaElements$ + $element$
End If
Next
```

Note: The range variable lists can also be filtered using the FilterList function. For example, Set Variable rCTAAccts = FilterList(rElements, "[[Symbol,AZTaxCTAElement,THIS]] == 'TRUE'").

Syntax example:

```
Calculate TAXFX translationCurrencies ctaElements daSource daRateSource
daRateMethod daRateOverride
```

See also

- Create Variable
- For Each
- FilterList
- If...
- Set Variable

ThrowError

Use this command to manually create an error situation within the procedure, and save a user specified error message to the LVS_ERRORMESSAGE system variable. This initiates Longview Application Framework's default error behavior. If an OnError statement exists within the procedure hierarchy, the default behavior is to trigger it. Otherwise, the default behavior is to terminate the procedure.

Syntax:

```
THROWERROR "String"
```

where:

- String is the error message that is saved into the LVS_ERRORMESSAGE system variable.

Syntax example:

```
THROWERROR "File input error."

THROWERROR "Array out of bounds error."
```

See also

- Show Error
- OnError
- Using the LVS_ERRORMESSAGE system variable

Timer

Use this command to start or stop a timer within a procedure. This command is used in conjunction with the TIMERELAPSED function. The timer is local to the procedure, other timers may exist within nested procedures. The TIMER and TIMERELAPSED command and function can be used to provide useful performance and benchmarking data without the overhead of the HISTORY file.

Syntax:

```
TIMER START

TIMER STOP
```

where:

Value	Description
START	Starts the timer in a procedure document. If TIMER START is called multiple times before a TIMER STOP command, the timer is restarted.
STOP	Stops the timer in a procedure document.

Syntax example:

```
TIMER START


// some code here

SET VARIABLE nTime = TIMERELAPSED()

// some code here

SET VARIABLE nTime = TIMERELAPSED()
TIMER START

// some code here

TIMER STOP
```

Unload

Use this command to remove a document from the document cache.

> **Note:** During development and testing, use this command with caution, since the copies in the document cache are static and are not updated with any changes you make to the original models and procedures.

Syntax:

```
Unload ALL|[[Path\]FileName]
```

where:

- ALL removes all documents from the document cache. If you specify ALL, you do not need to specify any other parameters.
- Path is optional and is the file path for the document. For Longview Apps, the default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For Longview Application Framework, the default path is the lvaf folder in the working directory.
- FileName is the name of the document to unload, including the relevant extension.

Syntax example:

```
Unload ALL
```

Syntax example:

```
Unload Budgeting\SalaryPlan.lvpro
```

See also

- Load
- LoadKar

Unpublish App

Use this command to unpublish a Longview App from a user group. Use separate lines to unpublish a Longview App from multiple groups. If you unpublish the Longview App from all groups, the Longview App is also removed from all categories.

> **Note:** To delete the .lvapp file from the application folder on the data server, you must do so manually. The Unpublish App command does not delete the .lvapp file.

Syntax:

```
UNPUBLISH APP AppName GROUP [Group]
```

where:

- AppName is the name of the Longview App to be unpublished from the group. If the Longview App name includes spaces, enclose it in double quotation marks. You don't have to include the .lvapp extension in the name. Longview App names cannot exceed 100 characters.
- Group is optional and is the name of the user group from which the Longview App is to be unpublished. If no user group is specified, the Longview App is removed from all user groups and all categories.

Syntax example:

```
UNPUBLISH APP SampleApp GROUP Administrators
```

Syntax (with an ASCII file):

```
UNPUBLISH APP @FileName
```

where:

- FileName is an existing ASCII file containing the user groups from which the Longview App is unpublished, with the following syntax:

```
AppName{GROUP{Group
```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
UNPUBLISH APP @"Sample App Groups.txt"
```

See also

- Publish App

Unpublish ProcessMap

Use this command to unpublish a Longview Process Map from a user group. Use separate lines to unpublish a Longview Process Map from multiple groups. If you unpublish the Longview Process Map from all groups, the Longview Process Map is also removed from all categories.

Note: The Unpublish ProcessMap command does not delete the process map.

Syntax:

```
UNPUBLISH PROCESSMAP ProcessID GROUP [Group]
```

where:

- ProcessID is the ID of the Longview Process Map to be unpublished from the group. If the Longview Process Map ID includes spaces, enclose it in double quotation marks. Longview Process Map IDs cannot exceed 100 characters.
- Group is optional and is the name of the user group from which the Longview Process Map is to be unpublished. If no user group is specified, the Longview Process Map is removed from all user groups and all categories.

Syntax example:

```
UNPUBLISH PROCESSMAP SampleProcessMap GROUP Administrators
```

Syntax (with an ASCII file):

```
UNPUBLISH PROCESSMAP @FileName
```

where:

- FileName is an existing ASCII file containing the user groups from which the Longview Process Map is unpublished, with the following syntax:

```
ProcessID{GROUP{Group
```

If the file is in the working folder, you don't need to specify the drive or path. If the file name includes spaces, enclose it in double quotation marks.

Syntax example:

```
UNPUBLISH PROCESSMAP @"Sample ProcessMap Groups.txt"
```

See also

- Publish ProcessMap

Unpublish Template

Use this command to unpublish templates already published to a specific user group or all user groups.

Publishing or unpublishing allocation patterns must be defined to all user groups.

Syntax regular:

```
UNPUBLISH TEMPLATE Group|All TemplateType|All FileName|All
```

where:

- Group is the user group name or All for all user groups. If you specify ALL for the user group, you must specify Pattern for the TemplateType.
- TemplateType may be Report or All.
- TemplateType can be one of the following:

Value	Description
Report	To unpublish report templates.
All	To unpublish all template types

- FileName is the file name of the template (including the .rpt or .kit file extension) or All.

Syntax example:

```
UNPUBLISH TEMPLATE ALL Report "Budget Budget.rtp"
```

Syntax (with an ASCII file):

```
UNPUBLISH TEMPLATE @GroupFile
```

where:

- GroupFile is an ASCII file generated with the Get TemplateInfo command, containing template user group assignments, with the following syntax:

```
Group{TemplateType{FileName
```

See also

- Publish Template

Update DataTableRows

Use this command to update rows in a DataTable object. You can update all rows or only those rows that meet certain criteria.

Syntax:

```
Update DatatableRows in DataTableName SET "[ColumnName1]='Value1'[,...
[ColumnNameN]='ValueN']" [WHERE "[ConditionalColumn1]Operator
ConditionValue1 [AND|OR [ConditionalColumn2] Operator ConditionValue2]"]
```

where:

- DataTableName is the name of the DataTable object that you want to update.
- ColumnName1 is the name of the column to update. If a column name contains spaces, you must enclose it in square brackets. For example, [Phone Number].
- Value1 is the new value for the column. You must enclose string values in single quotes. For example, 'contract'.

  Note: Separate multiple [ColumnName]='Value' statements with a comma.

- WHERE updates the values only for the rows that meet the specified condition.
- ConditionalColumn1 is the column that must contain a specific value in order for the row to be updated. If a column name contains spaces, you must enclose it in square brackets. For example [Employee Type]. You cannot specify a userlist column as a ConditionalColumn.
- Operator is the relational operator and can be one of the following:

Value	Description
EQ or ==	Equal to.
GE or >=	Greater than or equal to.
GT or >	Greater than.
LE or <=	Less than or equal to.
LT or <	Less than.
NE or !=	Not equal to.

- ConditionValue1 is the value that ConditionalColumn1 must contain in order for the row to be updated. Values can be numbers or strings, or, in the case of boolean columns, 1 for TRUE and 0 for FALSE. You must enclose string values in single quotes. For example, 'contract'.

  Note: You can specify two conditions in one And or Or clause.

Syntax example:

```
Update DatatableRows in DataTableName SET "[ColumnName1]='Value1'[,...
[ColumnNameN]='ValueN']" [WHERE "[ConditionalColumn1]Operator
ConditionValue1 [AND|OR [ConditionalColumn2] Operator ConditionValue2]"]
```

Syntax example:

```
Update DatatableRows in Employees SET "[PhoneNumber]='905-940-1510'" WHERE "
[Office Location]= 'Markham'"
```

Syntax example:

```
Update DatatableRows in Salaries SET "[Salary]=35" WHERE "[Contract
Employee]= '1'"
```

See also

- Delete DataTableRows
- Sort DataTable
- Create DataTable

Upload (for DataAreas)

You can use this command to upload changed data from the given DataArea to the database. Initially, a DataArea is created with structure only and the DOWNLOAD command must be used to populate it with data from the database. You must then use the UPLOAD command to send any data changes back to the database. When using the UPLOAD command, the maximum number of open DataAreas allowed at any given time is 31.

Note: The UPLOAD command will upload only data to which the user has write access.

The UPLOAD command depends on the existence and status of the lock ID for the DataArea being uploaded as follows.

Value	Description
DataArea has a valid lock ID	Delta values will be uploaded.
DataArea does not have a valid lock ID, and...	■ one or more locks exist that cover the entire DataArea: ◦ All writable data will be uploaded by reconciling the data in the DataArea with the data in the database. ■ one or more locks exist that cover part of the DataArea: ◦ All writable data covered by the lock will be uploaded by reconciling the data in the DataArea with the data in the database. ■ no lock exists that covers any part of the DataArea: ◦ The UPLOAD will fail. ◦ The UPLOAD will fail.

Syntax:

```
UPLOAD [ADD] DataAreaName ["Comment"] [BatchWait DisableServerRollup]
[UploadBatchSize]
```

where:

- DataAreaName is the name of the DataArea.
- ADD is an optional keyword that will add the data in the upload to the existing data in the database. The expected use of this option is to import a file into an empty (non-downloaded) DataArea so that the upload will add the values to the existing data in the database.

  Note: If ADD is specified, and the DataArea has been downloaded , the DataArea should be cleared before calling UPLOAD ADD. Otherwise, any data in the upload that matches the data in the DataArea will not be added during upload.

- Comment is an optional setting equivalent to having a BatchComment command before the UPLOAD command, and if it is left out the batch will have no comment.
- BatchWait is equivalent to having a BatchWait command before the UPLOAD command. If you specify BatchWait, you must also specify DisableServerRollup. BatchWait can be one of the following:
  - FALSE — Use this value if you want to exit without waiting for the batch to complete.
  - TRUE — Use this value if you want to wait for the batch to complete.

Note: If either BatchWait or DisableServerRollup is specified, the other must also be specified.

Syntax example:

```
CREATE DATAAREA daMaster USING Master.lvdsp

CREATE LOCK daMaster USING Master.lvdsp
UPLOAD daMaster "Upload Batch"


CREATE DATAAREA daMaster USING Master.lvdsp
CREATE LOCK daMaster USING Master.lvdsp
UPLOAD daMaster "Upload Batch" FALSE FALSE


CREATE DATAAREA daMaster USING Master.lvdsp
CREATE LOCK daMaster USING Master.lvdsp
UPLOAD daMaster "Upload Batch" 200000

CREATE DATAAREA daMaster USING Master.lvdsp
CREATE LOCK daMaster USING Master.lvdsp
UPLOAD daMaster "Upload Batch" TRUE FALSE 200000
```

Syntax example used with the BATCHWAIT command:

```
BATCHWAIT ON 30

UPLOAD D1 "Comment" FALSE FALSE
UPLOAD D2 "Comment" 200000
BATCHWAIT OFF

//The first UPLOAD has BatchWait set to FALSE, and thus turned off.
```

```
//The second UPLOAD has BatchWait turned on due to the preceding BATCHWAIT
command.
```

Syntax example used with the SERVERMATH command:

```
SERVERMATH OFF CURRENTSESSION

UPLOAD D1 "Comment" FALSE FALSE

UPLOAD D2 "Comment" 200000

SERVERMATH DEFAULT CURRENTSESSION

//The first UPLOAD has the server rollup on since DisableServerRollup set to
FALSE.

//The second UPLOAD has server rollup off due to the preceding SERVERMATH
command.
```

See also

- BatchComment
- BatchWait
- Create DataArea
- Download (for DataAreas)
- ServerMath

Upload (for DataTables)

You can use this command to upload changed data from the given DataTable object to the relevant App table in the database. Initially, a DataTable is created with structure only and you must use the DOWNLOAD command (or an ImportSpec) to populate it with data. You can then use the UPLOAD command to send any data changes back to the App table in the database. When using the UPLOAD command, the maximum number of open DataTables allowed at any given time is 31.

This command does not apply to virtual DataTables.

Syntax:

```
UPLOAD DataTableName
```

where:

- DataTableName is the name of the DataTable object that contains the data you want to upload.

See also

- Create DataTable
- Download (for DataTables)
- Defining DataTable objects
- Delete DataTableRows

UseInputRelatedSchedules

Use this command to allow the use of input-related schedules, such as the Line Item Details schedule, the Comments schedule, and the File Attachments schedule.

When this command is ON, the following commands take line item details, comments, and attachments into account:

You can also set UseInputRelatedSchedules to LineItemDetails, Comments, and/or Attachments. For more information, see Syntax.

Command	Effect of UseInputRelatedSchedules
Create DataArea	If UseInputRelatedSchedules is in use, when you created the DataArea, three additional DataAreas are created for line item details, comments, and attachments.
Create Lock	If UseInputRelatedSchedules is in use, when you created a lock on the DataArea, three additional locks are created on the DataAreas for line item details, comments, and attachments.
Delete Lock	If UseInputRelatedSchedules is in use, when you delete the lock for the DataArea, three additional locks on the DataAreas for line item details, comments, and attachments are also deleted.

If you want users to be able to access line item details, comments, and attachments in a Longview Data Grid, you must set this command to ON or to the appropriate keyword in the related procedure. You can use multiple USEINPUTRELATEDSCHEDULES commands in the same procedure.

Syntax:

```
USEINPUTRELATEDSCHEDULES Option
```

where:

- Option is one of the following:

Value	Description
ON	Allows users to access line item details, comments, and attachments in Data Grids. **Note:** File Attachments must be selected on the Application page of the Server Configuration dialog in Longview Server Manager to allow users to access attachments in Data Grids. For more information, see the Longview Server Manager Guide.
OFF	Does not allow users to access line item details, comments, or attachments. This is the default if you do not specify USEINPUTRELATEDSCHEDULES in your procedure.
Comments	Allows users to access comments.
LineItemDetails	Allows users to access line item details.
Attachments	Allows users to access attachments. **Note:** File Attachments must be selected on the Application page of the Server Configuration dialog in Longview Server Manager to allow users to access attachments in Data Grids. For more information, see the Longview Server Manager Guide.

The following code uses a singular USEINPUTRELATEDSCHEDULES command for the first DataArea and multiple USEINPUTRELATEDSCHEDULES commands for the second DataArea in a procedure document:

Example:

```
// Create, lock, and download first DataArea with Comments, LineItemDetails,
and Attachments schedules

USEINPUTRELATEDSCHEDULES ON

Create DATAAREA SampleDA1 USING "Sample1.lvdsp"

Create LOCK USER USING "Sample1.lvdsp"

Download SampleDA StandardALL


// Create, lock, and download second DataArea with Comments and
LineItemDetails schedules

USEINPUTRELATEDSCHEDULES OFF

USEINPUTRELATEDSCHEDULES Comments

USEINPUTRELATEDSCHEDULES LineItemDetails

Create DATAAREA SampleDA2 USING "Sample2.lvdsp"

Create LOCK USER USING "Sample2.lvdsp"
```

```
Download SampleDA2 StandardALL


// Show the two DataAreas with different input-related schedule settings in
a tabbed Data Grid

SHOW UI using Sample.lvui
```

UseListDelimiter

Use this command to convert the delimiter in a STRING list or NUM list variable at run time. You can use the LVS_ListDelimiter system variable to return the current delimiter.

For more information, see “Using the LVS_LISTDELIMITER system variable”.

This command is particularly useful when you are using a STRING list variable with the at sign (@) as the delimiter. In order for some functions to use these list variables, the at sign (@) must be converted to a pipe (|). This applies to the following functions/keywords:

- the Values keyword in the RowLimit (for DataTable definitions) function
- the ComboBox function for Longview Forms
- the RadioGroup function for Longview Forms

Syntax:

```
UseListDelimiter delimiter
```

where:

- delimiter is the delimiter to use in the current instance and can be either the @ sign (@) or the pipe (|).

Syntax example:

```
UseListDelimiter |
```

Syntax example:

```
UseListDelimiter @
```

While... End While

Use this command to repeat a series of instructions until the specified condition is no longer true.

The While command tests the initial condition first. If it equals zero, your system does not carry out the instructions.

Before running the While command, specify the initial value of the variable as zero or non-zero. Somewhere in the loop, change the value of the variable. This process continues until the value of the variable reaches zero.

Syntax:

```
WHILE <Conditions>

Instructions
END WHILE
```

where:

- Instructions is a set of instructions to be repeated as long as the logical value of the variable is not false (that is, not zero).
- Conditions are any of the following;

Value	Description
AND	When each of several conditions must be true.
OR	When one of several conditions must be true.
EQ	Equal to.
GE	Greater than or equal to.
GT	Greater than.
LE	Less than or equal to.
LT	Less than.
NE	Not equal to.

Syntax example:

```
CREATE VARIABLE Loop AS NUM

SET VARIABLE Loop = 1

WHILE $Loop$
<commands to execute>
SET VARIABLE Loop = $Loop$ + 1
```

```
IF $Loop$ = 10
SET VARIABLE Loop = 0
END IF

END WHILE
```

See also

- If...

Write

Use this command in conjunction with the Create Document command to write to a document in the document cache. For example, you may need to create a DataSpec document in which the number of temporary symbols is unknown until run time.

You can write to a document from a variable or from a physical file. When you write to a document from a variable, you are restricted by the maximum variable size of 4000 characters. To circumvent this restriction, you can write to a document from a physical file.

Syntax:

```
Write DocName VARIABLE|FILE VariableName|FileName [APPEND]
```

where:

- DocName is the name of the document object to write to, and must match the DocName as specified in the related Create Document command, including the path and extension.
- VARIABLE writes the contents of a variable to the document. If you use this option, you must specify the VariableName.
- FILE writes the contents of a physical file to the document. If you use this option, you must specify the FileName.
- VariableName is the name of the variable. You can specify a STRING, NUM, or RANGE variable. STRING list and NUM list variables are also supported.
- FileName is the name of the physical file. Optionally, you can include a file path. If the path includes spaces, you must enclose it in double quotation marks. For Longview Apps, the default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For Longview Application Framework, the default path is the lvaf folder in the working directory.
- APPEND is optional and specifies to append the variable or file contents to the document. If you do not specify this option, the document is cleared before writing.

Syntax example from a variable, with variable resolved:

```
CREATE DOCUMENT MyDataView.lvdvw

CREATE VARIABLE sOutPut as STRING

Set VARIABLE sOutPut = "InsertSeparator BLANK, ACCOUNTS, Validation$nLoop$,
Before, 1, $nRowSeparatorHeight$"

WRITE MyDataView.lvdvw VARIABLE sOutput
```

Syntax example from a variable, with variable unresolved:

```
Create DOCUMENT MyDataView.lvdvw


CREATE VARIABLE sOutPut as STRING

Set VARIABLE sOutput = "InsertSeparator BLANK, ACCOUNTS, Validation” + “$” +
“nLoop” + “$” + “, Before, 1,” + “$” + “nRowSeparatorHeight” + “$”

WRITE MyDataView.lvdvw VARIABLE sOutput
```

Syntax example from a file and appended:

```
CREATE DOCUMENT MyDataView.lvdvw

Write MyDataView.lvdvw FILE
"C:\Testing\ApplicationFramework\DataView1.lvdvw" APPEND
```

See also

- Create Document
- Save
- Unload

Using Procedure Functions

This section contains the Longview Application Framework functions you can use in a procedure. Click the links in the following table for detailed usage and syntax information.

For information on using Longview Application Framework functions in a model, see Using Model Functions.

Abs	GetSymbolDescription	IsParentSymbol	ScheduleExists
AttributeExists	GetSymbolModifiedTimeStamp	IsReadOnly (DataArea)	ShowFileChooser
Avg	v26.2 GetSymbolParent	IsReadOnly (Symbol)	
Count	v26.2 GetSymbolPartitionIndex	IsRowLimitExceeded	
CreateList	GetSymbolPriority	v26.2 IsWorkflowLocked	ShowPrompt
Exp	GetSymbolRollupStatus		ShowSymbolSelector
	GetSymbolSortOption	JDate	StrContains
FileExists	GetSymbolType	JDiff	StrFind
FilterList	GetSymbolVirtualStatus	JERollover	StrLeft
	GetSymbolWeight	JValue	StrLength
GetAppConfig	GetTime	ListAppend	StrLower
GetAttributeDescription	GetType	ListFindAll	StrReplace
GetAttributeType	GetWorkflowInfo	ListFindFirst	StrRight
GetCalcJEStatus		ListRemoveDuplicates	StrToNum
GetDataTableColumnIndex	GetWorkflowState	ListSort	StrTrim
GetDataTableRow	GetWorkflowStatus	Log	StrTrimL
GetDate	GetWorkflowUserRole	Max	StrTrimR
GetDocumentSize	GroupExists	Min	StrUpper
GetFileProperties	If	Neg	StrValue
GetFileSize	IsAreaBatchStatus	NumToStr	SubStr
GetLockComment	IsAreaEventStatus	Operators (arithmetic)	Sum
GetLockUser	IsBatchMode	Operators (relational)	SymbolExists
GetMaintenanceUser	IsChanged	Operators (conditional)	TimerElapsed

GetNextPeriod	IsConnected	Pos	UserExists
GetOpenPeriod	IsExclusiveOn	PropertyExists	Value
GetPreviousPeriod	IsHierarchicalStep	Round	VariableExists
GetResourceString	IsKeyword	RuleExists	
GetServerWarning	IsLocked	SetDataTableCell	
GetStatus	IsMaintenanceOn	SetWorkflowStatus	
GetSymbolBalanceType	IsMaintenanceUser		
GetSymbolCreatedTimeStamp	IsName		

Abs

Use this function in an equation to evaluate an expression and to calculate its absolute value. Absolute value is the expression of a number as a positive value, regardless of whether it is actually positive or negative.

You may find the Abs function useful for learning the deviation between planned and actual results.

Syntax:

```
ABS (Expression|Value)
```

where:

- Each Expression is a mathematical combination of constants, variables, and functions. In Model Documents, Expression can also contain symbols.
- Value is a quoted string or an expression that resolves to a string.

Syntax example:

```
AVar = ABS (75000 - 125000)
```

AttributeExists

Use this function to determine whether the specified attribute exists in the database. A return value of 1 means the attribute exists in the system; a value of 0 means that it does not.

Syntax:

```
ATTRIBUTEEXISTS ("AttrClass", "AttrName")
```

where:

- AttrClass is the attribute class that contains the attribute to search for.
- AttrName is the name of the attribute.

Syntax example:

```
AttributeExists("SYSTEM", "AZGPCurrentPeriodPAC")
```

Avg

Use this function in an equation to calculate the average value of a set of numbers.

For information on using this function in a model, see Avg.

Syntax:

```
AVG (Value|Expression[,Value|Expression])
```

where:

- Value is a quoted string or an expression that resolves to a string.
- Expression is a data intersection to be resolved as a text string. If no expression is provided, the current symbol calculated in the calculation block is used.

Syntax example:

```
Aver = AVG (471,(9*875.9))
```

Count

Use this command to return the number of items in a list. This command can be used with either a list variable, symbol, or a file name.

Note: Only those symbols to which the user has access will be included in the return list.

Syntax:

```
COUNT (Variable)

COUNT ("DimName", "SymName#")
COUNT ("ScheduleName", "DimName", "SymName#")
COUNT ("FileName")
```

where:

- Variable is the name of a variable.
- DimName is the name of a dimension containing the symbols.
- Schedule is the name of a schedule.
- SymName# is the name of a symbol and any relevant symbol hierarchy specification.
- FileName is the name of a file. In this case, the COUNT function returns the number of lines in a text file. This may be useful if you want to verify the record counts output to log files generated.

  Note: If only one parameter is specified, the function will check the validity of the parameter in the following order:

  - valid variable name
  - valid file in memory
  - valid physical file name

Syntax example:

```
CREATE VARIABLE Count1[] AS String

CREATE VARIABLE CNT as NUM
SET VARIABLE Count1 = "P01ytd | p02ytd"
SET VARIABLE CNT = count ("Count1")
```

Syntax example:

```
CREATE VARIABLE List

CREATE VARIABLE listcount

CREATE VARIABLE listcount2

CREATE VARIABLE listcount3

SET VARIABLE List = CreateList("Symbols", "Database" "Accounts",
"TrialBalance###")

SET VARIABLE listcount = Count(List)

SET VARIABLE listcount2 = Count(Accounts, TrialBalance###)
```

CreateList

Use this Longview Application Framework function to return a list of item names to a list variable.

The CreateList function requires the use of Set Variable List to define the variable to which the function will be returning data.

See the syntax examples below.

Syntax

The CreateList function can return a list of any of the following items:

List item	Description
Attributes	All Attributes of a particular class in the Data Server repository. For more information, see Attributes.
Columns	Properties of all columns in the specified Data Table. **Note:** This does not work on Virtual data tables.
Column Names	A list of column names in the specified Data Table. **Note:** This does not work on Virtual data tables.
Dimensions	Dimensions in the Data Server repository. For more information, see Dimensions.
Files	All files and/or folders in the specified local path. For more information, see Files.
Groups	User groups in the Data Server. For more information, see Groups.
Journal Entry Subcategories	Journal entry subcategories in the Data Server repository. For more information, see Journal entry subcategories.
Populated data	Populated elements in the selected DataArea. For more information, see Populated data.
Properties	Level-one properties of the specified object. For more information, see Properties.
Server rules	Server rules in the Data Server repository. For more information, see Server rules.
Schedules	Schedules in the Data Server repository. For more information, see Schedules.
Schedule Dimensions	Schedule dimensions in the Data Server repository. For more information, see Schedule Dimensions.
Schedule Symbols	Schedule symbols in the Data Server repository. For more information, see Schedule Symbols.
Symbols	Descendent symbols in the Data Server repository. In this case, the CreateList command will return a list of the symbols the user has access to in the specified DataArea. For more information, see Symbols.
Users	User IDs in the Data Server. For more information, see Users.

List item	Description
Symbol Descriptions	Descendent symbol descriptions in the Data Server repository. In this case, the CreateList command will return a list of the symbol descriptions for the symbols the user has access to in the specified DataArea. For more information, see Symbol Descriptions.
Schedule Symbol Descriptions	Schedule symbol descriptions in the Data Server repository. For more information, see Schedule Symbol Descriptions.
Workflows	Active Workflow processes in the Data Server Repository.
WorkflowSteps	Names of workflow steps for a specified Workflow process.

Attributes

Lists all Attributes in the specified Attribute class.

Syntax:

```
CREATELIST ("ATTRIBUTES","AttrClass")
```

where:

- AttrClass is the Attribute class. Select one of the following:

Value	Description
SYSTEM	Describes the entire Longview system at the highest level. Attributes of this Attribute class specify system-wide characteristics. There is only one object in the SYSTEM Attribute class, called DBDefault.
USER	Describes the Attributes of a particular user. Each user is an object in the USER Attribute class.
SYMBOL	Describes the characteristics of individual symbols. Each symbol is an object in the SYMBOL Attribute class.

Syntax example:

```
CREATE VARIABLE List1[] AS STRING
```

SET VARIABLE List1 = CREATELIST ("ATTRIBUTES","USER")

The output of the list appears as:

```
AttrName
```

Columns

Lists column information for all columns in the specified Data Table. The list is returned as an OBJECTLIST and lists the column name, Longview column data type, string length (if applicable) and whether the column is nullable for each column.

```
CREATELIST ("COLUMNS", "dataTableName")
```

The result will be an object list with the following properties:

- Name returns the name of the nth column
- Type returns the Longview column datatype for the nth column. Valid values are:
  - AutoInteger
  - AutoUser
  - Boolean
  - Date
  - Dimension
  - File
  - Integer
  - Number
  - String
  - Symbol
  - User
  - UserList
- StringLength returns the string length of the nth column, if applicable
- Nullable returns whether or not the nth column is nullable. Valid values are:
  - TRUE
  - FALSE

Syntax example:

```
SET VARIABLE ObjectList1 = CREATELIST ("COLUMNS", "SalaryTable")
```

The output of the object list appears as:

```
EmpID, Number, 0, FALSE
FirstName, String, 256, TRUE
LastName, String, 256, TRUE
Position, String, 256, TRUE
Manager, String, 256, TRUE
HireDate, Date, 0, TRUE
Salary, Number, 0, TRUE
```

ColumnNames

Lists all columns in the specified Data Table. The list is returned as a STRINGLIST.

```
CREATELIST ("COLUMNNAMES", "dataTableName")
```

Syntax Example

```
SET VARIABLE List1 = CREATELIST ("COLUMNNAMES", "SalaryTable")
```

The output of the list appears as:

```
EmpID, FirstName, LastName, Position, Manager, HireDate, Salary
```

Dimensions

Lists all dimensions in the database.

```
CREATELIST ("DIMENSIONS")
```

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("DIMENSIONS")
```

The output of the list appears as:

```
DimName
```

Files

Lists all the files and/or folders in the specified local path in a variable of type OBJECTLIST.

```
CREATELIST ("FILES", "localPath" [,"FILE|FOLDER|ALL"])
```

where:

- "localPath" is the local path to find all the files and/or folders to list. The local path can be set to:
  - "."
  - ".."
  - absolute path
  - relative path
- "FILE|FOLDER|ALL" indicates what to list. ALL is the default if not specified:
- FILE indicates that only files are included in the list.

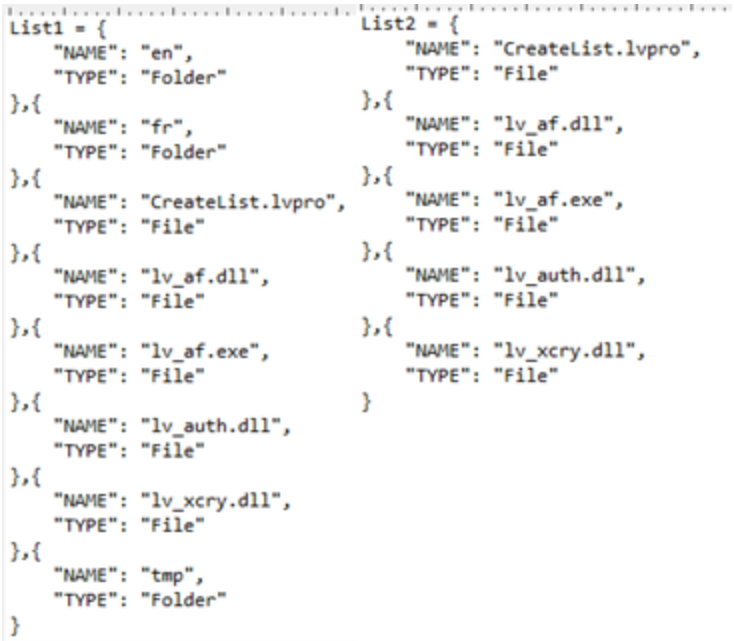
- FOLDER indicates that only folders are included in the list.
- ALL indicates both folders and files are included in the list.

Syntax example:

```
CREATE VARIABLE List1[] AS OBJECT
```

CREATE VARIABLE List2[] AS OBJECT SET VARIABLE List1 = CREATELIST ("FILES",".") SET VARIABLE List2 = CREATELIST ("FILES",".","FILE")

The output of the list appears as:



```
List1 = {
    "NAME": "en",
    "TYPE": "Folder"
},{
    "NAME": "fr",
    "TYPE": "Folder"
},{
    "NAME": "CreateList.lvpro",
    "TYPE": "File"
},{
    "NAME": "lv_af.dll",
    "TYPE": "File"
},{
    "NAME": "lv_af.exe",
    "TYPE": "File"
},{
    "NAME": "lv_auth.dll",
    "TYPE": "File"
},{
    "NAME": "lv_xcry.dll",
    "TYPE": "File"
},{
    "NAME": "tmp",
    "TYPE": "Folder"
}
```

```
List2 = {
    "NAME": "CreateList.lvpro",
    "TYPE": "File"
},{
    "NAME": "lv_af.dll",
    "TYPE": "File"
},{
    "NAME": "lv_af.exe",
    "TYPE": "File"
},{
    "NAME": "lv_auth.dll",
    "TYPE": "File"
},{
    "NAME": "lv_xcry.dll",
    "TYPE": "File"
}
```

Groups

Lists all user groups.

```
CREATELIST ("GROUPS")
```

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("GROUPS")
```

The output of the list appears as:

```
User Group
```

Journal entry subcategories

Lists Journal entry subcategory information in a variable of type STRINGLIST or OBJECTLIST.

```
CREATELIST ("JESUBCATEGORIES")
```

Syntax example:

```
CREATE VARIABLE List1[] AS OBJECT
```

SET VARIABLE List1 = CREATELIST ("JESUBCATEGORIES")

The result will be a list of strings or objects with the following properties:

- ID returns the subcategory ID
- Name returns the description of the subcategory
- Category returns either "STANDARD" or "ELIMINATION"
- ScheduleName retruns the schedule name, if applicable
- Reclassification returns 1 or 0.

Populated data

Lists the populated elements in the specified DataArea. Populated data excludes zeros and null values.

```
CREATELIST ("PopulatedData", "DataArea", "NUM|STRING|BOTH", "VALUES|DimName"
[, "dataspec.lvdsp"])
```

where:

- "DataArea" is the DataArea from which the data is extracted.
- "NUM|STRING|BOTH" indicates the types of values to be used in the list:
  - NUM indicates that only numeric values are used.
  - STRING indicates that only string values are used.
  - BOTH indicates that both numeric values and string values are used.

  Note: BOTH returns a list of strings, with any numeric values rendered as strings with full precision, if VALUES is used for the VALUES|DimName parameter.
- "VALUES|DimName" indicates whether data values or symbol names are returned in the list:
- VALUES returns the actual populated data values as a list of numbers or strings as specified by the NUM|STRING|BOTH parameter.

- DimName returns the names of the symbols in that dimension as a list of strings, corresponding to the populated data values.
- "dataspec.lvdsp" is an optional data spec that specifies the subset of the DataArea from which the data is extracted. If not specified, data is extracted from the entire DataArea.

Syntax example:

```
SET VARIABLE result = CREATELIST("PopulatedData","myWorkspace", "BOTH",
"VALUES","dataspec.lvdsp")
```

The output of the list appears as:

```
Result= 3.000000000,26.000000000,60.000000000,string1…
```

Properties

Lists the level-one properties of the specified object. The list is returned as a stringlist. You may use this function against object variables or object properties.

```
CREATELIST(PROPERTIES, objectVariableName)
CREATELIST(PROPERTIES, objectVariableName.objectPropertyName)
```

where:

- objectVariableName is the name of an object variable.
- objectVariableName.objectPropertyName is the name of an object and specific property.

> **Note:** You may also specify an object within an array of objects, in the form objectVariableName[index] or objectVariableName.objectPropertyName[index].

Example:

```
Given the following object:
{ "Name": "A17_06_YTD",
"IsFirst": 0,
"IsLast": 0,
"IsOpen": 0,
"IsPA": 0,
"IsYTD": 1,
```

```
        "RelatedPeriod": {

       "YTD": "A17_06_YTD",
              "PA": "A17_06",

       "Open": "A17_Open",

       "Previous": "A17_05_YTD",
              "Next": "A17_07_YTD"

       }
}
SET VARIABLE Result = CREATELIST (PROPERTIES, myObject)
```

The output of the list appears as:

```
Result = Name, IsFirst, IsLast, IsOpen, isPA, IsYTD, RelatedPeriod
```

Server rules

Lists all server rule IDs for the specified rule type.

```
CREATELIST ("SERVERRULES","RuleType")
```

Note: The list returns rule numbers, so the variable (List2, in this case) must be declared as type NUM.

where:

- RuleType is the type of server rule in the list. Select one of the following:

Value	Description
ALL	To include the ID of all types of server rules.
EVENT	To include the ID of event rules only.
MODELS	To include the ID of model rules only.
QUERY	To include the ID of query rules only.
ROLLUPS	To include the ID of rollup rules only.
VALIDATIONS	To include the ID of validation rules only.

Syntax example:

```
CREATE VARIABLE LIST2[] AS NUM
```

SET VARIABLE List2 = CREATELIST ("SERVERRULES","MODELS")

The output of the list appears as:

```
RuleID
```

Schedules

Lists all schedule names.

```
CREATELIST ("SCHEDULES")
```

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SCHEDULES")
```

The output of the list appears as:

```
SchedName
```

Schedule Dimensions

Lists all dimension names for the specified schedule.

```
CREATELIST ("SCHEDULEDIMENSIONS","SchedName")
```

where:

- SchedName is the schedule name.

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SCHEDULEDIMENSIONS","PFDETAIL")
```

The output of the list appears as:

```
SchedDimName
```

Schedule Symbols

Lists all symbol names for the specified schedule dimension in the specified schedule.

```
CREATELIST ("SCHEDULESYMBOLS","SchedName","SchedDim")
```

where:

- SchedName is the schedule name.
- SchedDim is the name of a dimension defined within the schedule.

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SCHEDULESYMBOLS","PFDETAIL","ListItems")
```

The output of the list appears as:

```
SchedSymName
```

Symbols

Lists all descendent symbol names for the specified dimension.

```
CREATELIST ("SYMBOLS","Source","DimName","SymName[HierarchySpecification]")
```

where:

- Source is either DATABASE or the name of a DataArea.
- DimName is the dimension in which the symbols reside.
- SymName is the name of a symbol in the dimension. You can also type ROOT to specify all root symbols in the dimension.
- HierarchySpecification is a specification determining the structure under SymName to retrieve.

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SYMBOLS","DATABASE","TIMEPER","AYR05#99")
```

The output of the list appears as:

```
SymName
```

Schedule Symbol Descriptions

Lists the symbol descriptions for the specified schedule dimension in the specified schedule.

```
CREATELIST("SCHEDULESYMBOLDESCRIPTIONS","SchedName","SchedDim")
```

where:

- SchedName is the schedule name.
- SchedDim is the name of a dimension defined within the schedule.

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SCHEDULESYMBOLDESCRIPTIONS
","PFDETAIL","ListItems")
```

The output of the list appears as:

```
SchedSymDesc
```

Symbol Descriptions

Lists all symbol descriptions for the specified dimension.

```
CREATELIST ("SYMBOLDESCRIPTIONS","Source","DimName","SymName[NumLevels]")
```

where:

- Source is either DATABASE or the name of a DataArea.
- DimName is the dimension in which the symbols reside.
- SymName is the name of a symbol in the dimension.
- NumLevels is a number representing the number of hierarchy levels below SymName.

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("SYMBOLDESCRIPTIONS
","DATABASE","TIMEPER","AYR05#99")
```

The output of the list appears as:

```
SymDesc
```

Users

Lists all user names.

```
CREATELIST ("USERS")
```

Syntax example:

```
SET VARIABLE List1 = CREATELIST ("USERS")
```

The output of the list appears as:

```
UserName
```

Workflows

Lists the active workflow processes. The list is returned as a stringlist.

```
CREATELIST(WORKFLOWS)
```

Note: If the Use Workflow server configuration parameter is FALSE, this function will return an empty list.

Syntax example:

```
CREATE VARIABLE WFProcesses[] AS String
SET VARIABLE WFProcesses = CREATELIST(WORKFLOWS)
```

WorkflowSteps

Lists the workflow steps for a specified Workflow process. The list is returned as a stringlist.

```
CREATELIST(WORKFLOWSTEPS, Process)
```

Where:

- Process is the name of an active Workflow process. If the process name contains spaces, enclose it in double quotation marks (" ").

Note: If the Use Workflow server configuration parameter is FALSE, this function will return an empty list.

Syntax example:

```
CREATE VARIABLE WFProcessSteps[] AS String
SET VARIABLE WFProcessSteps = CREATELIST(WORKFLOWSTEPS, MonthEndProcess)
```

Exp

Use this function in an equation to raise a base value to an exponential power.

For information on using this function in a model, see Exp.

Syntax:

```
EXP (Base, Power) (values only)

EXP (Power) (calculated as e^power)
```

where:

- Base is a symbol or a constant. The default is the natural logarithm, e (= 2.71828). If Base is a symbol, your system determines the logarithm for each cell in SymName from the corresponding cell in Base.
- Power is a symbol or a constant. If Power is a symbol, your system takes the exponent for each cell in SymName from the corresponding cell in the symbol specified for Power. If you do not specify Base, your system assumes the base of the natural logarithm, e (= 2.71828).

Syntax example:

```
EXP (10, 2)
```

FileExists

Use this function to determine whether the specified file exists in the specified directory.

Syntax:

```
FileExists("FileName")
```

where:

- FileName is the name of the file, including the extension, to search for. In batch mode, the file name must be relative to the current directory. For Longview Smart Client, the file name must be relative to the working directory. You can also specify a full path to search in an alternate directory.

Syntax example:

```
If FileExists("Reports.xlsx")

Set VARIABLE x = FileExists("Reports.xlsx")

// where: x is a variable of type NUM. x returns 1 if the file exists and 0
if it does not.
```

FilterList

Use this function to create a new list which is filtered in a specified condition. This function is only available in a procedure as part of the SET VARIABLE command.

Syntax:

```
FILTERLIST ("VariableName", "Condition")
```

where:

- VariableName is the name of a list variable.
- Condition is a condition that evaluates to a TRUE/FALSE (1/0) result. It can also be an Attribute-based filter (if the list contains objects that have Attributes, for example, a list of users, user groups, or symbols).

Syntax example:

```
CREATE VARIABLE X[] AS STRING


CREATE VARIABLE Y[] AS STRING

SET VARIABLE X = CreateList ("SYMBOLS", "DATABASE", "Accounts",
"TrialBal###")

SET VARIABLE Y = FilterList ("X", "[[SYMBOL, ZGPNativeCurrency,
THIS]]==CAD'')
```

GetAppConfig

Use this function to retrieve an app property value from a simple XML element in the .lvapp file, then return the value as a string. This function is specific to Longview Apps. Currently, retrieving nested elements such as <kars> is not supported.

Note: You cannot use the GetAppConfig function when running in batch mode.

Syntax:

```
GetAppConfig ("NameOfProperty")
```

where:

- NameOfProperty is the name of the property whose value you wish to retrieve. This value is not case-sensitive. You can also retrieve the name of the App by setting this parameter to "name".

Syntax example:

```
Create Variable sValue as STRING
Set Variable sValue = GetAppConfig("description")
Show Message "The app description is $sValue$"
```

GetAttributeDescription

Use this function to return the description of the specified attribute. The description is returned as a string.

Syntax:

```
GetAttributeDescription( "AttrClass", "AttrName" [, "LanguageCode"])
```

where:

- AttrClass is the Attribute class. Select from the following:

AttrClass	Description
SYSTEM	Describes the entire system at the highest level. Attributes of this Attribute class specify system-wide characteristics.
USER	Describes the Attributes of a particular user.
SYMBOL	Describes the characteristics of individual symbols.

- AttrName is the name of the attribute whose description you wish to retrieve.
- LanguageCode is the optional language code, with a default value of EN.

Syntax example:

```
SET VARIABLE attrDescription = GetAttributeDescription( "SYSTEM",
"ASReportingPeriod", "EN")
```

GetAttributeType

Use this function to identify the attribute value type of a specified attribute. The type is returned as a string.

Possible return values include the following:

- DATE
- DOUBLE
- INTEGER
- STRING
- SYMBOL
- DATELIST
- DOUBLELIST
- INTEGERLIST
- STRINGLIST
- SYMBOLLIST

Syntax:

```
GetAttributeType( "AttrClass", "AttrName" )
```

where:

- AttrClass is the Attribute class. Select from the following:

AttrClass	Description
SYSTEM	Describes the entire system at the highest level. Attributes of this Attribute class specify system-wide characteristics.
USER	Describes the Attributes of a particular user.
SYMBOL	Describes the characteristics of individual symbols.

- AttrName is the name of the attribute whose type you wish to retrieve.

Syntax example:

```
SET VARIABLE attrType = GetAttributeType( "SYSTEM", "ASReportingPeriod")
```

GetCalcJEStatus

Use this function to retrieve the status of a calculated journal entry. The variable type is returned as a Num.

Possible return values include the following:

- 0 - Does Not Exist - The calculated journal entry does not exist or cannot be retrieved. Non-shared calculated journal entries will not be retrieved for a user that does not have access to the JE.
- 1 - No Status - The calculated journal has not been validated or posted. This value is not expected for calculated journal entries.
- 2 - Posted - The calculated journal entry has been posted permanently
- 3 - Validated - The calcualted journal entry as been validate
- 4 - Errors - The calculated journal entry did not validate. This values is not expected for calculated journal entries, as calculated journal entries are validated when they are created.
- 5 - Review - The calculated journal entry has been posted temporarily

Syntax:

```
GETCALCJESTATUS (AppID, PostPeriod, CreationPeriod, Type)
```

where:

- AppID is the Application ID of the calculated journal entry.
- PostPeriod is the symbol name of the posted period of the calculated journal entry.
- CreationPeriod is the symbol name of the creation period of the calculated journal entry.
- Type is the type of calculated journal entry and can be one of the following:
  - CURRENTPERIOD
  - RESTATEMENT
  - PPA
  - FUTUREPERIOD

Syntax example:

```
CREATE VARIABLE JEStatus as NUM

SET VARIABLE JEStatus = GETCALCJESTATUS (CalcJE1, P03YTD, P03YTD,
CURRENTPERIOD)
```

GetDataTableColumnIndex

Use this function to retrieve the column index of a specified column in a DataTable object. If the column specified does not exist, the return result is -1.

Syntax:

```
GetDataTableColumnIndex (DataTableName, "ColumnName")
```

where:

- DataTableName is the name of the DataTable object in which to get the column index.
- ColumnName is the name of the column for which to get the index, enclosed in double quotation marks ("").

Syntax example:

```
Create Variable colIndex as NUM

Set Variable colIndex = GetDataTableColumnIndex (SalaryTable, "StartDate")
```

GetDataTableRow

Use this function to retrieve the row contents in a DataTable object. The row to retrieve is specified by the row index.

Syntax:

```
GetDataTableRow (DataTableName, RowIndex)
```

where:

- DataTableName is the name of the DataTable object in which to get the row.
- RowIndex is the row index of the cell to get.

Syntax example:

```
Create Variable EmpDetails[ ] as STRING
Create Variable RowIndex as NUM
Set Variable RowIndex = 1
Set Variable Status = GetDataTableRow (SalaryTable, $RowIndex$)
```

GetDate

Use this function to store the current date, in calendar format, in a symbol. You can use Julian date functions to convert the date to a serial number that can be used in calculations. You may find this function useful for keeping track of System Events.

Syntax:

```
GETDATE(["Format"])
```

where:

- Format is the formula for another method of returning a date, such as “dd/mm/yyyy”. This enables the user to have date information returned in the format appropriate to their region. If this parameter is not specified, the default “yyyy-mm-dd” format is followed.

Syntax example:

```
CurrDate=GETDATE()

CurrDate=GETDATE("yy/mm/dd")
```

GetDocumentSize

Use this function to return the document size in memory. The document must be loaded into memory, and saved if contents have been modified, when this function is called. The size of a text file in memory is not necessarily the same size as on a file system. If you require the size of a document on the file system, use the function GetFileSize. The GetDocumentSize function is useful when creating Solution-level REST APIs that may be called by third-party applications. In this case, it can be used to set the content-length header for the REST API. For more information see "Providing REST APIs" in the Longview Integration Guide.

Syntax:

```
GETDOCMENTSIZE("DocumentName")
```

where:

- DocumentName is the name of the text document enclosed in quotes.

Syntax example:

```
Set Variable size = GetDocumentSize("$LVS_APIOutputFile$")
```

GetFileProperties

Use this function to return the properties of a file. It will return the last modified date and time in ISO format, file size and the hash value in an object variable. This function works with absolute and relative paths.

The allowed paths for the GETFILEPROPERTIES function are specified by the PERMITTED_FILEMAINTENANCE_PATH parameter in the lv_af.cfg file, located in the working directory of the lv_af.exe. Files that cannot have properties retrieved, are defined by the EXCLUDED_FILEMAINTENANCE_FILES parameter in the same file. These parameters only apply when they are set.

Syntax:

```
GETFILEPROPERTIES("FileName")
```

where:

- FileName is the name of file enclosed in quotes that the properties will be returned for.

Syntax example:

```
CREATE VARIABLE oFileProperties AS Object
SET VARIABLE oFileProperties = GETFILEPROPERTIES("Execute.lvpro")
```

See also

- Copy File
- Delete File
- FileExists
- Move File
- Rename File

GetFileSize

Use this function to return the file size on the local accessible file system. This function must be called after the SAVE command.

Syntax:

```
GETFILESIZE("FileName")
```

where:

- FileName is the name of binary file enclosed in quotes.

Syntax example:

```
Set Variable size = GetFileSize("abc.txt")
```

GetLockComment

Use this function to retrieve the comment associated with a particular lock.

A lock is a Longview database security feature preventing access to data currently used by another user.

Syntax:

```
GETLOCKCOMMENT("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file.

Syntax example:

```
CREATE VARIABLE TXT as string

SET VARIABLE TXT = GETLOCKCOMMENT("Tax_Data.lvdsp")
```

GetLockUser

Use this function to retrieve the name of the user who created a particular lock.

A lock is a Longview database security feature preventing access to data currently used by another individual.

Syntax:

```
GETLOCKUSER("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file.

Syntax example:

```
CREATE VARIABLE TXT as string

SET VARIABLE TXT = GETLOCKUSER("Tax_Data.lvdsp")
```

GetMaintenanceUser

Use this command to return the ID of the user who currently has maintenance on for the system.

Syntax:

```
GETMAINTENANCEUSER()
```

You can use this command with the Set Variable command to retrieve the user ID and description to variables.

Example:

```
Set VARIABLE userIDmtce = GETMAINTENANCEUSER()

Set VARIABLE userDesc = [[USER, DESCRIPTION, $userIDmtce$]]
```

GetNextPeriod

Use this function to return the next symbol in the hierarchy which is a sibling of the specified symbol (if symbol is the last symbol, a blank is returned) in the time periods dimension.

Syntax:

```
GETNEXTPERIOD(SymName)
```

where:

- **SymName** is the name of a symbol in the time periods dimension. The symbol must be a descendent of a symbol specified in either the SGPTimePerActivity or SGPTimePerYTD attributes. For more information on attributes, see the Longview Application Administrator Guide.

  Note: GetNextPeriod returns an error message if the user does not have access to that particular symbol.

Syntax example:

```
CREATE VARIABLE X AS STRING

SET VARIABLE X = GetNextPeriod(A0503)
```

GetOpenPeriod

Use this function to return the open period symbol for the year. The function determines the root symbol for the symbol designated by either the SGPTimePerActivity or SGPTimePerYTD attributes, and then returns the symbol whose attribute ZFXTimePerOpenList contains that root symbol. If an open period cannot be found, the function returns a blank.

For more information on attributes, see the Longview Application Administrator Guide.

Syntax:

```
GETOPENPERIOD(SymName)
```

where:

- SymName is the name of a symbol in the time periods dimension. The symbol must be a descendent of a symbol specified in either the SGPTimePerActivity or SGPTimePerYTD attributes.

  Note: GetOpenPeriod returns an error message if the user does not have access to that particular symbol.

Syntax example:

```
CREATE VARIABLE X AS STRING

SET VARIABLE X = GetOpenPeriod(A0503)
```

GetPreviousPeriod

Use this function to return the previous symbol in the hierarchy which is a sibling of the specified symbol (if SymName is the first symbol, then a blank is returned).

Syntax:

```
GETPREVIOUSPERIOD(SymName)
```

where:

- SymName is the name of a symbol in the time periods dimension. The symbol must be a descendent of a symbol specified in either the SGPTimePerActivity or SGPTimePerYTD attributes. For more information on attributes, see the Longview Application Administrator Guide.

> **Note:** GetPreviousPeriod returns an error message if the user does not have access to that particular symbol.

Syntax example:

```
CREATE VARIABLE X AS STRING

SET VARIABLE X = GetPreviousPeriod(A0503)
```

GetResourceString

Use this function to retrieve resource strings from a resource bundle residing on the Longview Data Server. The resource bundle must be loaded into memory before using this token.

You may find this function useful when creating Longview Apps that need to be supported for multiple languages.

The language and culture of the operating system (OS), along with the name of the resource bundle determines how the resource key lookup is performed. The names of resource bundles can be one of the following formats:

Order of lookup	Name of resource	Example
1	resourceBundle.Language-Culture.extension	Strings.en-CA.lvres
2	resourceBundle.Language.extension	Strings.en.lvres
3	resourceBundle.extension	Strings.lvres

For example, for an OS that is en-CA (English language, Canada culture), the look up will occur in the following files in the following order:

- Strings.en-CA.lvres
- Strings.en.lvres
- Strings.lvres

By organizing resources properly, you can develop Longview Apps that will support multiple languages.

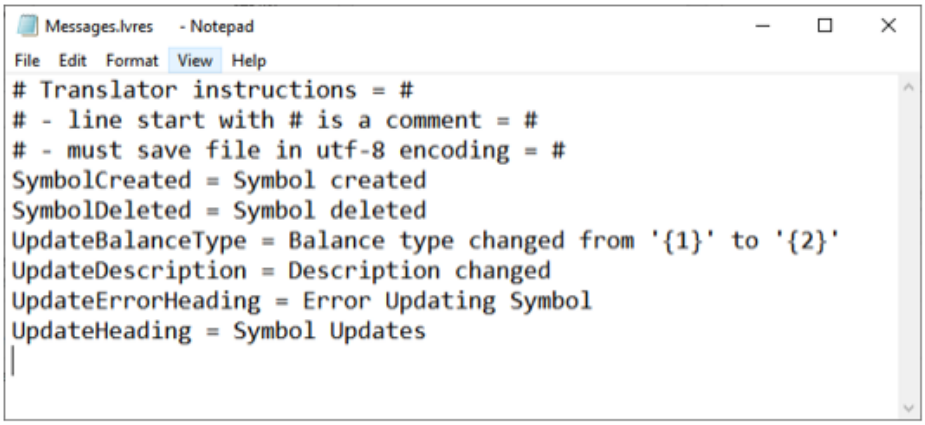
Syntax:

```
GetResourceString("resourceBundle", "resourceKey")
```

where:

- resourceBundle is the name of the resource bundle.
- resourceKey is the name of the resource key.

Sample format of resource file



Syntax example:

```
CREATE VARIABLE sMsg AS STRING
SET VARIABLE sMsg = GetResourceString("Messages", "Greeting")
SHOW MESSAGE "$sMsg$"
```

GetServerWarning

Use this function to retrieve the server error message when the servers have encountered errors due to a failed restatement or other server related errors.

If errors exist, a multi-line message will be returned:

"Server errors have occurred that require review:

System administrators: Check error details in Longview Server Manager under Data Server Logs.

All other users: Contact your system administrator for assistance."

If no errors exist, then nothing will be returned.

Note: For more information on Server warnings, see the Longview Server Manager Guide.

Syntax:

```
GetServerWarning()
```

Syntax example:

```
CREATE VARIABLE X AS STRING
SET VARIABLE X = GetServerWarning()
```

GetStatus

You can use this function to obtain the status of various server features, such as eliminations, journal entries, schedules, and so on.

These features are specified as either TRUE (activated) or FALSE (deactivated) in the Server configuration file (lvsrvr.cfg). When the server is started, it consults this file and switches these features on or off accordingly.

This function is useful when you want to perform an action in Longview but do not know whether you have access to that particular functionality.

When you connect to the Longview database, the status of these features is automatically stored. When the GetStatus function is called with a feature name, the status of the feature is returned to a variable. The status returned can be one of two possible values:

Value	Description
0	FALSE. The feature is deactivated on the server.
1	TRUE. The feature is activated on the server.

When you disconnect from the Longview database, the status of the features is cleared, until the next time you connect to the database.

Syntax:

```
GETSTATUS ("ServerOptionName")
```

where:

- ServerOptionName can be one of the following, enclosed in double quotation marks:

ServerOptionName value	Corresponding configuration parameter name in the lvsrvr.cfg
JE	USE_JOURNAL_ENTRIES
ELIM	DO_ELIMINATIONS
MODELRULES	USE_MODEL
QUERYRULES	USE_QUERY_RULES
ROLLUPRULES	USE_ROLLUP_RULES
NDD	ACCOUNTING_NDD
PAC	ACCOUNTING_PAC
REC	ACCOUNTING_REC
YTD	ACCOUNTING_YTD
TRN	ACCOUNTING_TRN

Syntax example:

```
GETSTATUS ("MODELRULES")
```

GetSymbolBalanceType

Use this function to return the balance type of the specified symbol. The balance type is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
CREDIT	The symbol has a credit balance type.
DEBIT	The symbol has a debit balance type.
NEITHER	The symbol has neither a credit or debit balance type.

Note: Credit and Debit is only used for the Accounts dimension and indicate whether the account is a credit account or a debit account. All other dimensions use Neither.

Syntax

```
GetSymbolBalanceType("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose balance type is to be retrieved.

Syntax example:

```
Set Variable symBalanceType = GetSymbolBalanceType("ACCOUNTS", "Trial_
Balance")
```

GetSymbolCreatedTimeStamp

Use this function to return the created time stamp of the specified symbol. The description is returned as a string in this format "YYYY-MM-DD hh:mm:ss".

Syntax:

```
GetSymbolCreatedTimeStamp("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose created time stamp is to be retrieved.

Syntax example:

```
Set Variable symCreatedTimeStamp = GetSymbolCreatedTimeStamp("ACCOUNTS",
"Trial_Balance")
```

GetSymbolDescription

Use this function to return the description of the specified symbol. The description is returned as a string.

Syntax:

```
GetSymbolDescription( "dimension", "symName"[, "languageCode"])
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose description is to be retrieved.
- languageCode is the is the optional language code, with a default value of EN.

Syntax example:

```
Set Variable symDescription = GetSymbolDescription( "ACCOUNTS", "A12455",
"EN")
```

GetSymbolModifiedTimeStamp

Use this function to return the last modified time stamp of the specified symbol. The description is returned as a string in this format "YYYY-MM-DD hh:mm:ss".

Syntax:

```
GetSymbolModifiedTimeStamp("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose modified time stamp is to be retrieved.

Syntax example:

```
Set Variable symModifiedTimeStamp = GetSymbolModifiedTimeStamp("ACCOUNTS",
"Trial_Balance")
```

v26.2 GetSymbolParent

Use this function to return the parent of the symbol specified from the hierarchy.

Syntax:

```
GetSymbolParent("dimension", "symName", "HierachySymName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose parent is to be retrieved.
- HierachySymName is the name of the hierarchy from which the parent symbol is retrieved.

Syntax example:

```
Create variable sSymParent as String

Set Variable sSymParent = GetSymbolParent( "ACCOUNTS", "NIBTAdjT",
"NIBTAdj")
```

v26.2 GetSymbolPartitionIndex

Use this function to return the partition index for a specified symbol. The partition index is returned as a numeric value.

Syntax:

```
GetSymbolPartitionIndex ("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol for which the partition index is retrieved.

Syntax example:

```
Create Variable nSymPartitionIndex as NUM

Set Variable nSymPartitionIndex = GetSymbolPartitionIndex( "ACCOUNTS",
"A12455")
```

GetSymbolPriority

Use this function to return the priority of the specified symbol to a specific parent. The priority of a symbol represents the position of the symbol with respect to its parent. The priority is returned as a numeric value between 0 and 99999999 (0 representing the lowest priority).

Syntax:

```
GetSymbolPriority("dimension", "symName", "parentSymName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose priority is to be retrieved.
- parentSymName is a parent symbol of the specified symbol.

Syntax example:

```
Set Variable symPriority = GetSymbolPriority("ACCOUNTS", "Cash", "Trial_
Balance")
```

GetSymbolRollupStatus

Use this function to return the rollup status of the specified symbol. The rollup status is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
ON	The parent data will be calculated dynamically (automatically).
OFF	The parent data will not be calculated dynamically (automatically), but only during restatements.

Syntax:

```
GetSymbolRollupStatus( "dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose rollup status is to be retrieved.

Syntax example:

```
Set Variable = GetSymbolRollupStatus( "ACCOUNTS", "Trial_Balance")
```

GetSymbolSortOption

Use this function to return the sort option of the specified symbol. The sort option is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
ASCENDING	The symbol sorts the child symbols in ascending order.
DESCENDING	The symbol sorts the child symbols in descending order.
MANUAL	The symbol sorts the child symbols manually as specified by the symbol priorities.

Syntax:

```
GetSymbolSortOption("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose sort option is to be retrieved.

Syntax example:

```
Set Variable symSortOption = GetSymbolSortOption("ACCOUNTS", "Trial_
Balance")
```

GetSymbolType

Use this function to return the symbol type of the specified symbol. The symbol type is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
STANDARD	The symbol is a standard symbol.
CARRYFORWARD	The symbol is a carry forward symbol.
STATIC	The symbol is a static symbol.

Syntax:

```
GetSymbolType("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose type is to be retrieved.

Syntax example:

```
Set Variable symType = GetSymbolType("ACCOUNTS", "Trial_Balance")
```

GetSymbolVirtualStatus

Use this function to return the virtual status of the specified symbol. The virtual status is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
ON	The symbol is a virtual symbol.
OFF	The symbol is not a virtual symbol.

Syntax:

```
GetSymbolVirtualStatus("dimension", "symName")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose virtual status is to be retrieved.

Syntax example:

```
Set Variable symVirtualStatus = GetSymbolVirtualStatus("ACCOUNTS", "Trial_
Balance")
```

GetSymbolWeight

Use this function to return the weight of the specified symbol to a specific parent. The weight is returned as a string. Review the following table for a summary of possible return values and their corresponding status.

Return Value	Description
+	Symbol adds to the parent value.
-	Symbol subtracts from the parent value.
0	Symbol has no mathematical effect on the specified parent.

Syntax:

```
GetSymbolWeight( "dimension", "symName", "parentSymame")
```

where:

- dimension is the name of the dimension that contains the symbol.
- symName is the name of the symbol whose weight is to be retrieved.
- parentSymName is a parent symbol of the specified symbol.

Syntax example:

```
Set Variable symWeight = GetSymbolWeight("ACCOUNTS", "Cash", "Total_Assets")
```

GetTime

Use this function to get the current time.

You may find this function useful for keeping track of system events. By default, the time appears in the format hh:mm:ss, if no other format is specified.

Syntax:

```
GETTIME (["Format"])
```

where:

- Format is the formula for another method of returning a times tamp, such as “mm:ss:hh”. If this parameter is not specified, the default “hh:mm:ss” format is followed.

Syntax example:

```
GETTIME ()
GETTIME ("mm:hh:ss")
```

GetType

Use this function to identify the variable type of a specified variable. The variable type is returned as a string.

Possible return values include the following:

- NUM
- STRING
- RANGE
- NUMLIST
- STRINGLIST
- OBJECT
- OBJECTLIST

For more information on variables, see Create Variable.

Syntax:

```
GETTYPE ("VariableName")
```

where:

- VariableName is the name of the variable whose type you would like to identify.

Syntax example:

```
Create Variable nCount as NUM

Create Variable sType as STRING
Set Variable sType = GETTYPE("nCount")
Show Message "The variable nCount is of type $sType$"
```

Syntax example:

```
Create Variable oEmployee as Object
Create Property oEmployee.ID as Num
Set Variable propType = GETTYPE (oEmployee)
Set Variable propType = GETTYPE (oEmployee.ID)
Show Message "The property oEmployee.ID is of type $sType$"
```

GetWorkflowInfo

Use this function to retrieve information for a specified hierarchical Workflow process and step. The function will return an object with the following properties:

Property	Description
DimensionName	String indicating the name of the Approval Dimension
SymbolName	String indicating the symbol name of the hierarchical spec in the approval dimension for a specified stepname
Level	Number indicating the level of the hierarchical spec in the approval dimension for a specified stepname

Note: The GetWorkflowInfo function can be used against simple steps but the results for the DimensionName, SymbolName and Level will be blank, as they do not apply.

Syntax:

```
GETWORKFLOWINFO(Process, Step[,Symbol])
```

where:

- Process is the name of an active Workflow process. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Symbol is optional for hierarchical steps, and is the symbol for the hierarchical step. Symbol is not used for simple steps.

Syntax example:

```
                Create Variable oWFInfo as Object

Set Variable oWFInfo = GETWORKFLOWINFO(MonthEndProcess, "ABC Canada",
E10000)
```

GetWorkflowState

Use this function to retrieve the current Workflow state for a specified Workflow process and step. The function will return an object with the following properties:

Property	Description
UserID	String indicating the username of the user who last changed the state
UserDescription	String indicating the user description of the user who last changed the state
Modified	String indicating the datetime stamp of the last state change
Comment	String indicating the comment for the last state change, if applicable
CurrentStatus	Number indicating the current status of the workflow step,
PreviousStatus	Number indicating the previous status of the workflow step,

The return values for the status properties CurrentStatus and PreviousStatus:

Property	Description
-1	Failed **Note:** Failed indicates the Longview Application Framework function did not execute properly or encountered an issue.
0	Not Started
999	Rejected
1000	In Progress
2000	Submitted for Approval
3000	Approved
9999	No Previous Status Available

Syntax:

```
GETWORKFLOWSTATE(Process, Step[,Symbol])
```

where:

- Process is the name of an active Workflow process. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Symbol is optional for hierarchical steps, and is the symbol for the hierarchical step. Symbol is not used for simple steps.

Syntax example:

```
                CREATE VARIABLE oWFState as OBJECT

Set Variable oWFState = GETWORKFLOWSTATE(MonthEndProcess, "ABC Canada",
E10000)
```

GetWorkflowStatus

Use this function to retrieve the status of a specific step within a Longview Workflow process. Review the following table for a summary of possible return values and their corresponding status.

Return value	Longview Workflow Status
-1	(Failed) **Note:** Failed indicates the Longview Application Framework function did not execute properly or encountered an issue.
0	Not Started
999	Rejected
1000	In Progress
2000	Submitted for Approval
3000	Approved

Syntax:

```
GETWORKFLOWSTATUS(Process, Step[, Symbol])
```

where:

- Process is the name of the Longview Workflow process for which to return the status. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Symbol is optional unless the specified step is hierarchical, and is the symbol for the step.

Syntax example:

```
Create Variable nWorkflowStatus as NUM
Set Variable nWorkflowStatus = GETWORKFLOWSTATUS(MonthEndProcess, "ABC
Canada")
```

Syntax example:

```
Create Variable nWorkflowStatus as NUM
Set Variable nWorkflowStatus = GETWORKFLOWSTATUS(MonthEndProcess, "ABC
Canada", E11111)
```

GetWorkflowUserRole

Use this function to retrieve a list of user roles for a specified Workflow process and step. The function will return a list indicating whether the currently connected user is an OWNER or APPROVER of the specified step. If the user is both an OWNER and APPROVER, then both roles are returned in the list.

Syntax:

```
GETWORKFLOWUSERROLE(Process, Step[,Symbol])
```

where:

- Process is the name of an active Workflow process. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Symbol is optional for hierarchical steps, and is the symbol for the hierarchical step. Symbol is not used for simple steps.

Syntax example:

```
                Create Variable sWFRole[] as STRING

Set Variable sWFRole = GETWORKFLOWUSERROLE(MonthEndProcess, "ABC Canada",
E10000)
```

GroupExists

Use this function to determine whether the specified group exists in the database. A return value of 1 means the group exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
GROUPEXISTS("Group")
```

where:

- Group is the name of the group to search for.

Syntax example:

```
GroupExists("Administrators")
```

If

Use this function to carry out a set of instructions if a condition is TRUE.

Do not confuse the If function with the When function:

Function	Description
If	To carry out a set of instructions if any value of a symbol matches a condition.
When	To carry out a set of instructions each time the contents of a cell match a condition.

Syntax:

```
IF Condition
Instructions
[ELSE
Instructions2]
END IF
```

where:

- Condition is a test to determine if a relationship is true or false. Your system applies the If condition to all data cells, and is considered TRUE if the condition is TRUE for any single value. (You can also compare a constant against another constant.) If Condition is TRUE, your system carries out Instructions.
- Instructions is a set of instructions that your system carries out if Condition is TRUE.

Syntax example:

```
IF VALUE(WDGTMFG, A20000, AYR01) LT 0.2 * VALUE(WDGTMFG, A99999, AYR01)

Div = 0.1 * A99999
END IF
```

IsAreaBatchStatus

Use this function to retrieve whether the specified status for batches that impact data in a specified DataArea is TRUE or FALSE. If the specified status for batches that impact data in a specified DataArea is TRUE, this function returns a value of 1. If the specified status for batches that impact data in a specified DataArea is FALSE, this function returns a value of 0.

Syntax:

```
ISAREABATCHSTATUS("dataspec.lvdsp",status)
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.
- status is either error or inprogress.

IsAreaEventStatus

Use this function to retrieve whether the specified status for events that were triggered by data in a specified DataArea is TRUE or FALSE. If the specified status for events that were triggered by data in a specified DataArea is TRUE, this function returns a value of 1. If the specified status for events that were triggered by data in a specified DataArea is FALSE, this function returns a value of 0.

Syntax:

```
ISAREAEVENTSTATUS("dataspec.lvdsp",status)
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.
- status is either error or inprogress.

IsBatchMode

Use this function to determine whether Longview Application Framework is running in batch mode or not. This information helps developers design their apps to work in both batch mode and Smart Client mode, since some commands cannot be run in batch mode (such as Show Message). A return value of 1 indicates that the function is running in batch mode, a return value of 0 indicates that it is not.

Syntax:

```
IsBatchMode()
```

IsChanged

Use this function to determine whether the DataArea has been modified since the initial download. This function can be useful if you want to prompt users to save their changes before closing a Data Grid.

A return value of 0 indicates that the DataArea is unchanged, while a value of 1 indicates a modification.

Syntax:

```
ISCHANGED(DataArea)
```

where:

- DataArea is the name of the DataArea.

IsConnected

Use this function to determine if there is a connection to the Longview database. A value of 1 represents a connection; a value of 0 represents no connection.

Syntax:

```
ISCONNECTED()
```

IsExclusiveOn

Use this function to determine if exclusive mode is on in the system.

Syntax

```
ISEXCLUSIVEON()
```

If exclusive mode is on, this function returns a 1. If exclusive mode is off, it returns a 0.

Syntax example

```
Set VARIABLE Flag= ISEXCLUSIVEON()
```

See also

- Set Variable
- IsMaintenanceOn
- GetMaintenanceUser

IsHierarchicalStep

Use this function to determine if a step in a Longview Workflow process is hierarchical. Review the following table for possible return values and their meanings.

Return value	Meaning
1	The step is hierarchical
0	The step is simple
-1	There was a failure in returning the step

Syntax:

```
ISHIERARCHICALSTEP(Process, Step)
```

where:

- Process is the name of the Longview Workflow process containing the step. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the step for which you want to determine the hierarchical status. If the step name contains spaces, enclose it in double quotation marks (" ").

IsKeyword

Use this function to determine if the specified word is already defined as a command, function, or document type. Enclose Text in double quotation marks. A value of 1 means the keyword is already in use in the system; a value of 0 means it is not.

Syntax:

```
ISKEYWORD("Text")
```

where:

- Text is a string representing the keyword, enclosed in double quotation marks.

IsLocked

Use this function to determine if there is a lock in the database covering all or part of the area contained in the specified DataSpec. A value of 1 means such a lock exists; a value of 0 means no such lock exists.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
ISLOCKED ("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.

IsMaintenanceOn

Use this function to determine if maintenance is on in the system.

Syntax:

```
ISMAINTENANCEON()
```

where:

- If maintenance is on, this function returns a 1. If maintenance is off, it returns a 0.

Syntax example:

```
Set VARIABLE Flag=ISMAINTENANCEON()
```

See also

• Set Variable

• IsExclusiveOn

• GetMaintenanceUser

IsMaintenanceUser

Use this function to determine whether the current user has authorization to turn maintenance on.

Syntax:

```
ISMAINTENANCEUSER()
```

IsName

Use this function to determine if the text string is a valid name of the specified type. The supported types are SYMBOL, USER, and OTHERS, though the specification of these types is optional. A value of 1 is returned if the name is valid for the specified type; a value of 0 means the name is not valid.

Syntax:

```
ISNAME("Text"[,"USER"|"SYMBOL"|"OTHERS"])
```

where:

- Text is the name whose validity is to be tested, enclosed in double quotation marks.
- SYMBOL sets the specified type to symbol.
- USER sets the specified type to user.
- OTHERS checks non-user and non-symbol types.

IsParentSymbol

Use this function to determine whether the specified database symbol is a parent. A value of 1 means the symbol is a parent symbol; a value of 0 means that it is not (therefore, the symbol is a leaf). If the user does not have access to that particular symbol, the function will return an error message.

Syntax:

```
ISPARENTSYMBOL("dimName", "symName")
```

where:

- dimName is the dimension in which the symbol exists, enclosed in double quotation marks.
- symName is the name of the symbol whose status is to be determined, enclosed in double quotation marks.

IsReadOnly (DataArea)

Use this function to determine whether any part of a DataArea is read only for the current user.

Returns 1 if any part of the area specified is read only for the current user; 0 if the symbol is not read-only for the current user.

Syntax:

```
ISREADONLY ("SymName0[Structure]","SymName1[Structure]"..."SymNameN
[Structure]")
```

where:

- SymName is the name of the symbol whose status is to be determined, enclosed in double quotation marks (including Structure, if specified).
- Structure is optional and is a range of symbols. To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

IsReadOnly (Symbol)

Use this function to determine whether a symbol is read-only for the current user.

Returns 1 if SymName is read-only for the current user; 0 if the symbol is not read-only for the current user.

Syntax:

```
ISREADONLY("dimName", "symName")
```

where:

- dimName is the dimension in which the symbol exists.
- symName is the name of the symbol whose status is to be determined.

IsRowLimitExceeded

Use this function to determine if the number of rows downloaded exceeds the RowLimit specified in the associated Data Table Definition. A value of 1 indicates that the row limit has been exceeded. A value of 0 indicates that the row limit has not been exceeded.

Note: This function must be called immediately after the DOWNLOAD DataTable command to ensure correct results.

Syntax:

```
IsRowLimitExceeded("DataTableName")
```

where:

- DataTableName is the name of the DataTable object that was downloaded.

Syntax example:

```
Create DataTable myTable Using "AllProducts.lvdtd"
Download myTable
Set Variable nFlag = IsRowLimitExceeded ("myTable")
```

JDate

Use this function to find the calendar date equivalent of a date in Julian date format.

Julian date format consists of a five digit number; calendar dates appear in the format yyyy-MM-dd hh:mm:ss. (Note that uppercase “M” is used to designate months, and lowercase “m” is used to designate minutes).

Note: The valid range for the JDate function is between 25567 and 402133 (which correspond to 1970 - 3000). Entering an invalid date will cause the following error message to appear: "Invalid JDATE range. JDATE must be between 25567 and 402133."

Do not confuse the JDate function with the JValue function:

Function	Description
JDate	To find the calendar date equivalent of a Julian date.
JValue	To find the Julian date equivalent of a calendar date.

Syntax:

```
JDATE (JulianDate[,"Format"])
```

where:

- JulianDate is the five-digit number (between 25567 and 50421) representing a date in the Julian calendar to be returned as a calendar date.
- Format is the format of the calendar date to be returned. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JDATE (28226, "dd/MM/yyyy")
```

JDiff

Use this function in an equation to calculate the number of days between two specified calendar dates.

Syntax:

```
JDIFF (SymName|"Date1", SymName|"Date2"[,"Format"])
```

where:

- SymName is a TIMEPER symbol representing either a start or end date for the range.
- Date1 is the start date of the range.
- Date2 is the end date of the range.
- Format is the format of the dates to be compared. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JDIFF ("2007-05-06", "2007-12-31")
```

JERollover

If your application uses journal entries, use this Longview Application Framework function as part of a period-end or year-end rollover.

At the end of a period or year, you can use the application to transfer values to the new time period.

- The period-end rollover occurs at the end of each period.
- The year-end rollover occurs at the end of each fiscal year.

To prepare for a rollover, you need to use the JERollover function. For a period-end rollover, JERollover:

- changes the current period to the next period (uses the SGPCurrentPeriod system attribute)
- duplicates all recurring journal entries for the current period and marks them as Validated

For a year-end rollover, JERollover:

- changes the current period to the first period of the next year (uses the SGPCurrentPeriod system attribute)
- changes the current year to the next year (uses the SGPCurrentYear system attribute)
- duplicates all recurring journal entries for the current period and marks them as Validated

For more information on attributes, see the Longview Application Administrator Guide.

Before using this function, you must ensure that Longview Journal Entries is not open on the desktop and that all journal entries have been posted.

The period-end and year-end rollover are exclusive from each other. If the current period-end is also a year-end, you cannot perform a period-end rollover.

Syntax:

```
SET VARIABLE VarName = JEROLLOVER ("RollPeriod", "ExecuteMode")
```

where:

- RollPeriod can be one of the following:

Value	Description
PERIODEND	To specify a period-end rollover.
YEAREND	To specify a year-end rollover.

- ExecuteMode can be one of the following:

Value	Description
FALSE	This is the default setting and specifies that the rollover runs in test mode. You do not need to turn maintenance on to run a rollover test.
TRUE	This setting specifies that the rollover runs in execute mode. To use this mode, you must turn maintenance on.

The application creates a code in the active worksheet to indicate the results of the rollover:

Value	Description
0	Rollover was successful.
3	Rollover failed because the user is not authorized to perform a journal entry rollover. A user requires authorization to create current period journal entries and modify system attributes to perform a journal entry rollover.
2	Rollover failed because there were outstanding or unposted journal entries for the current period.
1	Period-end rollover failed because the current period requires a year-end rollover.
-1	Rollover failed for some other reason.

Syntax example:

```
CREATE VARIABLE jeRollStatus AS NUM

MAINTENANCE ON
SET VARIABLE jeRollStatus = JERollover("YEAREND", "TRUE")
MAINTENANCE OFF
```

JValue

Use this function in an equation to find the equivalent of a calendar date in Julian date format.

Julian date format consists of a five digit number; calendar dates appear in the format yyyy-MM-dd hh:mm:ss. (Note that uppercase “M” is used to designate months, and lowercase “m” is used to designate minutes).

Note: The valid range for the JDate function is between 25567 and 402133 (which correspond to 1970 - 3000). Entering an invalid date will cause the following error message to appear: "Invalid JDATE range. JDATE must be between 25567 and 402133."

Do not confuse the JValue function with the JDate function:

Function	Description
JValue	To find the Julian date equivalent of a calendar date.
JDate	To find the calendar date equivalent of a Julian date.

Syntax:

```
JVALUE (SymName|"Date"[, "Format"])
```

where:

- SymName is TIMEPER symbol from which the Julian date is to be calculated.
- Date is the calendar date to be returned as a Julian date.
- Format is the format of the calendar date. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JVALUE (31/07/2000, "dd/MM/yyyy")
```

ListAppend

Use this function to append a value or another list to a list.

Syntax:

```
ListAppend(list, value|list2)
```

where:

- list is a variable of type NUM[], STRING[], or RANGE.
- value is one of the following:
  - If list is of type NUM[], value is the number to append to list.
  - If list is of type STRING or RANGE, value is the string to append to list.
- list2 is a variable of the same type as list that contains the value to append.

Syntax example:

```
CREATE VARIABLE myList1[] AS STRING

CREATE VARIABLE myList2[] AS STRING
CREATE VARIABLE aResult[] AS STRING
CREATE VARIABLE bResult[] AS STRING

SET VARIABLE myList1 = "a|b|c|d|e|f"
SET VARIABLE myList2 = "a|c|f"

Set VARIABLE aResult = ListAppend(myList1, "z")
Set VARIABLE bResult = ListAppend(myList1, myList2)

// In the above example, aResult equals a,b,c,d,e,f,z and bResult equals
a,b,c,d,e,f,a,c,f.
```

Syntax example:

```
CREATE VARIABLE stringList[] AS STRING
CREATE VARIABLE entity AS STRING
CREATE VARIABLE stringValue AS STRING
Set VARIABLE stringList = "A| B |C"
Set VARIABLE entity = "E11110"
Set VARIABLE stringValue = "Entity"
```

```
Set VARIABLE stringList = ListAppend(StringList, "$stringValue$")

Set VARIABLE stringList = ListAppend(StringList, "Entity")

Set VARIABLE stringList = ListAppend(StringList, "$Entity$")

Set VARIABLE stringList = ListAppend(StringList, $stringValue$)

Set VARIABLE stringList = ListAppend(StringList, $$stringValue$$)

Set VARIABLE stringList = ListAppend(StringList, stringValue)

//In the above example, stringList equals A, B ,C, Entity, Entity, E11110,
E11110, E11110, Entity
```

ListFindAll

Use this function to search a list for occurrences specified in a second list and return the index of any matches.

Syntax:

```
ListFindAll(list, find)
```

where:

- list is a variable of type NUM[], STRING[], or RANGE.
- find is a variable of the same type as list that contains the values to search for in list.

Syntax example:

```
CREATE VARIABLE myList1[] as STRING

CREATE VARIABLE myList2[] as STRING

CREATE VARIABLE myList3[] as NUM

SET VARIABLE myList1 = "a|b|c|d|e|f"

SET VARIABLE myList2 = "b|a|e|g"
SET VARIABLE myList3 = ListFindAll(myList1,myList2)

//In this example, myList3 equals 1,2,5.
```

ListFindFirst

Use this function to search a list for the first occurrence of a specified value and return the index of the match.

Syntax:

```
ListFindFirst(list, find)
```

where:

- list is a variable of type NUM[], STRING[], or RANGE.
- find is the value to search for or a variable of the same type as list that contains the values to search for in list. If you are searching for a string value, you must enclose it in double quotation marks.

Syntax example:

```
CREATE VARIABLE myList1[] AS STRING
CREATE VARIABLE nResult AS NUM
CREATE VARIABLE bResult AS NUM
CREATE VARIABLE sValue AS STRING
SET VARIABLE myList1 = "a|b|c|d|e|f"
SET VARIABLE sValue = "d"
SET VARIABLE nResult = ListFindFirst(myList1, sValue)
SET VARIABLE bResult = ListFindFirst(myList1, "b")
// In this example, nResult equals 4 and bResult equals 2.
```

ListRemoveDuplicates

Use this function to remove duplicate entries from a list.

Syntax:

```
ListRemoveDuplicates(list)
```

where:

- list is a variable of type NUM[], STRING[], or RANGE that contains the list to remove duplicates from.

Syntax example:

```
CREATE VARIABLE myList1[] AS STRING
CREATE VARIABLE myList2[] AS STRING

SET VARIABLE myList1 = "a|b|c|a|a|d|e|e"
Set VARIABLE myList2 = ListRemoveDuplicates(myList1)
//In this example, myList2 equals a,b,c,d,e.
```

ListSort

Use this function to sort a list in ascending or descending order.

Syntax:

```
ListSort(list, ASCENDING|DESCENDING)
```

where:

- list is a variable of type NUM[], STRING[] or RANGE that contains the list to sort.
- ASCENDING sorts the list in ascending alphabetical or numerical order.
- DESCENDING sorts the list in descending alphabetical or numerical order.

Syntax example:

```
CREATE VARIABLE myList1[] AS STRING

CREATE VARIABLE myList2[] AS STRING


SET VARIABLE myList1 = "jan|feb|mar"
Set VARIABLE myList2 = ListSort(myList1, ASCENDING)

// In this example, myList2 equals feb|jan|mar.
```

Log

Use this function in an equation to calculate the logarithm of a symbol or value.

Syntax:

```
LOG (Num[, Base])
```

where:

- Num is a numeric value.
- Base is a symbol or a constant. The default is the natural logarithm base e (= 2.71828). If Base is a symbol, your system determines the logarithm for each cell in SymName from the corresponding cell in Base.

Syntax example:

```
LogValue = LOG (ExpValue,10)
```

Max

Use this function to calculate the largest value in a series.

Syntax:

```
Max (Value[, Value])
```

where:

- Value is a number.

Syntax example:

```
MAX (A2100#99)
```

Min

Use this function to calculate the smallest value in a series.

Syntax:

```
Min (Value[, Value])
```

where:

- Value is a number.

Syntax example:

```
MIN (A2100#99)
```

Neg

Use this function in an equation to find symbols with negative values. If the symbol has a positive value, the Neg function returns zero.

Syntax:

```
NEG (SymName|Number)
```

where:

- SymName is a symbol whose negative values you want to specify.
- Number is a value.

Syntax example:

```
Account1 = NEG (Account2)
```

NumToStr

Use this function to convert a numeric value to a character string.

You may find this function useful for comparison purposes. For example, suppose a complete account number in your system appears in the format A11234. You can use the NumToStr function to change any numeric values to character strings, so that they can be compared correctly.

The value provided by your system depends on the format of the source item being converted:

Format of Source	Value of TargetSymName	Example
Character string	Returns the source string	
Numeric and an integer	Character string representing the number	If Source is 1234, TargetSymName becomes the character string "1234".
Numeric but not an integer	Number	No specific number of decimals. No zero-padding.

Syntax:

```
NUMTOSTR (SymName|Num)
```

where:

- SymName is a symbol name.
- Num is a value.

Syntax example:

```
NumToStr (1234)
```

Operators (arithmetic)

Use arithmetic operators to designate the operational relationships between expressions in an equation. For example, to add two numbers, use the "+" operator (as in "5+5").

Unlike most other functions, arithmetic operators do not use an actual function word within syntax — they use arithmetic symbols instead. If an equation contains more than one arithmetic operator, your system evaluates the expressions in the following order of precedence:

1. functions and expressions in parentheses
2. multiplication and division (* , /)

   Caution: The use of the asterisk as a multiplication symbol in the application requires that it be enclosed in quotation marks ("*"). An asterisk without quotation marks is interpreted by the system as a wildcard search character, and operations including it will not give the expected results.

3. addition and subtraction (+ , -)

If the arithmetic operations have equal precedence, your system calculates them from left to right.

Syntax:

```
Expression1 OPERATOR Expression2
```

where:

- Each Expression is a mathematical expression consisting of symbols, constants, values, and functions. You can use parentheses to nest one expression within another.
- OPERATOR is an arithmetic operator. Although extra spacing is not required, you should make the expression easier to read and edit by typing a space before and after each operator:

Value	Description
+	Addition
-	Subtraction
/	Division
"*"	Multiplication

Syntax example:

```
Div = 0.1 "*" NetInc
```

Error messages caused by arithmetic calculations

If you use an invalid arithmetic operation in a statement, one of several error messages may appear:

Error	Description
#OVERFLOW	Result of a value greater than the maximum value the CPU can handle.
#UNDERFLOW	Result of a value less than the minimum value the CPU can handle.
#INFINITE	Result of an out-of-value range number, underflow, or overflow divided by 0.
-0	Result of 0 divided by 0.

These messages may appear as a result of unusual data or erroneous logic.

For possible solutions to logic errors (for example, to limit calculations when the divisor would be equal to zero), see IRR and When...

Operators (conditional)

Use conditional operator functions to link two conditional expressions.

To instruct Longview to carry out certain instructions, use a conditional operator:

Conditional operator	Description
AND	When each of several conditions must be true.
OR	When one of several conditions must be true.

You can use parentheses to change or change the order in which conditions are evaluated; for example:

```
IF (Cash1 GT 0 OR Cash2 LT 100) AND (Cash3 LT 0 OR Cash4 NE 100)
```

Syntax:

```
Condition1 Operator Condition2
```

where:

- Each Condition is a relational expression containing the EQ, GE, GT, NE, LT, or LE functions, or another expression containing the And or Or functions. For information on relational operators, see Operators (relational).
- Operator is a conditional operator. Select one of the following:

Value	Description
AND	To find an expression that matches both conditions.
OR	To find an expression that matches either condition.

Syntax example:

```
IF NetInc GT 0 AND Debt LT 0.2 "*" NetInc
```

Operators (relational)

Use relational operators to compare two mathematical expressions.

To display data under certain conditions in a Model or Procedure Document, use relation operator functions to specify a condition that must be true. You can use parentheses to change the order in which expressions are compared.

Syntax:

```
Expression1 OPERATOR Expression2
```

where:

- Each Expression is a mathematical combination of symbols, constants, and functions. If you use the EQ or NE functions, Expression can be a character string enclosed in double quotation marks.
- OPERATOR is a relational operator. Select one of the following:

Value	Description
EQ or ==	Equal to.
GE or >=	Greater than or equal to.
GT or >	Greater than.
LE or <=	Less than or equal to.
LT or <	Less than.
NE or !=	Not equal to.

Syntax example:

```
IF NetInc GT 0 AND Debt LT 0.2 "*" NetInc
```

Pos

Use this function in an equation to find symbols with positive values. If the symbol contains a negative value, the Pos function assigns a value of zero.

Syntax:

```
POS (SymName|Number)
```

where:

- SymName is a symbol whose positive values you want to specify.
- Number is a value.

Syntax example:

```
Account1 = POS (Account2)
```

PropertyExists

Use this function to determine whether the specified property exists on the specified object. A return value of 1 means the property exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
PROPERTYEXISTS (ObjectVarName.Property)
```

where:

- ObjectVarName is the name of the object type variable to search for.
- Property is the name of the property to verify the existence of.

Syntax example:

```
Create Variable nExists as NUM

Set Variable nExists = PROPERTYEXISTS("oEmployee.hireDate")
```

Round

Use this function in an equation to round values to the nearest decimal place.

Syntax:

```
ROUND (Value[,NumPlaces])
```

where:

- Value is a number, or a valid mathematical expression.
- NumPlaces is optional and is a positive integer, or a symbol whose values are all positive integers, representing the number of decimal places to which each value must be rounded. The default is to round to the nearest whole number.

RuleExists

Use this function to determine whether the specified rule ID exists in the database. A return value of 1 means the rule ID exists; a value of 0 means it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
RULEEXISTS (RuleID)
```

where:

- RuleID is the rule ID to search for.

Syntax example:

```
RuleExists(100)
```

ScheduleExists

Use this function to determine whether the specified schedule exists in the database. A return value of 1 means the schedule exists; a value of 0 means it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
SCHEDULEEXISTS("SchedName")
```

where:

- SchedName is the name of the schedule to search for.

Syntax example:

```
SCHEDULEEXISTS("ICStandard")
```

SetDataTableCell

Use this function to set a specific cell in a DataTable object. The intersection of the row index and column index specify the cell to set.

Syntax:

```
SetDataTableCell (DataTableName, RowIndex, ColumnIndex, "Value")
```

where:

- DataTableName is the name of the DataTable object in which to set the cell.
- RowIndex is the row index of the cell to update.
- ColumnIndex is the column index of the cell to update.
- Value is the value to set for the cell, enclosed in double quotation marks ("").

Syntax example:

```
Create Variable status as NUM
Create Variable NewStartDate as STRING
Set Variable NewStartDate = "2016-01-25"
Set Variable status = SetDataTableCell (SalaryTable, 1, 1, "$NewStartDate$")
```

SetWorkflowStatus

Use this function to set the status of a specific step within a Longview Workflow process. A return value of 0 indicates the step was set successfully and a return value of -1 indicates the step failed to set.

Note: You must be an Owner or Approver to set the status of a step. For more information, see the Longview Workflow Designer Guide.

Syntax:

```
SetWorkflowStatus(Process, Step, Status, "Comment"[, Symbol])
```

where:

- Process is the name of the Longview Workflow process for which to return the status. If the process name contains spaces, enclose it in double quotation marks (" ").
- Step is the name of the Longview Workflow step within the process. If the step name contains spaces, enclose it in double quotation marks (" ").
- Status is a numeric value, is the status to which you want to set the step, and can be one of the following:

To set status to...	Use...
Rejected	999
In Progress	1000
Submitted for Approval	2000
Approved	3000

Note: Not Started is not a valid status option when using Longview Application Framework. Use Longview Workflow Designer to set the status of a step to Not Started. For more information, see the Longview Workflow Designer Guide.

- Comment is the comment to apply to the workflow log, enclosed in double quotation marks. Do not use double quotation marks (") or the line break character (\n) within the comment itself. Comments cannot be longer than 4000 characters.
- Symbol applies only to hierarchical steps and is optional. Review the following table for when to use the symbol parameter.

Note: Do not use the symbol parameter for simple steps.

To...	Use...
set the status for a dependent step only	the symbol parameter to specify the symbol for the dependent step for which you want to set the status.

To...	Use...
set the status for the hierarchical step, including dependent steps	do not use the symbol parameter.

Syntax example:

```
SetWorkflowStatus(MonthEndProcess, "ABC Canada", 1000, "Validating month end
data", E11111)
```

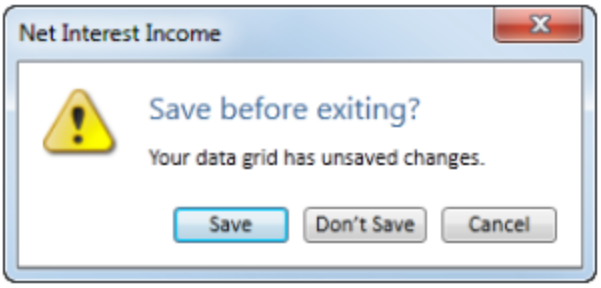
Syntax example:

```
SetWorkflowStatus(MonthEndProcess, "ABC Canada", 1000, "")
```

ShowAdvancedPrompt

Use this Longview Application Framework function in a procedure document to display a customizable prompt that returns a value based on a user's selection. You can use ShowAdvancedPrompt to specify the following:

- Message icon
- Caption
- Message text
- Up to five Buttons, including button text and return values
- The button with focus by default



Example: Advanced Prompt

This function is specific to Longview Apps. The window title of messages and prompts is specified in your .lvapp file with the <description> parameter. The look and feel of messages and prompts is specified in your .lvapp file with the <theme> parameter. For more information, see Configuring Longview Apps.

Note: You must use the ShowAdvancedPrompt function with Set Variable to set the value to the variable to which the function will be returning data. See the syntax example below. You cannot use the ShowAdvancedPrompt function in batch mode.

Syntax:

```
SHOWADVANCEDPROMPT (MessageType, "Caption", "Message", "[DEFAULT;]ButtonText
[;ButtonValue]"[, ... ,"[DEFAULT;]Button5Text[;Button5Value]"])
```

where:

- MessageType determines the icon to display and can be one of the following values:
  - Error
  - Information
  - Warning

- Question
- None

- Caption is the main title text of the message prompt, enclosed in double quotation marks. To omit this parameter, use two double quotation marks ("").
- Message is mandatory and is any additional information to include, enclosed in double quotation marks.
- DEFAULT specifies the button that has focus by default when the message prompt opens. If you do not specify DEFAULT for any button, the first button listed has focus when the message prompt opens.
- ButtonText is mandatory and is the text to display on the button. You must specify at least one button. You can optionally specify up to five buttons in your prompt.
- ButtonValue is the return value if the user selects the button. If no value is specified, ButtonText is the return value.

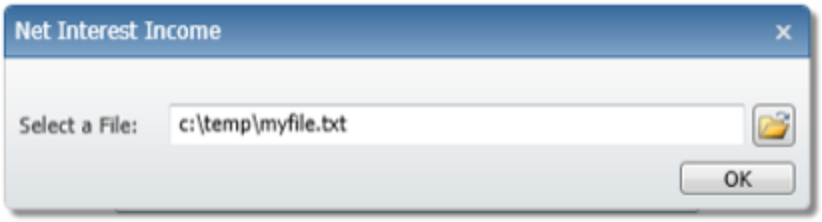
Syntax example:

```
CREATE VARIABLE msgboxResult AS String

SET VARIABLE msgboxResult = SHOWADVANCEDPROMPT(Warning, "Save before
exiting?", "Your data grid has unsaved changes.", "DEFAULT;Save;Save",
"Don’t Save;NoSave", "Cancel;Cancel")
```

ShowFileChooser

Use this Longview Application Framework function in a procedure document to display a customizable prompt that allows users to select a file. This function is specific to Longview Apps. You can use ShowFileChooser to specify the following:

- the button options
- the label for the file chooser
- the parameters of the file chooser



Note: You must use the ShowFileChooser function with Set Variable to set the value to the variable to which the function will be returning data. See the syntax example below. You cannot user the ShowFileChooser function in batch mode.

Syntax:

```
SHOWFILECHOOSER (Button, "Label", "FileRestrictions", "BrowseType",
"defaultPath", "initial")
```

where:

- Button is the set of buttons to display and can be OK or OKCancel. OK displays a single OK button; OKCancel displays both an OK button and a Cancel button. The lvs_buttonClicked variable is populated by the button clicked in the File Chooser.

  Note: In Longview Designer, OK is the only valid button option for data import apps.

- Label is the label to display for the file chooser, enclosed in double quotation marks. If you want to omit this parameter, use two double quotation marks ("").
- FileRestrictions is a list of file type and extension pairs allowed for selection, enclosed in double quotation marks. Separate each file type and extension with a pipe (|). If you want to allow all file types, use two double quotation marks ("").
- BrowseType specifies the title for the browse dialog that opens when users click the folder icon in the file chooser, can be Open or Save, enclosed in double quotation marks.
- defaultPath is the default path selected for the browse dialog, enclosed in double quotation marks. If you want to display the user's current directory, use two double quotation marks ("").

- initial is the initial file selected in the browse dialog that opens when users click the folder icon in the File Chooser, enclosed in double quotation marks. If you don't want a file selected initially, use two double quotation marks ("").

Syntax example:

```
CREATE VARIABLE sFilePath AS String

Set Variable sFilePath = SHOWFILECHOOSER (OK, "Select a File:", "csv
file|*.csv|text file|*.txt", Open, "C:\temp", "C:\temp\myfile.txt")
```

ShowPrompt

Use this Longview Application Framework function in a procedure document to display a prompt that users can interact with. This function is specific to Longview Apps. This function is not used in model documents. The window title of messages and prompts is specified in your .lvapp file with the <description> parameter. The look and feel of messages and prompts is specified in your .lvapp file with the <theme> parameter.

For more information, see Configuring Longview Apps.

Note: You must use the ShowPrompt function with Set Variable to set the value to the variable to which the function will be returning data. See the syntax example below. You cannot use the ShowPrompt function in batch mode.

Syntax:

```
SHOWPROMPT ("Message", Button)
```

where:

- Message is the message to display.
- Button can be one of the following values:

Value	Description
OKCancel	Displays OK and Cancel buttons below the message.
YesNo	Displays Yes and No buttons below the message.
YesNoCancel	Displays Yes, No, and Cancel buttons below the message.

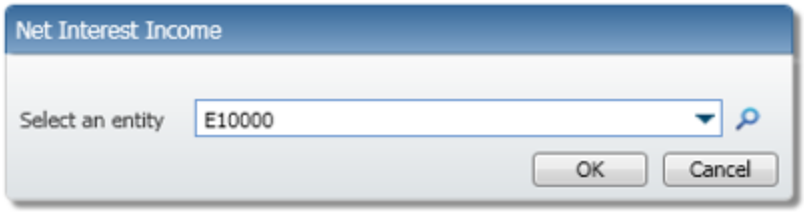
Syntax example:

```
CREATE VARIABLE msgboxResult AS String
SET VARIABLE msgboxResult = SHOWPROMPT("message box test",YesNoCancel)
```

ShowSymbolSelector

Use this Longview Application Framework function in a procedure document to display a customizable prompt that allows users to select a symbol from the Symbol Selector for a specified dimension. You can use ShowSymbolSelector to specify the following:

- the button options
- the label for the Symbol Selector
- the parameters of the Symbol Selector



This function is specific to Longview Apps.

Note: You must use the ShowSymbolSelector function with Set Variable to set the value to the variable to which the function will be returning data. See the syntax example below. You cannot use the ShowSymbolSelector function in batch mode.

Syntax:

```
SHOWSYMBOLSELECTOR (Button, "Label", "dimension", "spec", "initial",
allowLeaf, allowParent, allowReadOnly, allowMultiple, "attributefilters")
```

where:

- Button is the set of buttons to display and can be OK or OKCancel. OK displays a single OK button; OKCancel displays both an OK button and a Cancel button. The lvs_buttonClicked variable is populated by the button clicked in the Symbol Selector.
- Label is the label to display for the symbol selector, enclosed in double quotation marks. If you want to omit this parameter, use two double quotation marks ("").
- dimension is the name of the dimension containing the symbols that should appear in the Symbol Selector, enclosed in double quotation marks.
- spec is the level of symbols to display in the Symbol Selector hierarchy, in relation to the specified symbol, for example, SymbolName### or SymbolName#99. You may also include multiple hierarchical specs separated by a pipe charactcer (|), for example, SymbolName#99|SymbolName2#99. If you want to include all symbols a user has access to, omit this parameter and use two double quotation marks (""). For more information on symbol hierarchy specifications, see Symbol hierarchies.

- initial is the initial symbol selected, enclosed in double quotation marks. If you don't want an initial symbol selected, use two double quotation marks ("").
- allowLeaf specifies whether users can select leaf symbols and can have a value of TRUE or FALSE.
- allowParent specifies whether users can select parent symbols and can have a value of TRUE or FALSE.
- allowReadOnly specifies whether users can select read-only symbols and can have a value of TRUE or FALSE.
- allowMultiple specifies whether users can select multiple symbols and can have a value of TRUE or FALSE.
- attributefilters can be up to two attribute filters linked by AND or OR, enclosed in double quotation marks, using the syntax FilterType{AttrName{Operation{Expression where:

Value	Description
FilterType	specifies the method to use to search the hierarchy for symbols matching the filter criteria and can be one of ALL, PARENT, LEAF, or ROOT. If you specify two attribute filters, FilterType must be the same for both filters.
AttrName	is the name of an attribute.
Operation	can be EQ for exactly equal to or NE for not equal to.
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For Non-List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

If you specify two attribute filters, enclose each filter in parentheses, for example " (attributefilter1) AND (attributefilter2)".

If you don't want to include an attribute filter, use two double quotation marks ("").

Syntax example:

```
Create VARIABLE TargetVar1 AS String

Set VARIABLE TargetVar1 = SHOWSYMBOLSELECTOR (OKCANCEL, "Select an entity",
"Entities", "TENTITIES#99", "TENTITIES", TRUE, TRUE, TRUE, FALSE, "")
```

Syntax example:

```
Create VARIABLE TargetVar2 AS String

Set VARIABLE TargetVar2 = SHOWSYMBOLSELECTOR (OK, "Select a US entity",
"Entities", "", "TENTITIES", TRUE, TRUE, TRUE, FALSE, "ALL{ZGPNativeCurrency
{EQ{CUSD")
```

StrContains

Use this function to verify if the search string exists within a specific string. A result value of 1 is returned if the search string exists; a value of 0 is returned if the search string does not exist.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
STRCONTAINS ("String","SearchString")
```

where:

- String is a character string (enclosed in double quotation marks).
- SearchString is a character string (enclosed in double quotation marks), which will be searched for inside String.

Syntax example:

```
CREATE VARIABLE NumTemp1 AS NUMSET
VARIABLE NumTemp1 = STRCONTAINS ("StringTest","Test")

//NumTemp1 will equal 1
```

StrFind

This function returns a numeric value indicating the starting position within a string of the search string. For example, searching for “ad” within “Nevada” would return a value of 4.

Syntax:

```
STRFIND ("String", "SearchString")
```

Syntax example:

```
CREATE VARIABLE NumTemp1 AS NUMVariable = STRFIND
("SourceString","SearchString")SET VARIABLE NumTemp1 = STRFIND
("StringTest","Test")


// NumTemp1 will equal 7
```

StrLeft

Use this function to extract the leftmost character or characters from a symbol or character string.

You may find the StrLeft function useful for finding individual values from a combined value.

For example, suppose a complete account number in your system appears in the format A11234 — consisting of a prefix code (A), followed by two digits (11) representing an entity code, followed by three digits (234) representing an account code.

You can use the StrLeft function to find the prefix code (the leftmost character) from the complete account number.

Syntax:

```
STRLEFT ("String",NumChars)
```

where:

- "String" is a character string, enclosed in double quotation marks.
- NumChars is a positive integer representing the number of characters to extract from the beginning of the character string. NumChars cannot be greater than the number of characters in the character string.

Syntax example:

```
CodeLet = STRLEFT("A11234",1)
```

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGSET VARIABLE StrTemp1 = STRLEFT
("StringTest",6)


//StrTemp1 will equal "String" (without the double quotation marks)
```

StrLength

Use this function to calculate the length of a character string.

Syntax:

```
STRLENGTH ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Syntax example:

```
NumChars = STRLENGTH ("11234")
```

Code example:

```
CREATE VARIABLE NumTemp1 AS NUMSET VARIABLE NumTemp1 = STRLENGTH
("StringTest")


// NumTemp1 will equal 10
```

StrLower

Use this function to convert text in a symbol or character string to lowercase letters.

You can use the StrLower function to convert all uppercase letters in symbols to lowercase.

Syntax:

```
STRLOWER ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Syntax example:

```
LowerCase = STRLOWER ("A11234")
```

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRLOWER ("SourceString")


//StrTemp1 will equal "stringtest" (without the double quotation marks)
```

StrReplace

Use this function to replace a part of a character string with a different character string.

You may find the StrReplace function useful for comparison purposes. For example, suppose a complete symbol in your system appears in the format A11234. You can use the StrReplace function to change all prefix codes from A to B.

Syntax:

```
STRREPLACE ("String","SearchString","NewString")
```

where:

- String is a character string, enclosed in double quotation marks.
- SearchString is an existing character string.
  - Enclose in double quotation marks.
  - This parameter is case-sensitive and must be typed with the correct combination of uppercase and lowercase letters.
- NewString is a new character string.
  - Enclose in double quotation marks.
  - This parameter is case-sensitive and must be typed with the correct combination of uppercase and lowercase letters.

Syntax example:

```
STRREPLACE (CodeLetB,"A","B")
```

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRREPLACE
("SourceString","SearchFor","ReplaceWith")SET VARIABLE StrTemp1 = STRREPLACE
("StringTest","Test","xxxx")


// StrTemp1 will equal "Stringxxxx" (without the double quotation marks)
```

StrRight

Use this function to extract the rightmost character or characters from a symbol or character string.

You may find the StrRight function useful for finding individual values from a combined value.

For example, suppose a complete symbol in your system appears in the format A11234 — consisting of a prefix code (A), followed by two digits (11) representing an entity code, followed by three digits (234) representing an account code. You can use the StrRight function to find the account code (the three rightmost characters) from the complete symbol.

Syntax:

```
STRRIGHT ("String",NumChars)
```

where:

- Structure is a range of symbols.

  Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

- String is a character string enclosed in double quotation marks.
- NumChars is a positive integer representing the number of characters to extract from the end of the character string. NumChars cannot be greater than the number of characters in the character string.

Syntax example:

```
STRRIGHT (AcctCode,"A11234",3)
```

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRRIGHT
("SourceString",Number)SET VARIABLE StrTemp1 = STRRIGHT ("StringTest",4)


// StrTemp1 will equal "Test" (without the double quotation marks)
```

StrToNum

Use this function to convert a character string to a numeric value.

You may find this function useful for comparison purposes. For example, suppose a complete account number in your system appears in the format A11234. You can use the StrToNum function to change any character strings to numeric values, so they can be compared correctly.

The value provided by your system depends on the format of the source item being converted:

Format of source	Value of TargetSymName	Example
Symbol with numeric value	An error message appears.	String is a valid number.
Empty character string	An error message appears.	Unable to convert string to number.
Character string that cannot change to numeric value	An error message appears.	Unable to convert string to number.

Syntax:

```
STRTONUM ("String")
```

where:

- String is a character string enclosed in double quotation marks.

Syntax example:

```
AcctNum = STRTONUM ("11234")
```

Code example:

```
CREATE VARIABLE NumTemp1 AS NUMVariable = STRTONUM ("SourceString")SET
VARIABLE NumTemp1 = STRTONUM ("1234")


// NumTemp1 will equal 1234 as a numeric value, not a string.
```

StrTrim

Use this function to isolate the text of a string from all leading and trailing spaces.

Syntax:

```
STRTRIM ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRTRIM ("SourceString")SET
VARIABLE StrTemp1 = STRTRIM (" StringTest ")


// StrTemp1 will equal "StringTest" (without the double quotation marks)
```

StrTrimL

Use this function to isolate the text of a string from all leading spaces.

Syntax:

```
STRTRIML ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRTRIML ("SourceString")SET
VARIABLE StrTemp1 = STRTRIML (" StringTest ")


// StrTemp1 will equal "StringTest " (without the double quotation marks)
```

StrTrimR

Use this function to isolate the text of a string from all trailing spaces.

Syntax:

```
STRTRIMR ("String")
```

where:

String is a character string, enclosed in double quotation marks.

Syntax example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRTRIMR ("SourceString")SET
VARIABLE StrTemp1 = STRTRIMR (" StringTest ")


//StrTemp1 will equal " StringTest" (without the double quotation marks)
```

StrUpper

Use this function to convert text in a symbol or character string to uppercase letters.

You may find the StrUpper function useful for comparison purposes. For example, suppose a complete symbol in your system appears in the format A11234, but with either an uppercase or lowercase A. You can use the StrUpper function to convert all lowercase letters in account numbers to uppercase.

Syntax:

```
STRUPPER ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = STRUPPER ("SourceString")SET
VARIABLE StrTemp1 = STRUPPER ("StringTest")

// StrTemp1 will equal "STRINGTEST" (without the double quotation marks)
```

StrValue

Use this function to retrieve a string value from a data intersection and store it to another data intersection, or a string variable. This function will only return a string value for a DataArea if the user has access to that DataArea.

> **Note:** This function will only return string data. If the intersection contains a numeric value, then the function will return an empty string ("").

Syntax:

```
STRVALUE ("DataAreaName1", "SymName1", "SymName2", …, "SymNameN")
```

where:

- DataAreaName is the name of the DataArea in which the symbols are found, and must be used in a procedure, enclosed in double quotation marks.
- SymName is the name of a symbol, enclosed in double quotation marks.

Code example:

```
SALEST### = STRVALUE ("DataArea1", "CASHT###", "Dim2Set", …, "Dim7Set")

SALEST### = STRVALUE ("", "CASHT###", "Dim2Set", …, "Dim7Set")

SET VARIABLE myString = STRVALUE ("DataArea1", "CASHT", "Dim2Set", …,
"Dim7Set")
```

SubStr

Use this function to extract part of a string.

Syntax:

```
SUBSTR (SymName|"String", StartChar, NumChars)
```

where:

- SymName is the name of a symbol.
- String is a character string enclosed in double quotation marks.
- StartChar is the first character of the string that you would like to extract.
- NumChars is the number of characters that you would like to extract.

Code example:

```
CREATE VARIABLE StrTemp1 AS STRINGVariable = SUBSTR
("SourceString",StartNum,Length)SET VARIABLE StrTemp1 = SUBSTR
("StringTest",2,4)


// StrTemp1 will equal "trin" (without the double quotation marks)
```

Sum

Use this function in an equation to calculate total values for a symbol or group of symbols that do not roll up to a specific parent.

In a hierarchy, it is easy to calculate the sum of all child symbols of a parent. However, you may find the Sum function useful to calculate the total of a series of symbols that do not roll up to a specific parent. All symbols must have the same weight (positive, negative, or zero).

Do not confuse the Sum function with the WSum command:

Function	Description
Sum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols have the same weight.
WSum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols do not have the same weight.

Syntax:

```
SUM (Value1[,Value2][,Value3])
```

where:

- Each Value is the name of a NUM list variable, or a numeric value or expression that evaluates to a number.

Syntax example:

```
SUM (Var1, Var2, Var3)
```

SymbolExists

Use this function to assess whether the specified symbol exists in a specified DataArea. A return value of 1 means the symbol exists in the specified DataArea; a return value of 0 means the symbol does not exist in the specified DataArea.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
SYMBOLEXISTS ("dataAreaName", "dimName", "symName")
```

where:

- dataAreaName is the name of the DataArea that contains the symbol to search for.
- dimName is the name of the dimension that contains the symbol to search for.
- symName is the name of the symbol to search for.

TimerElapsed

Use this function to return the elapsed time from the last TIMER START statement in a procedure. The time is returned in seconds, accurate to 3 decimal places. This function is used in conjunction with the TIMER START|STOP command. The TIMER and TIMERELAPSED command and function can be used to provide useful performance and benchmarking data without the overhead of the HISTORY file.

Syntax:

```
TIMERELAPSED()
```

Syntax example:

```
TIMER START
// some code here
 SET VARIABLE nTime = TIMERELAPSED()
        // some code here
SET VARIABLE nTime = TIMERELAPSED()
TIMER START
// some code here
TIMER STOP
```

UserExists

Use this function to determine whether the specified user exists in the database. A return value of 1 means the user exists in the system; a value of 0 means the user does not exist in the system.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
USEREXISTS ("UserID")
```

where:

- UserID is the user ID to search for.

Syntax example:

```
USEREXISTS ("jsmith")
```

Value

Use this function in an equation to get the value from an intersection of data, and to assign that value to an intersection of data, or to use it in a calculation.

Note: This function will only return numeric data. If the intersection contains a string value, then the function will return a zero (0).

Syntax:

```
VALUE ("DataAreaName","SymName0","SymName1", ..., "SymNameN")
```

where:

- DataAreaName is the name of the DataArea in which the symbols are found, enclosed in double quotation marks.
- SymName is the name of a symbol, enclosed in double quotation marks.

Syntax example:

```
VALUE ("daMaster","Acct1", "A1201", "Entity100", …, "WorkingVersion")
```

VariableExists

Use this function to determine whether the specified variable exists in the database. A return value of 1 means the variable exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
VARIABLEEXISTS("Variable")
```

where:

- Variable is the name of the variable to verify the existence of.

Syntax example:

```
VariableExists("sEmployeeType")
```

Developing Longview DataSpecs

Creating DataSpec documents

A DataSpec document consists of functions and comments. There are several Longview Application Framework functions used in DataSpecs as well as other documents, and a few functions specific to DataSpec documents.

Using DataSpec functions

The following lists the functions that are used exclusively used in a DataSpec document. Click the links to view detailed usage and syntax information:

- AdjustedDetail
- AttributeFilter
- Dimension
- Rule
- Schedule
- Sym
- TempSym

The following lists functions that are used in a DataSpec document when used in conjunction with the GET DATAAUDITTRAIL command. Click the links to view detailed usage and syntax information:

- BatchID
- DataChangedThreshold
- DateFilter
- Dimension
- Schedule
- UserFilter

For a list of other commands and functions available for use in DataSpec documents, see Using commands and functions that reference DataSpec documents.

AdjustedDetail

Use this function to indicate whether to download adjusted or unadjusted data.

The AdjustedDetail function in a DataSpec document has the following syntax:

Syntax:

```
AdjustedDetail value
```

where:

- value is one of the following:

Value	Description
UNADJUSTED	To download unadjusted values only to your DataArea.
ADJUSTED	To download adjusted values to your DataArea.

Note: If the AdjustedDetail function is not specified in a DataSpec, only adjusted values are downloaded to the DataArea.

AdjustedDetail Syntax Example

```
ACCOUNTS STAT_ACCOUNTS#99
TIMEPER A1006YTD
ENTITIES LEGAL_ENTITY#99
DETAILS LEDGER#99
CURRENCY Dim4Set
SEGMENTS DIVISIONS#99
ELEMENTS SUMMELE
CONTROLS Dim7Set
ADJUSTEDDETAIL UNADJUSTED
```

AttributeFilter

Use this function to limit the effect of a calculation to symbols that meet the filter condition. This function could also be used when the dimension is restricted in the CalculationBlock which would have the effect of further restricting the symbols for the dimension based on an attribute filter. For information on using AttributeFilter within a CalculationBlock, see AttributeFilter (CalculationBlock).

Syntax:

```
ATTRIBUTEFILTER DimName, FilterType, AttrName,Operation, "Expression"
```

where:

- DimName is the dimension in which calculations will occur.
- FilterType specifies the method to use to search the hierarchy for symbols matching the filter criteria. Select one of the following:

Value	Description
ALL	To find only the symbols whose attributes match the filter criteria, with no descendents.
PARENT	Starting from the top of the hierarchy, to find the symbols whose attributes match the filter criteria, including all descendents. This only checks symbols that are parents to other symbols.
LEAF	Starting from the bottom of the hierarchy, to find the symbols whose attributes match the filter criteria, including all ancestors. This only checks symbols that are leaf symbols; that is, that have no descendents.
ROOT	Searching only the top of the hierarchy to find root symbols whose attributes match the filter criteria, including all descendents.

- AttrName is the name of an attribute.
- Operation can be EQ for exactly equal to or NE for not equal to.
- "Expression" is a character string enclosed within double quotation marks. If no expression is provided, the current symbol being calculated in the calculation block will be used. If the expression is a list, separate multiple items with a pipe (|).

The filter works in the following manner:

- Non-List Attributes:

Parameter	Behavior
Attribute EQ “expression”	Matches only if attribute is an exact match of the “expression”
Attribute NE “expression”	Matches if attribute is not an exact match of the “expression”

- List Attributes

Parameter	Behavior
Attribute EQ “expression”	Matches if attribute is an exact match of the “expression”, or is a list of values, any one of which exactly matches the “expression”
Attribute NE “expression”	Matches if attribute is ■ Empty, or ■ A list of values, none of which exactly matches the “expression”

Syntax example:

```
ATTRIBUTEFILTER Accounts, ALL, ZGPNativeCurrency, EQ, "CUSD"
```

BatchID

Use this function to indicate whether to download audit trail information associated with a particular batch or range of batches. This function only applies to DataSpec documents when used in conjunction

with the GET DATAAUDITTRAIL command.

The BatchID function in a DataSpec document has the following syntax:

Syntax:

```
BatchID value1, TO, value2
```

where:

- value1 indicates the lower range limit of batch IDs to retrieve.
- value2 indicates the upper range limit of batch IDs to retrieved.

Note: To specify a specific BatchID, use the same value for the parameters value1 and value2.

Syntax example:

```
BATCHID 6, TO, 9
BATCHID 123, TO,123
```

DataChangedThreshold

Use this function to indicate whether to download audit trail information associated with a particular threshold. This function only applies to DataSpec documents when used in conjunction with the GET DATAAUDITTRAIL command.

The DataChangedThreshold function in a DataSpec document has the following syntax:

Syntax:

```
DataChangedThreshold operation, value1[, value2]
```

where:

- operation is one of the following:

Value	Description
BETWEEN	is between
EQ	Is exactly equal to
GT	Is greater than
LT	Is less than

- value1 is the value to specify. If only value1 is specified, then this indicates a specific value. If both value1 and value2 are specified, then value1 indicates the lower range threshold for the BETWEEN operation.

- value2 is indicates the upper range threshold when the BETWEEN parameter is specified. This parameter only applies when the operation specified is BETWEEN and must be used in this case

Syntax example:

```
DATACHANGEDTHRESHOLD BETWEEN, 100, 335.5
DATACHANGEDTHRESHOLD GT, 100
DATACHANGEDTHRESHOLD LT, 100
DATACHANGEDTHRESHOLD EQ, 100
```

DateFilter

Use this function to indicate whether to download audit trail information associated with a particular date or range of dates. This function only applies to DataSpec documents when used in conjunction with the GET DATAAUDITTRAIL command.

The DateFilter function in a DataSpec document has the following syntax:

Syntax:

```
DateFilter startRange, TO, endRange
```

where:

- startRange is the starting range for the date in YYYY-MM-DD format.
- endRange is the ending range for the date in YYYY-MM-DD format.

Note: To specify a specific Date, use the same value for the parameters startRange and endRange.

Syntax example:

```
DATEFILTER 2018-09-05, TO, 2018-09-30
DATEFILTER 2018-09-05, TO, 2018-09-05
```

Dimension

Use this function to specify a dimension, its root symbol, and the number of levels in a DataSpec. You can specify more than one symbol for the same dimension.

Syntax:

```
DIMENSION SymName[NumLevels][, SymName[NumLevels]]
```

where:

- DIMENSION is the name of a dimension.
- SymName is the name of a symbol in the dimension.
- NumLevels is optional and is the number of levels down to be included from the symbol.

Syntax Example

```
ACCOUNTS TrialBalance#99
```

Rule

Use this function to specify the server rule for which to download validation data. Use this function in conjunction with the Validation option of the Download (for DataAreas) command.

Syntax:

```
RULE ID
```

where:

- ID is the rule ID to specify.

Syntax example:

```
RULE ID
```

See also

- Download (for DataAreas)

Schedule

Use this function to specify the schedule from which you want to download data in a datalink. You then use the DimSpec function to specify schedule dimensions and symbols.

Use this function in DataSpec documents as follows:

Must be used with...	Dimension
Positioning	Must precede any Dimension statements, and must be followed by Dimension statements for each Dimension included in the schedule.

Syntax:

```
SCHEDULE SchedName
```

```
DIMENSION SymName[NumLevels][, SymName[NumLevels]]
```

where:

- SchedName is a schedule you want to define in the DataSpec.
- DIMENSION is the name of a dimension.
- SymName is the name of a symbol in the dimension.
- NumLevels is the number of levels down to be included from the symbol.

Syntax example:

```
Accounts Accounts_Default
TimePeriods PYr_Total#99
Entities E10000#99
Currencies CAD
Versions Versions_Working
Schedule ICStandard
CONTRAS Total#99
```

Sym

Use this function to assign one or more symbols in a dimension to a temporary symbol. This can be useful when you want to create a visual structure that is a combination of real and temporary symbols.

For information on creating a temporary symbol, see TempSym.

Syntax:

```
SYM DimName, SymName1[,SymName2, ..., SymNameN], TemporaryParent [,WEIGHT,
"weightvalue"]
```

where:

- DimName is the dimension that contains the symbol(s) to assign.
- SymName1[,SymName2, ..., SymNameN] are the symbols to assign to the temporary symbol.

  Note: Symbols appear in the hierarchy in the order in which they were assigned.

- TemporaryParent specifies the temporary symbol to which to assign the symbol(s).

- WEIGHT is optional and specifies that you want to define a weight for the symbol with respect to its parent. If you specify this parameter, you must also specify weightvalue. If you do not specify this parameter, the system assigns a positive weight to the symbol(s).
- weightvalue defines the symbol's relationship with respect to its parent and can be one of the following:

Value	Description
+ or 1	To add the symbol's value to the temporary parent
- or -1	To subtract the symbol's value from the temporary parent.
0	To ensure the symbol has no mathematical effect on the temporary parent.

Note: The weightvalue applies to all symbols assigned using the same SYM statement. If you have symbols that should use different weights, you must write another SYM statement for those symbols

Syntax example:

```
SYM ENTITIES, Maine, NewHampshire, Vermont, Connecticut, NorthEastGroup,
WEIGHT, "+"
```

TempSym

Use this function to create a temporary symbol in the specified dimension.

Syntax:

```
TEMPSYM DimName, SymName, SymType [, "SymDesc"] [, TemporaryParent, Weight]
```

where:

- DimName is the dimension that will contain the new temporary symbol.
- SymName is a name for the new symbol. For guidelines, see Symbol names.
- SymType specifies the way the symbol rolls up to its parent symbol, and can be one of the following:

Value	Description
Standard	To add the symbol into the parent (for example, monthly expenses added into annual expenses).
Carryforward	When the parent value equals the value of the cell immediately preceding it (for example, cash balance as a running year-to-date, where the year-end amount equals the amount for December).

Value	Description
Static	When the child value has no effect on the parent value – for example, for price symbols. (The parents of such symbols are used only to group symbols, not to aggregate their values.) Static symbols do not roll up in any direction, and override any assigned weights.

- "SymDesc" is a character string to be used as a description of SymName, in English, enclosed in double quotation marks. For guidelines, see Symbol descriptions.
- TemporaryParent is optional and specifies the temporary parent symbol to which to assign SymName. If you do not specify a parent, the is created as a root symbol in the dimension specified for DimName.
- Weight defines the symbol's relationship with respect to its parent and can be one of the following:

Value	Description
+ or 1	To add the symbol's value to its parent
- or -1	To subtract the symbol's value from its parent.
0	To ensure the symbol has no mathematical effect on it's parent.

Syntax example:

```
TEMPSYM TIMEPER, CTEMP1, STANDARD, "Temp Time Period Symbol One", CTEMP, 1
```

UserFilter

Use this function to indicate whether to download audit trail information associated with a particular user. This function only applies to DataSpec documents when used in conjunction with the GET DATAAUDITTRAIL command.

Syntax:

```
UserFilter userID
```

where:

- userID is the user to specify.

Syntax example:

```
USERFILTER user123
```

Using commands and functions that reference DataSpec documents

In addition to the functions used only with DataSpec documents (see Using DataSpec functions) the following Longview Application Framework commands and functions can reference a DataSpec document:

- Copy Data command
- Create DataArea command
- Delete Lock command
- Get DataAuditTrail command
- Set WorkflowStatus command

For more information on these and other commands and functions, see Developing Longview Procedures and Developing Longview Models.

Sample DataSpec document

The following example illustrates a DataSpec document named Forcast.lvdsp, establishes a DataArea called Fordata, and downloads dimensions, symbols, structures, data, etc., to the local memory of the user's computer. The DataArea can consist of multi-level symbol Structures for all dimensions. It is not limited to three active dimensions.

Note: To add a comment to syntax, insert two forward slashes (//) at the beginning of the comment.

The following rules apply to the use of DataAreas within a DataSpec created by another script:

- If a DataArea is created in the parent script, it can be used in a child script (i.e., a child script inherits all the DataAreas created in its parent scripts).
- If a DataArea is created in the child script, it is visible in that script and all of its child scripts only.
- If an inherited DataArea is deleted in the child script, the data will not be cleared from the memory and it is still accessible from the parent script. However, the child script will lose its reference to that DataArea.
- Downloading or Uploading an inherited DataArea in a child script has the same effect as doing so in the main script.

The following commands are executed in a Procedure document:

```
CREATE DATAAREA FORDATA USING Forcast.lvdsp
CREATE LOCK USER USING Forcast.lvdsp "ForData"
```

```
DOWNLOAD FORDATA
```

Syntax example:

```
// DataSpec sample document comment
ATTRIBUTEFILTER TIMEPER,ALL,AZSSIND,EQ, "FALSE"
SCHEDULE SRG08
TempSym TIMEPER, CTEMP, STANDARD, "Temp Time Period Symbol Parent"
TempSym TIMEPER, CTEMP1, STANDARD, "Temp Time Period Symbol #1", CTEMP, +
ACCOUNTS BALSHEET #99
TIMEPER AYR07#99
ENTITIES W11110#99, W1110#45
DETAILS P1000#99
CURRENCY CCAD
SCHEDDIM1 TOTAL#99
CONTROLS DIM5SET
```

The following commands are executed in a Procedure document:

```
RUN MODEL FORCPY.lvmod ON FORDATA
UPLOAD FORDATA
DELETE LOCK USER USING Forcast.lvdsp MATCH
```

Developing Longview JEQuerySpecs

Creating JEQuerySpec documents

A JEQuerySpec document consists of functions and comments. The JEQuerySpec document is used in conjuction with the Application Framework commands GET JEHEADERS and GET JEDETAILS to query specific journal entry headers and details from the data server repository.

Using JEQuerySpec functions

The following lists the functions that are used in a JEQuerySpec document:

- ApplicationID
- Category
- Calculated
- Created
- CreationPeriod
- Deleted
- Dimension
- ID
- LastModified
- NonShared
- PostPeriod
- Status
- Type

AppliationID

Use this function to retrieve journal entries by Application ID.

The ApplicationID function in a JEQuerySpec document has the following syntax:

Syntax:

```
ApplicationID “value”
```

where:

- value is the Application ID to specify.

Syntax example:

```
APPLICATIONID "TEST"
APPLICATIONID "12345"
```

Category

Use this function to retrieve journal entries of a specific category.

The Category function in a JEQuerySpec document has the following syntax:

Syntax:

```
Category EQ|NE, jeCategory
```

where:

- EQ indicates exactly "equal to"
- NE indicates "not equal to"
- jeCategory is one of the following:

Value	Description
STANDARD	To query standard journal entries
ELIMINATION	To query elimination journal entries

Syntax example:

```
CATEGORY EQ STANDARD
CATEGORY NE ELIMINATION
```

Calculated

Use this function to retrieve calculated journal entries.

The Calculated function in a JEQuerySpec document has the following syntax:

Syntax:

```
Calculated status
```

where:

- status is one of the following:

Value	Description
INCLUDE	To include calculated journal entries. This is the default if the Calculated function is not specified.
EXCLUDE	To exclude calculated journal entries.

Syntax example:

```
CALCULATED EXCLUDE
```

Created

Use this function to retrieve journal entries created on a specific date or range of dates.

The Created function in a JEQuerySpec document has the following syntax:

Syntax:

```
Created startRange, TO, endRange
```

where:

- startRange is the starting range for the created date in YYYY-MM-DD format.
- endRange is the ending range for the created date in YYYY-MM-DD format.

Note: To specify a specific created date, use the same value for the parameters startRange and endRange.

Syntax example:

```
CREATED 2018-01-01, TO, 2018-01-01
CREATED 2018-01-01, TO, 2018-12-31
```

CreationPeriod

Use this function to retrieve journal entries created between certain periods of time.

The CreationPeriod function in a JEQuerySpec document has the following syntax:

Syntax:

```
CreationPeriod startSymbol, TO, endSymbol
```

where:

- startSymbol is the starting range for the creation period. It must be the name of a time period symbol.
- endSymbol is the ending range for the creation period. It must be the name of a time period symbol.

```
To specify a specific creation period, use the same value for the parameters
startSymbol and endSymbol.
```

Syntax example:

```
CREATIONPERIOD A1801_YTD, TO, A1801_YTD
CREATIONPERIOD A1801_YTD, TO, A1812_YTD
```

Deleted

Use this function to retrieve deleted journal entries. This option applies only if your system has been configured to save deleted journal entries.

The Deleted function in a JEQuerySpec document has the following syntax:

Syntax:

```
Deleted status
```

where:

- status is one of the following:

Value	Description
INCLUDE	To retrieve all JEs that have been assigned a journal entry ID.
EXCLUDE	To retrieve journal entries that have not been deleted. This is the default if the Deleted function is not specified.
ONLY	To retrieve only journal entries that have been deleted.

Syntax example:

```
DELETED EXCLUDE
```

Dimension

Use this function to specify a dimension and a symbol in a JEQuerySpec to retrieve journal entries containing specific symbols from certain dimensions. You can specify more than one symbol for the same dimension.

Syntax:

```
DIMENSION SymName
```

where:

- DIMENSION is the name of a dimension.
- SymName is the name of a symbol in the dimension

Syntax example:

```
ENTITIES E11111
ENTITIES E12345
ACCOUNTS A11220
```

ID

Use this function to retrieve journal entries by journal entry ID or within a range of contiguous IDs.

Syntax

The ID function in a JEQuerySpec document has the following syntax:

```
ID value1, TO ,value2
```

where:

- value1 is the lower range limit of journal entry IDs to retrieve.
- value2 is the upper range limit of journal entry IDs to retrieve.

Note: To specify a specific journal entry ID, use the same value for the parameters value1 and value2.

Syntax example:

```
ID 123, TO, 123
ID 1, TO, 8
```

LastModified

Use this function to retrieve journal entries last modified between certain dates.

The LastModified function in a JEQuerySpec document has the following syntax:

Syntax:

```
LastModified startRange, TO, endRange
```

where:

- startRange is the starting range for the last modified date in YYYY-MM-DD format.
- endRange is the ending range for the last modified date in YYYY-MM-DD format.

Note: To specify a specific last modified date, use the same value for the parameters startRange and endRange.

Syntax example:

```
LASTMODIFIED 2018-01-01, TO, 2018-01-01
LASTMODIFIED 2018-01-01, TO, 2018-12-31
```

NonShared

Use this function to retrieve non-shared journal entries.

The NonShared function in a JEQuerySpec document has the following syntax:

Syntax:

```
NonShared status
```

where:

- status is one of the following:

Value	Description
INCLUDE	To include non-shared journal entries.
EXCLUDE	To exclude non-shared journal entries. This is the default if the NonShared function is not specified.

Syntax example:

```
NONSHARED INCLUDE
```

PostPeriod

Use this function to retrieve journal entries posted between certain periods of time.

The PostPeriod function in a JEQuerySpec document has the following syntax:

Syntax:

```
PostPeriod startSymbol, TO, endSymbol
```

where:

- startSymbol is the starting range for the post period. It must be the name of a time period symbol.
- endSymbol is the ending range for the post period. It must be the name of a time period symbol.

Note: To specify a specific post period symbol, use the same value for the parameters startSymbol and endSymbol.

Syntax example:

```
POSTPERIOD A1801_YTD, TO, A1801_YTD
POSTPERIOD A1801_YTD, TO, A1812_YTD
```

Status

Use this function to retrieve journal entries with a specific status.

The Status function in a JEQuerySpec document has the following syntax:

Syntax:

```
Status EQ|NE, jeStatus
```

where:

- EQ indicates exactly “equal to”
- NE indicates “not equal to”
- jeStatus is one of the following:

Value	Description
NoStatus	To retrieve journal entries with No Status.
Posted	To retrieve journal entries that have been posted.
Validated	To retrieve journal entries that have been validated.
Errors	To retrieve journal entries with errors (i.e.: journal entries that did not validate).
Review	To retrieve journal entries with a status of Review.

Syntax example:

```
STATUS EQ, POSTED
STATUS NE, REVIEW
```

Type

Use this function to retrieve journal entries of a specific type.

The Type function in a JEQuerySpec document has the following syntax:

Syntax:

```
Type EQ|NE, jeType
```

where:

- EQ indicates exactly “equal to”
- NE indicates “not equal to”
- jeType is one of the following:

Value	Description
CurrentPeriod	To retrieve journal entries that were created as current period JEs.
Restatement	To retrieve all restatement journal entries.
PPA	To retrieve all PPA journal entries.
FuturePeriod	To retrieve all journal entries that were created as future period JEs.

Syntax example:

```
TYPE EQ, “CURRENTPERIOD”
TYPE NE, “CURRENTPERIOD”
```

Using commands and functions that reference JEQuerySpec documents

The following Longview Application Framework commands can reference a JEQuerySpec document:

- Get JEHeaders
- Get JEDetails

For more information on these and other commands and functions, see Developing Longview Procedures and Developing Longview Models.

Sample JEQuerySpec document

The following example illustrates a JEQuerySpec document named JEQuery.lvqje.

Note: To add a comment to syntax, insert two forward slashes (//) at the beginning of the comment.

JEQuerySpec Syntax Example

```
// Sample comment


CATEGORY EQ, STANDARD
TYPE EQ, CURRENTPERIOD
ID 1, TO, 8
STATUS EQ, POSTED
CALCULATED INCLUDE
DELETED EXCLUDE
CREATIONPERIOD A1706_YTD, TO, A1706_YTD
POSTPERIOD A1706_YTD, TO, A1706_YTD
APPLICATIONID “TEST”
NONSHARED EXCLUDE
ENTITIES E1000
```

Developing Longview CalculatedJESpecs

Creating CalculatedJESpec documents

Journal entries are used to record adjustments to the consolidation process in the Longview system. A calculated journal entry is a journal entry that is created by a calculation, instead of manually through the Journal Entries client. Calculated journal entries are used to record recurring adjustments; for example, a sales bonus adjustment that occurs every month.

Note: To view created calculated journal entries, query all journal entries in the Journal Entries client. For more information, see the Longview Journal Entries Guide.

A CalculatedJESpec document in Longview Application Framework is an ASCII file that is used to specify information about a calculated journal entry such as the journal entry type and description, as well as a line of detail for each adjustment that the calculated journal entry contains. The recommended extension for CalculatedJESpecs is .lvcje.

You must create a CalculatedJESpec document before you can create, delete, post, or unpost the specified calculated journal entry with Longview Application Framework functions. You can create and edit CalculatedJESpecs in your preferred text editor.

A CalculatedJESpec document has the following general syntax:

General Syntax

```
ApplicationID "AppID"

[Type JEType]
PostPeriod JEPostPeriod
[Subcategory "JESubcategory"]
Description "JEDescription"
[Notes "JENotes"]
[Financial FinancialJE]
Shared SharedJE]
Detail DetailType, DetailValue, DetailAreaSpec
```

where:

- AppID is the unique ID to assign to the calculated journal entry, enclosed in double quotation marks. Specify an alphanumeric character string up to 31 characters in length.
- JEType is an optional parameter for the type of calculated journal entry to create, and can be one of the following:

Value	Description
CURRENTPERIOD	To create a calculated journal entry that adjusts balances in the current period. This is the default value.
RESTATEMENT	To create a calculated journal entry that restates balances in the current period or a prior period. Restatement journal entries are used for changes in accounting rules.
PPA	To create a calculated journal entry that adjusts balances in a prior period.
FUTUREPERIOD	To create a calculated journal entry that adjusts balances in a future period.

- JEPostPeriod is the time period symbol for the posting period of the calculated journal entry. The valid values for this parameter depend on the value you specified for JEType:

Value	Description
If JEType is CURRENTPERIOD...	This parameter is optional. Specify the current period. If you do not specify a value for this parameter, the time period symbol is set by the SGPCurrentPeriod system attribute. For more information, see the Longview Application Administrator Guide.
If JEType is RESTATEMENT...	This parameter is optional. Specify the current period or a time period symbol prior to the current period. If you do not specify a value for this parameter, the time period symbol is set by the SGPCurrentPeriod system attribute. For more information, see the Longview Application Administrator Guide.
If JEType is PPA...	This parameter is mandatory. Specify a time period symbol prior to the current period.
If JEType is FUTUREPERIOD...	This parameter is mandatory. Specify a time period symbol after the current period.

- JESubcategory is an optional parameter for the subcategory to assign the calculated journal entry to, enclosed in double quotation marks. If you do not specify a value for this parameter, the calculated journal entry subcategory is set to Adjustment. For more information on subcategories, see the *Longview Application Administrator Guide*.

- JEDescription is a brief description of the calculated journal entry, enclosed in double quotation marks. Specify a description up to 100 characters in length.

- JENotes is an optional parameter that specifies any detailed notes for the calculated journal entry, enclosed in double quotation marks. Specify notes up to 1280 characters in length.

- FinancialJE is an optional parameter that specifies whether the calculated journal entry is financial or non-financial. Specify one of the following values:

Value	Description
TRUE	To create a financial calculated journal entry. Financial journal entries must be balanced (credit values must balance debit values). This is the default value.
FALSE	To create a non-financial calculated journal entry. Non-financial journal entries do not have to be balanced (credit values do not have to balance debit values).

- SharedJE is an optional parameter that specifies whether the calculated journal entry is shared or non-shared. Specify one of the following values:

Value	Description
TRUE	To create a shared calculated journal entry. Shared journal entries can be viewed, posted, unposted, and deleted by other users, depending on their authorization.
FALSE	To create a non-shared calculated journal entry. Non-shared journal entries can be viewed, posted, unposted, and deleted by only the user that created the journal entry.

  If you do not specify a value for SharedJE, the value set for the SJEDefaultShared system attribute determines whether the calculated journal entry is shared. For more information, see the Longview Application Administrator Guide.

- DetailType is the type of detail line to create in the calculated journal entry. Specify one of the following values:

Value	Description
DEBIT	To create a detail line where the specified value is debited.
CREDIT	To create a detail line where the specified value is credited.

Note: One DetailType, DetailAreaSpec, and DetailValue must be specified for each detail line. If FinancialJE is set to TRUE, the calculated journal entry must have at least two lines of detail that balance.

- DetailValue is the adjustment amount for the detail line. Specify a numeric value. You can also define a numeric value for DetailValue using a numeric variable. For more information, see Create Variable and Set Variable.

- DetailAreaSpec is the data area specification for the detail line. Specify one symbol name for each dimension in your system (except for the time periods dimension, usually the dimension named TIMEPER), delimited by commas. The time period symbol for DetailAreaSpec is automatically set to the symbol as specified for JEPostPeriod. The data area specification for DetailAreaSpec is also dependent on the subcategory set for JESubcategory:

Value	Description
If JESubcategory is set to a subcategory that is associated with a schedule and allows reclassification...	The data area can contain either of the following: ■ Base dimensions only or ■ Base dimensions and the schedule dimensions from the schedule used by the specified subcategory.
If JESubcategory is set to a subcategory that is associated with a schedule and does not allow reclassification...	The data area must contain base dimensions and the schedule dimensions from the schedule used by the specified subcategory.
If JESubcategory is not set to a value, is set to Adjustment, or is set to a subcategory that is not associated with a schedule...	The data area can contain base dimensions only.

If your data area contains schedule dimensions, you can specify one symbol name for each extra dimension as set in the schedule definition. Symbol names for extra dimensions must be included after the symbol names for the base dimensions in your system. For more information on subcategories and schedules, see the *Longview Application Administrator Guide*.

CalculatedJESpec Document Example

```
ApplicationID CalcJE3
Type CURRENTPERIOD
PostPeriod P03YTD

Description "Calculated Journal Entry 3"
Notes "The third calculated journal entry"
Subcategory "Adjustment"
Financial TRUE
Shared FALSE

Detail CREDIT, 100, Cash, TORONTO, DIM3SET, DIM4SET, DIM5SET, DIM6SET,
DIM7SET
Detail DEBIT, 100, AccountsReceivable, TORONTO, DIM3SET, DIM4SET, DIM5SET,
DIM6SET, DIM7SET
```

Using calculated journal entry functions

In order to use any of the calculated journal entry functions, you must also create a CalculatedJESpec document. For more information, see Creating CalculatedJESpec documents.

In addition, the following functions are used in conjunction with calculated journal entry functions:

- Create Variable. For more information, see Create Variable.
- Set Variable. For more information, Set Variable.

JournalEntryCreate

Use this function to create a calculated journal entry.

Users can utilize the journal entry system to record adjustments to the consolidation process. Some journal entries are required once only; others are required every month. A calculated journal entry (JE) is a recurring journal entry that you can prepare to run every month.

Creating a calculated journal entry with the JournalEntryCreate function includes the following steps:

1. Create a CalculatedJESpec document. For more information, see Creating CalculatedJESpec documents.
2. Enter the JournalEntryCreate function. For more information, see Syntax.

> **Note:** To create a calculated journal entry, you must have the applicable authorization. For more information on journal entries authorization, see the Longview Application Administrator Guide.

After you have created the CalculatedJESpec document, you can use the JournalEntryCreate function to create the calculated journal entry.

The JournalEntryCreate function returns the following values:

Value	Description
-1	An error occurred, and the calculated journal entry was not created successfully. **Note:** For more details about an error, you can view the LVS_ERRORCODE and LVS_ERRORMESSAGE system variables. For more information, see Working with system variables.
0	The calculated journal entry was created successfully.

Syntax

```
Set VARIABLE VariableName = JournalEntryCreate("CalculatedJESpec")
```

where:

- VariableName is a variable of type NUM as created by the Create Variable command. For more information, see Create Variable.
- CalculatedJESpec is the name of the text file that contains the CalculatedJESpec document, enclosed in double quotation marks. For more information, see Creating CalculatedJESpec documents.

JournalEntryDelete

Use this function to delete a calculated journal entry.

Users can utilize the journal entry system to record adjustments to the consolidation process. Some journal entries are required once only; others are required every month. A calculated journal entry (JE) is a recurring journal entry that you can prepare to run every month.

Note: A calculated journal entry that is temporarily posted must be unposted before it can be deleted. For more information see JournalEntryUnPost. Permanently posted journal entries cannot be deleted.

Syntax:

```
Set VARIABLE VariableName = JournalEntryDelete("CalculatedJESpec")
```

where:

- VariableName is a variable of type NUM as created by the Create Variable command. For more information, see Create Variable.
- CalculatedJESpec is the name of the text file that contains the CalculatedJESpec document, enclosed in double quotation marks. For more information, see Creating CalculatedJESpec documents.

Note: If SharedJE is set to FALSE in the CalculatedJESpec file, the calculated journal entry can be deleted by the user that created it only.

Syntax example:

```
CREATE VARIABLE SuccessVar AS NUM

SET VARIABLE SuccessVar = JournalEntryDelete("CalcJESpec.lvcje")
```

See also

- JournalEntryCreate
- Create Variable
- Set Variable

Syntax example:

```
CREATE VARIABLE SuccessVar AS NUM
SET VARIABLE SuccessVar = JournalEntryCreate("CalcJESpec.lvcje")
```

See also

- Create Variable
- Set Variable

JournalEntryPost

Use this function to post a calculated journal entry.

Users can utilize the journal entry system to record adjustments to the consolidation process. Some journal entries are required once only; others are required every month. A calculated journal entry (JE) is a recurring journal entry that you can prepare to run every month.

In order to post a calculated journal entry, the following requirements must be met:

- The calculated journal entry (as specified in the CalculatedJESpec document) must be valid. For more information, see Creating CalculatedJESpec documents.
- You must have authorization to post journal entries. For more information on journal entries authorization, see the *Longview Application Administrator Guide*.

Syntax:

```
Set VARIABLE VariableName = JournalEntryPost("CalculatedJESpec",
PostingMethod)
```

where:

- VariableName is a variable of type NUM as created by the Create Variable command. For more information, see Create Variable.
- CalculatedJESpec is the name of the text file that contains the CalculatedJESpec document, enclosed in double quotation marks. For more information, see Creating CalculatedJESpec documents.

  Note: If SharedJE is set to FALSE in the CalculatedJESpec file, the calculated journal entry can be posted by the user that created it only.

- PostingMethod can be one of the following:

Value	Description
TEMPORARY	To temporarily post the calculated journal entry. Temporarily posted journal entries can be unposted.
PERMANENT	To permanently post the calculated journal entry. **Caution:** Permanently posted journal entries cannot be unposted.

Syntax example:

```
CREATE VARIABLE SuccessVar AS NUM

SET VARIABLE SuccessVar = JournalEntryPost("CalcJESpec.lvcje", TEMPORARY)
```

See also

- JournalEntryCreate
- JournalEntryUnPost
- Create Variable
- Set Variable

JournalEntryUnPost

Use this function to unpost a calculated journal entry. You can unpost a temporarily posted calculated journal entry if you want to delete it or modify it.

Users can utilize the journal entry system to record adjustments to the consolidation process. Some journal entries are required once only; others are required every month. A calculated journal entry (JE) is a recurring journal entry that you can prepare to run every month.

Note: Permanently posted calculated journal entries cannot be unposted.

Syntax:

```
Set VARIABLE VariableName = JournalEntryUnPost("CalculatedJESpec")
```

where:

- VariableName is a variable of type NUM as created by the Create Variable command. For more information, see Create Variable.
- CalculatedJESpec is the name of the text file that contains the CalculatedJESpec document, enclosed in double quotation marks. For more information, see Creating CalculatedJESpec documents.

  Note: If SharedJE is set to FALSE in the CalculatedJESpec file, the calculated journal entry can be unposted by the user that created it only.

Syntax example:

```
CREATE VARIABLE SuccessVar AS NUM

SET VARIABLE SuccessVar = JournalEntryUnPost("CalcJESpec.lvcje")
```

See also

- JournalEntryCreate
- JournalEntryPost
- Create Variable
- Set Variable

Developing Longview AuditQuerySpecs

Creating AuditQuerySpec documents

An AuditQuerySpec document consists of functions and comments. The AuditQuerySpec document is used in conjunction with the Application Framework commands GET METADATAAUDITRAIL and RESET METADATAAUDITTRAIL to query specific metadata audit details, and to reset specific metadata audit trail activity.

Using AuditQuerySpec functions

The following table lists the functions that are used in a AuditQuerySpec document. Click the links to view detailed usage and syntax information.

- CategoryFilter
- DateFilter
- UserFiter

CategoryFilter

Use this function to retrieve metadata or reset audit trail activity for a subset of categories. The Category function in a AuditQuerySpec document has the following syntax:

Syntax:

```
CategoryFilter Category1|Catetgory2|..|CategoryN
```

where:

- CategoryN is one of the following:

Field	Description
ALL	All Activity Types
ATTRIBUTE	Attribute Maintenance Activity
AUTHORIZATION	Authorization Maintenance Activity
GROUP	Group Maintenance Activity
GROUPMEMBERSHIP	Group Membership Activity
METADATA	Metadata Audit Trail Activity. Applies to GET METADATAAUDITTRAIL only.
ROLE	Role Maintenance Activity
SYMBOL	Symbol Maintenance Activity
USER	User Maintenance Activity

Syntax example:

```
CATEGORY USER|GROUP|GROUPMEMBERSHIP
```

DateFilter

Use this function to indicate whether to download metadata audit trail information associated with a particular date or range of dates.

The DateFilter function in an AuditDataSpec document has the following syntax:

Syntax:

```
DateFilter startRange, TO, endRange
```

where:

- startRange is the starting range for the date in YYYY-MM-DD format.
- endRange is the ending range for the date in YYYY-MM-DD format.

UserFilter

Use this function to indicate whether to filter metadata audit trail information associated with a particular user who initiated the metadata activity.

Syntax:

```
UserFilter userID|ALL
```

where:

- userID is the user to specify.
- ALL specifies that metadata audittrail should be retrieved for all users who initiated metadata activity.

Note: To specify a specific Date, use the same value for the parameters startRange and endRange.

Syntax example:

```
USERFILTER user123
USERFILTER ALL
```

Using commands and functions that reference AuditQuerySpec documents

The following Longview Application Framework commands can reference a AuditQuerySpec document:

- Get MetadataAuditTrail
- Reset MetadataAuditTrail

For more information on these and other commands and functions, see Developing Longview Procedures.

Sample AuditQuerySpec document

The following example illustrates an AuditQuerySpec document named AuditQuery.lvaud.

Note: To add a comment to syntax, insert two forward slashes (//) at the beginning of the comment.

AuditQuerySpec Syntax Example

```
// Sample comment


DATEFILTER 2018-09-05, TO, 2018-09-30
CATEGORYFILTER USER|GROUP|GROUPMEMBERSHIP
USERFILTER ALL
```

Developing Longview Models

Creating Model documents

A Model document in Longview Application Framework contains calculation rules for your Longview system to create or change data. It modifies the data by calculating the value of symbols according to rules you specify. Your Longview system can then use the calculated data in Reports and other documents.

Note: To add a comment to syntax, insert two forward slashes (//) at the beginning of the comment.

A Model document can run any calculation, including financial functions such as Net Present Value (NPV), and conditional or simultaneous (recursive or circular) equations.

```
// Model Document - to run functions

CALCULATIONBLOCK(TIMEPER;ACCOUNTS SALES1)
PO1YTD = 111
END CALCULATIONBlock

///CALCULATIONBLOCK(ACCOUNTS;TIMEPER PO1YTD)
///SALES1 = 111
///END CALCULATIONBlock
```

A Model document can only perform calculations on a single DataArea, although it can use values from different DataAreas as a source for the calculations.

Use a Model document to:

- perform calculations
- run the same calculations for data stored in several DataAreas
- provide more flexibility in specifying calculation types when you need to perform calculations in non-standard order.

Parts of a Model document

A Model document must contain the following types of function statements within the document body

Statement	Description
CalculationBlock statement	Applies subsequent calculations to symbols in a specified dimension only.
Calculation statement	Specifies the required calculations.

The CalculationBlock function is used to define the DataArea to which subsequent calculations are applied. A CalculationBlock statement stays in effect until the End CalculationBlock is executed. All calculation statements that pertain to the symbols specified within the CalculationBlock statement must appear after the CalculationBlock and before the End CalculationBlock statements.

The CalculationBlock statement parameters define the symbol ranges for all dimensions to limit or restrict the calculated DataArea.

The calculated dimension determines the symbols used for the subsequent calculation statements.

```
CalculationBlock (CalcDim[;RestrictedDim SymName
[Structure][,SymName[Structure]]][;RestrictedDim SymName
[Structure][,SymName[Structure]]])
End CalculationBlock
```

where:

- CalcDim defines the dimension that will be used in the calculation statements.
- RestrictedDim is a dimension that will define a limited or restricted calculated DataArea.
- SymName refers to the symbol name that will define a limited or restricted calculated DataArea.

  Note: To specify a range of symbols to limit the calculation to, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

- [Structure] is optional and is the symbol structure that will define a limited or restricted set of symbols.

Sample Model document

This example illustrates a Model document named ForCpy.lvmod. This Model document defines several calculation blocks and calculation statements.

The following command is executed in a Procedure document:

Sample Procedure Document

```
RUN MODEL ForCpy.lvmod ON FORDATA
```

The following is executed in the ForCpy.lvmod Model document:

Sample Model Document

```
// Model sample document comment
```

```
 CalculationBlock (TIMEPER; ACCOUNTS BALSHEET#99; ENTITIES W11110#99;
DETAILS P1000#99)

A31120=0

End CalculationBlock

CalculationBlock (ACCOUNTS; CURRENCY CCAD)
TRANSFER COMMON,A0701,COMMON,COMMON,CCAD,DIM5SET TO
COMMON,F0702,COMMON,COMMON,CCAD,DIM5SET FROM "ACTDATA"

End CalculationBlock


CalculationBlock (TIMEPER; ACCOUNTS BALSHEET#99; ENTITIES W11110#99; DETAILS
P1000#99)

WHEN F0701 EQ 99

F0703=55

END WHEN

End CalculationBlock
```

Using Model Functions

This topic contains the Longview Application Framework functions you can use in a model. Click the links in the following table for detailed usage and syntax information.

For information on using Longview Application Framework functions in a procedure, see Using Procedure functions.

Abs	IsLocked	ScheduleExists
Allocate	IsMaintenanceOn	Series
And	IsMaintenanceUser	Simult
Attribute	IsName	SkipNoData
AttributeFilter (CalculationBlock)	IsParentSymbol	StrContains
Avg	IsReadOnly (DataArea)	StrFind
BaseValue	IsReadOnly (Symbol)	StrLeft
By	JDate	StrLength
CalculationBlock	JDiff	StrLower
Call	JRoll	StrReplace
Count	JValue	StrRight
Depr	Lag, Lead	StrToNum
Exp	Log	StrTrim
FileExists	LookUp (Average)	StrTrimL
Filter	LookUp (Match)	StrTrimR
FValue	LookUp (Range)	StrUpper
GetDate	LookUp (Total)	StrValue
GetLockComment	Max	SubStr
GetLockUser	Min	Sum
GetStatus	Neg	SumArea
GetTime	NPV	Symbol
GroupExists	NumToStr	SymbolExists
Grow	Operators (arithmetic)	SymbolRange
Growth	Operators (relational)	Transfer
If	Operators (conditional)	UserExists

IRR	Or	Value
IsAreaBatchStatus	PGrow	VariableExists
IsAreaEventStatus	Pos	When
IsChanged	Rollup	WSum
IsConnected	RollupActive	
IsKeyword	Round	
	RuleExists	

Abs

Use this function in an equation to evaluate an expression and to calculate its absolute value. Absolute value is the expression of a number as a positive value, regardless of whether it is actually positive or negative.

You may find the Abs function useful for learning the deviation between planned and actual results.

For information on using this function in a procedure, see Abs.

Syntax:

```
ABS (SymName[Structure])
```

where:

- SymName is the name of a symbol for which a value is to be returned.
- Structure is a range of symbols. The structure for the source and target must be the same.

To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
AVar = ABS (AYR00 - AYR01)
```

Allocate

Use this function in an equation to allocate a proportional set of values across another symbol.

Note: Only those symbols to which the user has access will be included in the allocation.

You can specify allocations in the dimension specified in a By statement (for example, By TIMEPER).

Syntax:

```
Allocate (IntersectSym, PatternSym[, PatternSourceSym][, Precision[,
PlugSymName]])
```

where:

- IntersectSym is a symbol in the BY dimension that intersects with the target symbol. The cell where they intersect is the source value. This source value is a total value to be allocated.
- PatternSym is the symbol that contains the pattern used for the allocation.
- PatternSourceSym is used when the source of the pattern is different than the target symbols in the BY dimension. This symbol specification and the number of symbols must match the number specified in the calculation block.
- Precision is the number of digits after the decimal.
- PlugSymName is the symbol that receives any difference (referred to as the plugged amount) due to rounding. Precision and PlugSymName relate to each other in the following ways:

If...	Then...
Precision is not specified	no rounding occurs, and there is no plugged amount to allocate
Precision is specified but PlugSymName is not specified	rounding occurs, and the highest target value gets the plugged amount
Precision and PlugSymName are both specified	rounding occurs and PlugSymName gets the plugged amount

Syntax example:

```
A57100 = ALLOCATE (AYR01, A41000, 1, COL3)
```

And

See Operators (arithmetic).

Attribute

Use this function to determine whether the specified attribute exists in the database. A return value of 1 means the attribute exists in the system; a value of 0 means that it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
ATTRIBUTEEXISTS ("AttrClass", "AttrName")
```

where:

- AttrClass is the attribute class that contains the attribute to search for.
- AttrName is the name of the attribute, enclosed in double quotation marks.

Syntax example:

```
AttributeExists("SYSTEM", "AZGPCurrentPeriodPAC")
```

AttributeFilter (CalculationBlock)

You can use an AttributeFilter to limit the effect of a calculation to symbols that meet the filter condition.

Within a model, an AttributeFilter is embedded in a CalculationBlock, does not use the function keyword, and uses left braces as delimiters ({) containing the filter settings. Attribute filters can be used as AND or OR expressions. Attribute filters that immediately follow each other behave like AND expressions, and apply to the symbols that follow the expression. Attribute filters behave like OR expressions if the same symbol specification is used for two separate attributes.

Syntax:

```
CALCULATIONBLOCK (CalculationDim[;RestrictedDim] {"FilterType",
"AttributeName", "FilterOperation", "Expression"}[, SymName])

//statements for this CalculationBlock
End CalculationBlock
```

where:

- CalculationDim defines the dimension that will be used in the calculation statements.
- RestrictedDim is a dimension that will define a limited or restricted calculated DataArea.
- "FilterType" specifies the method to use to search the hierarchy for symbols matching the filter criteria and must be enclosed within double quotation marks. Select one of the following:

Value	Description
ALL	To find only the symbols whose attributes match the filter criteria, with no descendents.
PARENT	Starting from the top of the hierarchy, to find the symbols whose attributes match the filter criteria, including all descendents. This checks only parent symbols.
LEAF	Starting from the bottom of the hierarchy, to find the symbols whose attributes match the filter criteria, including all ancestors. This only checks symbols that are leaf symbols; that is, that have no descendents.
ROOT	Searching only the top of the hierarchy to find root symbols whose attributes match the filter criteria, including all descendents.

Syntax example:

```
CalculationBlock (Accounts; Entitites {"ALL", "ZGPNativeCurrency", "EQ"
"CUSD"}, TENTITIES###)

End CalculationBlock
```

Avg

Use this function in an equation to calculate the average value of a set of numbers.

Syntax:

```
AVG (SymName[Structure], SymName[Structure])
```

where:

- Each SymName is a symbol, a series of symbols, or a range of symbols in the dimension specified in a CalculationBlock statement, whose total value you want to average.
- Structure is a range of symbols.

Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
Aver = AVG (AYR01#99)
```

BaseValue

Use this function in an equation to find the first value for a symbol in a range, in the dimension specified in the By statement. For information on By, see By.

You can use the BaseValue function by itself to copy the first value into all the corresponding cells of another symbol in the calculation dimension. You can also use the BaseValue function to produce a value for use by another function. You can restrict the range of the BaseValue function in any dimension.

The order of calculation depends on the priority of the symbols in the dimension specified in a By statement.

Syntax:

```
BASEVALUE (SymName)
```

where:

- SymName is a symbol in the calculation dimension.

Syntax example:

```
BVDelta = SaleT - BASEVALUE(SaleT)
```

By

Use this function to specify the dimension through which calculations are performed.

Normally, when doing a row calculation, your system calculates the values of the first row symbol in all columns and on all pages. Then it moves to the next row calculation, and proceeds in the same way; however, you may want to change the order of calculation so that your system does row calculations in a different sequence — for example, by column. In this case, your system calculates the values of the row symbol in the first column of all pages, and then moves to the next row calculation. Once your system performs all the row calculations for the first column, it performs the first row calculation for the second column, and so on.

Use the By function to calculate along a dimension other than the default. Although you do not always need to use a By statement, you may find calculations are faster when calculating by a different dimension. In addition, some function statements require a By calculation — Lag and BaseValue, for example.

You can use the By function in several ways:

- to optimize the calculations in a Model. When you specify a dimension with a smaller number of symbols in a By statement, you optimize the processing time of the calculation.
- with the Allocate function to specify the direction of the allocation
- to specify the dimension in which a lag or lead is applied
- in conjunction with the following functions:
  - Allocate
  - BaseValue
  - FValue
  - JRoll
  - Lag, Lead

Since your system uses parent/child relationships, use priority to specify the calculation order of symbols. For functions that rely on symbols in a certain order (for example, Lag and Lead), you can specify priorities in the Longview database.

For example, if you have 12 months rolling up to four quarters, you may want to calculate cash flows between periods. You will first have to calculate the cash flows between months, and then between quarters.

Syntax:

```
BY DimName
```

where:

- DimName is the dimension in which calculations will occur.

Syntax example:

```
BY TIMEPER
```

CalculationBlock

Use this function to define the DataArea to which subsequent calculations must be applied.

A CalculationBlock statement stays in effect until you end the CalculationBlock. A CalculationBlock statement applies only while you run the document in which it appears. After creating the CalculationBlock statement, you can use range functions to further limit the calculation to the DataArea specified in the DataSpec. You can also use the AttributeFilter function within a CalculationBlock to further refine the area affected by the calculations.

For more information, see AttributeFilter (CalculationBlock).

The dimension containing a symbol determines its calculation type.

Note: This function must appear before the statement to which it applies.

Syntax:

```
CALCULATIONBLOCK (CalculationDim[;RestrictedDim SymName [Structure][,SymName
[Structure]]][;RestrictedDim SymName [Structure][,SymName[Structure]]])

//statements for this CalculationBlock
End CalculationBlock
```

where:

- CalculationDim defines the dimension that will be used in the calculation statements.
- RestrictedDim is a dimension that will define a limited or restricted calculated DataArea.
- SymName refers to the symbol name that will define a limited or restricted calculated DataArea.

  Note: To specify a range of symbols to limit the calculation to, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

- [Structure] is the symbol structure that will define a limited or restricted set of symbols.

Syntax example:

```
CalculationBlock (TIMEPER; ACCOUNTS BALSHEET#99; ENTITIES W11110#99; DETAILS
P1000#99)

A31120=0

End CalculationBlock
```

Call

Use this function in Longview Application Framework to run a Subroutine Document from a Model Document.

Use a Subroutine Document to avoid repeating common calculations in multiple Longview documents. You can also create a subroutine when you have to create a complex calculation.

Syntax:

```
CALL "Subroutine";ListPassedSyms;ListConstants;ListReturnSyms
```

where:

- "Subroutine" is a Subroutine Document. It can include a complete or partial folder path in the format Subroutine.Folder. If the document is in the Longview working folder, you do not need to specify the folder. The folder path must be enclosed in double quotation marks.
- ListPassedSyms is a list of input symbol values required by the calculations specified in the Subroutine Document. The number of symbols (both SymNameIn and SymNameOut) and constants in the Call statement must equal the number found in the header line (the first line) of the Subroutine Document. If no symbols or constants are passed between the Subroutine Document and the calling document, the Subroutine Document does not contain a header line. Use semicolons (;) to separate each part of the Call statement, even if some of them are not required. If necessary, use commas (,) to separate multiple parameters within a part. Whether you have any symbols to pass on depends on whether the Subroutine Document needs them. Specify the input symbols only if the Subroutine Document requires the values of the symbols to complete the calculations.
- ListConstants is a list of constant values to pass to the Subroutine Document.
- ListReturnSyms is a list of output symbols or calculated results passed back by the Subroutine Document to the original document.

Note: The names of the symbols and constants you use in the Call statement may differ from the symbol names specified in the header line of the Subroutine Document. However, they must be of the same type and contain the same number of parameters.

Syntax example:

```
CALL "Subl.txt"; CASH1, CASH2; 33; “abc”; CASH3, CASH4
```

Count

Use this Longview Application Framework command to return the number of items in a list. This command can be used with either a list variable or a symbol.

Note: Only those symbols to which the user has access will be included in the return list.

Syntax:

```
COUNT (Variable)
COUNT ("DimName","SymName#")
COUNT (Schedule,"DimName","SymName#")
```

where:

- Variable is the name of a variable.
- "DimName" is the name of a dimension containing the symbols.
- Schedule is the name of a schedule.
- "SymName#" is the name of a symbol and any relevant symbol hierarchy specification.

Syntax Example 1

```
CREATE VARIABLE Count1[] AS String

CREATE VARIABLE CNT as NUM
SET VARIABLE Count1 = "P01ytd | p02ytd"
SET VARIABLE CNT = count ("Count1")
```

Syntax Example 2

```
CREATE VARIABLE List

CREATE VARIABLE listcount
CREATE VARIABLE listcount2
CREATE VARIABLE listcount3

SET VARIABLE List = CreateList("Symbols", "Database" "Accounts",
"TrialBalance###")
SET VARIABLE listcount = Count(List)
```

```
SET VARIABLE listcount2 = Count(Accounts, TrialBalance###)
SET VARIABLE listcount3 = Count(CreateList(Users))
```

Depr

Use this function to calculate depreciation amounts.

Syntax:

```
DEPR ("DB", Source, Rate/Life, "Compute"[, OpeningBalance], [Salvage])
```

where:

- Source is a symbol or value representing the value of the asset being depreciated.
- Rate is either a number expressed as a percentage or number of periods. This must be a number between 0 and 1.
- Compute indicates whether the period (PD) or year-to-date (YTD) depreciation is calculated (only YTD is supported currently). This parameter must be enclosed in double quotation marks.
- OpeningBalance is a value repesenting the opening balance for the depreciation calculation.
- Salvage is a value representing the salvage value of the asset.

Syntax example:

```
DeprVal=DEPR("DB", A1000, 0.97, "YTD", 5000, 10)
```

Exp

Use this function in an equation to raise a base value to an exponential power.

Syntax:

```
EXP (SymName|Structure[,SymName|Structure])
```

where:

- SymName is a symbol name containing numeric values, in the dimension specified in a CalculationBlock statement.
- Structure is a range of symbols. To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
EXP (A1000, 10)
```

FileExists

Use this function to determine whether the specified file exists in the specified directory. A return value of 1 means the file exists in the specified directory; a value of 0 means it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
FileExists("FileName")
```

where:

- FileName is the name of the file, including the extension, to search for. In batch mode, the file name must be relative to the current directory. For Longview Smart Client, the file name must be relative to the working directory. You can also specify a full path to search in an alternate directory.

Syntax example:

```
SymbolSpec = FileExists("Reports.xlsx")

If FileExists("Reports.xlsx")
```

Filter

Use this function to filter the symbols affected by a calculation. A FILTER statement is similar to a WHEN statement, except that the filtering is not confined to the calculation block.

Use this function to:

- specify a DataArea in which to check the filtering condition
- identify condition using operators
- state an operation when the condition is evaluated to TRUE
- allow for filtering condition symbols, since symbols for specified dimensions need not be contained in the calculation block

Syntax:

```
FILTER (DimName SymName; Condition)
//statements for this Filter
END FILTER
```

where:

- DimName is a dimension other than the calculation dimension.
- SymName is the name of the symbol in the dimension that holds the filter key value.
- Condition is an expression that evaluates to a TRUE/FALSE indicator (1/0). For a list of valid conditions, see the Is functions.

Syntax example:

```
FILTER (Entities TempEnt; =="AAA")

...
END FILTER
```

FValue

Use this function in an equation to calculate the future value of a payment, or series of payments, at a given future date when compounded by a given interest rate.

Syntax

```
FVALUE OpeningSymName, AddSymName, Rate, IncreaseSymName, NumPeriods
```

where:

- OpeningSymName is a symbol in the dimension specified in a CalculationBlock statement, representing the original value of the investment at the beginning of the first time period.
- AddSymName is a symbol in the current dimension representing the additional amount invested on a regular basis through the investment period at the beginning of each time period (including the first period). It can be 0 or greater.
- Rate is a symbol in the current dimension representing the interest rate per time period. Type one of the following:

Value	Description
a value less than 1	Your system treats it as a decimal form of the percentage. For example, if you enter 0.05, it assumes 5 percent.
a value greater than or equal to 1	Your system treats it as the intended percentage value. For example, if you enter 5, it assumes 5 percent.

- IncreaseSymName is a symbol in the current dimension representing a regular percentage increase in contribution each subsequent time period. For example, if AddSymName equals $100, with IncreaseSymName at 2 percent, the additional payment equals $100 in the first period, $102 in the second period, $104.04 in the third, and so on.
- NumPeriods is a symbol in the current dimension representing the number of time periods over which the investments will grow.

Note: Make sure you use consistent time periods for IntSymName and NumPeriods. For example, for annual payments on a five year loan bearing a 12 percent interest rate, NumPeriods would be 5 and IntSymName would be 12. For the same loan, if you make monthly payments, NumPeriods would be 60 (5 times 12) and IntSymName would be 1 (12 percent divided by 12).

Syntax example:

```
CALCULATIONBLOCK(ACCOUNTS)
BY TIMEPER
CASH1 = FValue(1000,10,0.0025,10,CASH2)
```

```
END CALCULATIONBLOCK
```

GetDate

Use this function to store the current date, in calendar format, in a symbol. You can use Julian date functions to convert the date to a serial number that can be used in calculations. You may find this function useful for keeping track of System Events.

Syntax:

```
GETDATE(["Format"])
```

where:

- Format is the formula for another method of returning a date, such as “dd/mm/yyyy”. This enables the user to have date information returned in the format appropriate to their region. If this parameter is not specified, the default “yyyy-mm-dd” format is followed.

Syntax example:

```
CurrDate=GETDATE()

CurrDate=GETDATE("yy/mm/dd")
```

GetLockComment

Use this function to retrieve the comment associated with a particular lock.

A lock is a Longview database security feature preventing access to data currently used by another user.

Syntax:

```
GETLOCKCOMMENT("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file.

Syntax example:

```
CREATE VARIABLE TXT as string
SET VARIABLE TXT = GETLOCKCOMMENT("Tax_Data.lvdsp")
```

GetLockUser

Use this function to retrieve the name of the user who created a particular lock.

A lock is a Longview database security feature preventing access to data currently used by another user.

Syntax:

```
GETLOCKUSER("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file.

Syntax example:

```
CREATE VARIABLE TXT as string

SET VARIABLE TXT = GETLOCKUSER("Tax_Data.lvdsp")
```

GetStatus

You can use this function to obtain the status of various server features, such as eliminations, journal entries, schedules, and so on.

These features are specified as either TRUE (activated) or FALSE (deactivated) in the Server configuration file (lvsrvr.cfg). When the server is started, it consults this file and turns these features on or off accordingly.

This function is useful when you want to perform an action in Longview but do not know whether you have access to that particular functionality.

When you connect to the Longview database, the status of these features is automatically stored. When the GetStatus function is called with a feature name, the status of the feature is returned to a variable. The status returned can be one of two possible values:

Value	Description
0	FALSE. The feature is deactivated on the server.
1	TRUE. The feature is activated on the server.

When you disconnect, the status of the features is cleared, until the next time you connect to the database.

Syntax:

```
GETSTATUS ("ServerOptionName")
```

where:

- ServerOptionName can be one of the following, enclosed in double quotation marks:

ServerOptionName value	Corresponding configuration parameter name in the lvsrvr.cfg
JE	USE_JOURNAL_ENTRIES
ELIM	DO_ELIMINATIONS
MODELRULES	USE_MODEL
QUERYRULES	USE_QUERY_RULES
ROLLUPRULES	USE_ROLLUP_RULES
NDD	ACCOUNTING_NDD
PAC	ACCOUNTING_PAC
REC	ACCOUNTING_REC
YTD	ACCOUNTING_YTD
TRN	ACCOUNTING_TRN

Syntax:

```
GETSTATUS ("MODELRULES")
```

GetTime

Use this function to get the current time.

You may find this function useful for keeping track of System Events. By default, the time appears in the format hh:mm:ss, if no other format is specified.

Syntax:

```
GETTIME (["Format"])
```

where:

- Format is the formula for another method of returning a timestamp, such as “mm:ss:hh”. If this parameter is not specified, time appears in the default “hh:mm:ss” format.

Syntax:

```
GETTIME ()

GETTIME ("mm:hh:ss")
```

GroupExists

Use this function to determine whether the specified group exists in the database. A return value of 1 means the group exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
GROUPEXISTS("Group")
```

where:

- Group is the name of the group to search for.

Syntax example:

```
GroupExists("Administrators")
```

Grow

Use this function to grow a series of values by a fixed amount. Each successive value is calculated as follows:

Value1	Value2	Value3	Value4
Open	Open _ addition	open + 2 * addition	open + 3 * addition

This function must be used with the BY function. The BY function determines the dimension in which the successive values are calculated.

Syntax:

```
GROW(OpenValue Addition [, Periods])
```

where:

- OpenValue is the opening value, which will be placed in the first symbol in the range.
- Addition is the amount to add to each successive value. This parameter can be a negative amount.
- Periods is optional and is the number of values calculated. If this parameter is not specified, values are calculated for the entire range of the BY dimension.

Syntax example:

```
CdnSls = GROW(100, 20, 4)
```

See also

- By

Growth

Use this function to calculate the growth rate percentage of a series of values.

For each symbol in the BY dimension, growth is calculated as:

```
[(symbolvalue1 - symbolvalue2) / symbolvalue1(-1)] "*" 100
```

Consider the following two examples of symbol values and the resulting calculated growth rate percentages:

SymbolValue1	SymbolValue2	Growth Rate Percentage
100	95	-5
95	112	17.8947

This function must be used in conjunction with the BY function, typically using the time periods dimension.

Syntax

```
GROWTH (SymName)
```

where:

- SymName is the name of the source symbol in the calculation dimension.

Syntax example:

```
GROWTH (A41000)
```

See also

- Using conditional operators
- By
- Operators (relational)
- When...

If

Use this function to carry out a set of instructions if a condition is TRUE.

Do not confuse the If function with the When function:

Function	Description
If	To carry out a set of instructions if any value of a symbol matches a condition.
When	To carry out a set of instructions each time the contents of a cell match a condition.

Syntax:

```
IF Condition
Instructions
[ELSE
Instructions2]
END IF
```

where:

- Condition is a test to determine if a relationship is true or false. Your system applies the If condition to all data cells, and is considered TRUE if the condition is TRUE for any single value. (You can also compare a constant against another constant.) If Condition is TRUE, your system carries out Instructions.
- Instructions is a set of instructions that your system carries out if Condition is TRUE.

Syntax example:

```
IF VALUE (WDGTMFG, A20000, AYR01) LT 0.2 "*" VALUE(WDGTMFG, A99999, AYR01)
    Div = 0.1 "*" A99999
END IF
```

IRR

Use this function to calculate the internal rate of return (the rate that sets the NPV of cash flows to 0) or a series of cash flows. For each period, net inflow = inflow - outflow (outflow is expected to be a positive value). IRR must be used in conjunction with the BY function. The symbols in the BY dimension hold the series of values. The result of IRR is stored in the first symbol in the range of symbols in the BY dimension as specified in the calculation block.

Note: To specify the range of symbols in the BY dimension, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax:

```
IRR (InflowSym [, OutflowSym])
```

where:

- InflowSym is the symbol that contains the cash flows (positive and negative).
- OutflowSym is an optional symbol that contains expenditures.

Syntax:

```
CalculationBlock(Accounts;TimePeriods A0101:A0112)
BY TimePeriods
target = IRR (ChiBalIn, ChiBalOut)
End CalculationBlock
```

Note: If there is no solution to the equation, a value of 0 is returned.

See also

- By

IsAreaBatchStatus

Use this function to retrieve whether the specified status for batches that impact data in a specified data area is TRUE or FALSE. If the specified status for batches that impact data in a specified data area is TRUE, this function returns a value of 1. If the specified status for batches that impact data in a specified data area is FALSE, this function returns a value of 0.

Syntax:

```
ISAREABATCHSTATUS("dataspec.lvdsp",status)
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.
- status is either error or in progress.

IsAreaEventStatus

Use this function to retrieve whether the specified status for events that were triggered by data in a specified data area is TRUE or FALSE. If the specified status for events that were triggered by data in a specified data area is TRUE, this function returns a value of 1. If the specified status for events that were triggered by data in a specified data area is FALSE, this function returns a value of 0.

Syntax:

```
ISAREAEVENTSTATUS("dataspec.lvdsp",status)
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.
- status is either error or inprogress.

IsChanged

Use this function to determine whether the data area has been modified since the initial download. This function can be useful if you want to prompt users to save their changes before closing a Data Grid.

A return value of 0 indicates that the data area is unchanged, while a value of 1 indicates a modification.

Syntax:

```
ISCHANGED(DataArea)
```

where:

- DataArea is the name of the data area.

IsConnected

Use this function to determine if there is a connection to the Longview database. A value of 1 represents a connection; a value of 0 represents no connection.

Syntax:

```
ISCONNECTED()
```

IsKeyword

Use this function to determine if the specified word is already defined as a command, function, or document type. Enclose Text in double quotation marks. A value of 1 means the keyword is already in use in the system; a value of 0 means it is not.

Syntax:

```
ISKEYWORD("Text")
```

where:

- Text is a string representing the keyword, enclosed in double quotation marks.

IsLocked

Use this function to determine if there is a lock in the database covering all or part of the area contained in the specified DataSpec. A value of 1 means such a lock exists; a value of 0 means no such lock exists.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
ISLOCKED ("dataspec.lvdsp")
```

where:

- dataspec.lvdsp is the name of the DataSpec file, enclosed in double quotation marks.

IsMaintenanceOn

Use this function to determine if maintenance is on in the system.

Syntax:

```
ISMAINTENANCEON()
```

If maintenance is on, this function returns a 1. If maintenance is off, it returns a 0.

Syntax example:

```
Set VARIABLE Flag=ISMAINTENANCEON()
```

See also

- Set Variable
- GetMaintenanceUser

IsMaintenanceUser

Use this function to determine whether the current user has authorization to turn maintenance on.

Syntax:

```
ISMAINTENANCEUSER()
```

IsName

Use this function to determine if the text string is a valid name of the specified type. The supported types are USER and OTHERS, though the specification of these types is optional. A value of 1 is returned if the name is valid for the specified type; a value of 0 means the name is not valid.

Syntax:

```
ISNAME("Text"[,"USER"|"OTHERS"])
```

where:

- Text is the name whose validity is to be tested, enclosed in double quotation marks.
- USER sets the specified type to user.
- OTHERS checks non-user types.

IsParentSymbol

Use this function to determine whether the specified database symbol is a parent. A value of 1 means the symbol is a parent symbol; a value of 0 means that it is not (therefore, the symbol is a leaf). If the user does not have access to that particular symbol, the function will return an error message.

Syntax:

```
ISPARENTSYMBOL("dimName", "symName")
```

where:

- dimName is the dimension in which the symbol exists, enclosed in double quotation marks.
- symName is the name of the symbol whose status is to be determined, enclosed in double quotation marks.

IsReadOnly (DataArea)

Use this function to determine whether any part of a data area is read only for the current user.

Returns 1 if any part of the area specified is read only for the current user; 0 if the symbol is not read-only for the current user.

Syntax:

```
ISREADONLY ("SymName0[Structure]","SymName1[Structure]"..."SymNameN
[Structure]")
```

where:

- SymName is the name of the symbol whose status is to be determined, enclosed in double quotation marks (including Structure, if specified).
- Structure is optional and is a range of symbols. To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612

IsReadOnly (Symbol)

Use this function to determine whether a symbol is read-only for the current user.

Returns 1 if SymName is read-only for the current user; 0 if the symbol is not read-only for the current user.

Syntax:

```
ISREADONLY("dimName", "symName")
```

where:

- dimName is the dimension in which the symbol exists.
- symName is the name of the symbol whose status is to be determined.

JDate

Use this function to find the calendar date equivalent of a date in Julian date format.

Julian date format consists of a five digit number; calendar dates appear in the format yyyy-MM-dd hh:mm:ss. (Note that uppercase “M” is used to designate months, and lowercase “m” is used to designate minutes).

Note: The valid range for the JDate function is between 25567 and 402133 (which correspond to 1970 - 3000). Entering an invalid date will cause the following error message to appear: "Invalid JDATE range. JDATE must be between 25567 and 402133."

Do not confuse the JDate function with the JValue function:

Function	Description
JDate	To find the calendar date equivalent of a Julian date.
JValue	To find the Julian date equivalent of a calendar date.

Syntax:

```
JDATE (JulianDate[,"Format"])
```

where:

- JulianDate is the five digit number (between 25567 and 402133) representing a date in the Julian calendar to be returned as a calendar date.
- Format is the format of the calendar date to be returned. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JDATE (28226, "dd/MM/yyyy")
```

JDiff

Use this function in an equation to calculate the number of days between two specified calendar dates.

Syntax:

```
JDIFF (SymName|"Date1", SymName|"Date2"[,"Format"])
```

where:

- SymName is a TIMEPER symbol representing either a start or end date for the range.
- Date1 is the start date of the range.
- Date2 is the end date of the range.
- Format is optional and is the format of the dates to be compared. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JDIFF ("2007-05-06", "2007-12-31")
```

JRoll

Use this function, in conjunction with the By function, to calculate a regular series of calendar dates rolled from a start date in a regular frequency.

Syntax:

```
JROLL (SymName|"StartDate", NumDays, SkipWeekends[,"Format"])
```

where:

- SymName is TIMEPER symbol from which the range is to be calculated.
- StartDate is the first date in the range from which all other dates are counted forward. The format is “MM/dd/yyyy”.
- NumDays is the number of days between dates in the range to be generated.
- SkipWeekends designates that range dates falling on days of the weekend should be included in or excluded from the results and can be one of the following:

Value	Description
0	Designates that range dates falling on days of the weekend should be included in the results.
1	Designates that range dates falling on days of the weekend should be excluded from the results.

- Format is optional and is the format of the date to be returned. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JROLL ("2010-04-05", 5, SkipWeekend)
```

See also

- By

JValue

Use this function in an equation to find the equivalent of a calendar date in Julian date format.

Julian date format consists of a five digit number; calendar dates appear in the format yyyy-MM-dd hh:mm:ss. (Note that uppercase “M” is used to designate months, and lowercase “m” is used to designate minutes).

Note: The valid range for the JDate function is between 25567 and 402133 (which correspond to 1970 - 3000). Entering an invalid date will cause the following error message to appear: "Invalid JDATE range. JDATE must be between 25567 and 402133."

Do not confuse the JValue function with the JDate function:

Function	Description
JValue	To find the Julian date equivalent of a calendar date.
JDate	To find the calendar date equivalent of a Julian date.

Syntax example:

```
JVALUE (SymName|"Date"[, "Format"])
```

where:

- SymName is TIMEPER symbol from which the Julian date is to be calculated.
- Date is the calendar date to be returned as a Julian date.
- Format is optional and is the format of the calendar date. The default is “yyyy-MM-dd hh:mm:ss”.

Syntax example:

```
JVALUE (31/07/2000, "dd/MM/yyyy")
```

Lag, Lead

Use these functions in an equation to point to different cells of a symbol based on a shift (forwards or backwards) in the dimension specified in the By statement.

You may find these functions useful when calculating cash flows.

Unlike most other functions, the Lag and Lead functions do not use the actual function word within syntax. Instead, Lag and Lead are denoted by negative values (with a minus sign) or positive values (no sign), respectively, enclosed in parentheses.

The order in which the values are selected depends on their priority:

Value	Description
Positive shift values	Use to point to cells of a symbol appearing to the right (in the lead) of the current calculation cell in the dimension specified in a By statement. Values to the right have a higher priority number.
Negative shift values	Use to point to cells of a symbol appearing to the left (lagging behind) of the current calculation cell in the dimension specified in a By statement. Values to the left have a lower priority number.

Use to point to cells of a symbol appearing to the left (lagging behind) of the current calculation cell in the dimension specified in a By statement. Values to the left have a lower priority number.

The Lag and Lead functions are determined by the By and CalculationBlock functions. For example, if you use a CalculationBlock Rows statement followed with a By Columns statement, your system applies the Lag and Lead in the columns dimension.

For Lag statements to work properly, specify the priorities of the symbols. For example, if you have 12 months rolling up into four quarters and one year with the following priorities:

Symbol			Time period	Priority
Yr			Total Year	1
	Q1		First Quarter	1
		P1	January	1
		P2	February	2
		P3	March	3
	Q2		Second Quarter	2
		P4	April	4
		P5	May	5
		P6	June	6
	Q3		Third Quarter	3
		P7	July	7
		P8	August	8
		P9	September	9
	Q4		Fourth Quarter	4
		P10	October	10

Symbol	Time period	Priority
P11	November	11
P12	December	12

When priorities are specified this way:

- Yr ### will refer to P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, and P12.
- Yr ##1 will refer to Q1, Q2, Q3, and Q4.
- Yr #99 will refer to P1, P2, P3, Q1, P4, P5, P6, Q2, P7, P8, P9, Q3, P10, P11, P12, Q4, and Yr. This would give you unexpected results for a Lag or Lead calculation.

Note: The Lag and Lead functions can be used successfully only if symbols specified in the CalculationBlock are on the same level relative to their parent. In the examples above, Yr### and Yr##1 are valid but Yr#99 is not.

You cannot use Lag and Lead inside other functions such as Sum, Avg, and so on — they are meant to be used in equations only.

Syntax:

```
TargetSymName = SymName (Shift)
```

where:

- TargetSymName is a symbol that will contain the result of the calculation.
- SymName is a symbol in the dimension specified in a CalculationBlock statement.
- Shift is a number of cells away from the current calculation cell in the dimension specified in a By statement. Use Shift to select the value to be used in calculations.

If you use a Lag of (1), the first symbol in the By dimension is not calculated — it becomes the Lag reference for the second symbol. Similarly, if you use a Lead of (3), the last three symbols in the By dimension are not calculated — they become the Lead reference for other symbols.

Syntax example:

```
CHAR = A11200(-1) - A11200
```

See also

- By
- CalculationBlock

Log

Use this function in an equation to calculate the logarithm of a symbol or value.

Syntax:

```
LOG (SymName[Structure][, Base])

or

LOG (Num[, Base])
```

where:

- SymName is a symbol in the dimension specified in a CalculationBlock statement.
- Structure is a range of symbols.

  Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

- Num is a numeric value.
- Base is a symbol or a constant. The default is the natural logarithm base e (= 2.71828). If Base is a symbol, your system determines the logarithm for each cell in SymName from the corresponding cell in Base.

Syntax example:

```
LogValue = LOG (ExpValue,10)
```

LookUp (Average)

This function returns the average of all the ReturnValues of the lines in an ASCII file where the LookUpValue equals the SourceSymbol value.

Syntax:

```
LOOKUP ("Average","Table",SourceSymName)
```

where:

- Average is the name of the lookup method.
- Table is the name of an ASCII file containing the table of values to be referenced by the LookUp function.
- SourceSymName is a symbol in the dimension specifying the criteria for selection.

If the criteria are not found, the cell is not changed. To avoid any confusion, specify the values in the target symbol as zero before running a Model Document.

Table Format

```
LookUpValue @ ReturnValue
All LookUp values must be numeric.
```

Example Table

```
10 @ 110
11 @ 200.50
222 @ FOUND1
12 @ 130
13 @ 140
11 @ 333.01
15 @ 150
16 @ 50.25
11 @ 100.25
222 @ FOUND2
```

Syntax example:

```
SALES2=LOOKUP("Average",LookupTable.txt,CASH1)
```

If the LookUpTable file contains the entries shown above and CASH1 contains 11, the value returned is average of 200.50, 333.01 and 100.25 and this is assigned to SALES2.

(If the ReturnValue for a line that matches is a string, it is treated as numeric 0).

LookUp (Match)

This function returns the ReturnValue in the lookup table whose LookUpValue equals to SourceSymbol value.

Syntax:

```
LOOKUP ("Match","Table",SourceSymName)
```

where:

- Match is the name of the lookup method.
- Table is the name of an ASCII file containing the table of values to be referenced by the LookUp function.
- SourceSymName is a symbol in the dimension specifying the criteria for selection.

Table Format

```
LookUpValue @ ReturnValue


All LookUp values must be numeric.
```

Example Table

```
10 @ 110
11 @ 200.50
222 @ FOUND1
12 @ 130
13 @ 140
11 @ 333.01
15 @ 150
16 @ 50.25
11 @ 100.25
222 @ FOUND2
```

Syntax example:

```
SALES1=LOOKUP("Match",LookupTable.txt,SALES2)
```

If the LookupTable file contains the entries shown above and SALES2 contains 222, the value returned is "FOUND1" and this is assigned to SALES1. Match finds the first entry in the table that matches.

LookUp (Range)

Use this function to compare a value with a range of values in a table, and to return a character string.

Syntax:

```
LOOKUP ("Range","Table", SourceSymName[,CompareSymName])
```

where:

- Range is the name of the lookup method.
- Table is the name of an ASCII file containing the table of values to be referenced by the LookUp function.
- SourceSymName is a symbol in the dimension specifying the criteria for selection.
- CompareSymName is a symbol containing a numeric data value.

Table Format

```
Without the CompareSymName:
 LowerRange, UpperRange @ ReturnValue
Example (table):
3, 4 @ LINE1
10,15 @ LINE2
110,230 @ LINE3
With the CompareSymName:
CompareSymName, LowerRange, UpperRange @ ReturnValue
Example (table):
9, 3, 4 @ LINE1
10, 10,15 @ LINE2
11, 110,230 @ LINE3
```

Note: If the compare value is not found, it will return "NOT FOUND".

Syntax Example (without CompareSymbol)

```
SALES4=LOOKUP("Range",LookupTable.txt,CASH1)
```

If the RangeLookupTable contains the entries shown above and CASH1 contains 12, the value of LINE2 is returned (lines 2, 3, and 6 all match the range but the one with the upper range closest to 12 is line 2).

Syntax Example (with CompareSymbol)

```
SALES5=LOOKUP("Range",LookupTable.txt,SALES2,CASH1)
```

If the RangeLookupTable2 contains the entries shown above and SALES2 contains 224 and CASH1 contains 11, these four lines match the criteria:

```
11,110,230 @ LINE3
11,177,210 @ LINE4
11,200,225 @ LINE6
11,222,229 @ LINE8
```

Since the source symbol is 224, only three are within this range and LINE6 contains the upper range that is closet to the source symbol (225). The function returns LINE6 to SALES5.

LookUp (Total)

This function is similar to average except it returns the total value of all the ReturnValues of the lines where the LookUpValue is equal to the SourceSymbol value.

Syntax:

```
LOOKUP ("Total","Table",SourceSymName)
```

where:

- Total is the name of the lookup method.
- Table is the name of an ASCII file containing the table of values to be referenced by the LookUp function.
- SourceSymName is a symbol in the dimension specifying the criteria for selection.

If the criteria is not found, the cell is not changed. To avoid any confusion, specify the values in the target symbol as zero before running a Model Document.

Table Format

```
    LookUpValue @ ReturnValue



    All LookUp values must be numeric.
```

Example Table

```
10 @ 110
11 @ 200.50
12 @ 130
13 @ 140
11 @ 333.01
15 @ 150
16 @ 50.25
11 @ 100.25
```

Syntax example:

```
SALES3=LOOKUP("Total",LookupTable.txt,CASH1)
```

If the LookupTable file contains the entries shown above and CASH1 contains 11, the value returned is sum of 200.50, 333.01 and 100.25 and this is assigned to SALES3.

(If the ReturnValue for a line that matches is a string, it is treated as numeric 0).

Max

Use this function to calculate the largest value in a series.

Use Max to calculate the largest value between multiple symbols. The maximum is determined for each symbol within the area defined by the calculation block. (The user must have access to the symbols in an area in order to use this function in that particular area).

Use MaxArea to calculate the single largest value in an area. The area is not restricted by the calculation block.

Syntax

```
Max (Value|SymName[Structure][,Value|SymName[Structure]]) (in Models)

Max (Value[, Value])

MaxArea ("SymName0[Structure]", "SymName1[Structure]", ..., "SymNameN
[Structure]") (in Models)
```

where:

- Value is a number.
- SymName is a symbol name.
- Structure is a range of symbols.

Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
MAX (A2100#99)
```

Min

Use this function to calculate the smallest value in a series.

Use Min to calculate the smallest value between multiple symbols. The minimum is determined for each symbol within the area defined by the calculation block. (The user must have access to the symbols in an area in order to use this function in that particular area).

Use MinArea to calculate the single smallest value in an area. The area is not restricted by the calculation block.

Syntax:

```
Min (Value|SymName[Structure][,Value|SymName[Structure]]) (in Models)

Min (Value[, Value])

MinArea ("SymName0[Structure]", "SymName1[Structure]", ..., "SymNameN
[Structure]") (in Models)
```

where:

- Value is a number.
- SymName is a symbol name.
- Structure is a range of symbols.

Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
MIN (A2100#99)
```

Neg

Use this function in an equation to find symbols with negative values. If the symbol has a positive value, the Neg function returns zero.

Syntax:

```
NEG (SymName|Number)
```

where:

- SymName is a symbol whose negative values you want to specify.
- Number is a value.

Syntax example:

```
Account1 = NEG (Account2)
```

See also

- Pos

NPV

Use this function to calculate the net present value of future cash flows. For each period, net inflow equals inflow minus outflow (outflow is expected to be a positive value). NPV must be used with BY dimension. The symbols in the BY dimension hold the series of values. The result of NPV is stored in the first symbol in the range of symbols in the BY dimension as specified in the calculation block.

Note: To specify the range of symbols in the BY dimension, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax:

```
NPV (InflowSymName,Rate[, OutflowSymName])
```

where:

- InflowSymName is a symbol that contains the cash flows (+ and -).
- Rate is the interest rate to discount the cash flows at.
- OutflowSymName is an optional symbol that contains expenditures.

Syntax example:

```
CalculationBlock(Accounts;TimePeriods A0101:A0112)
BY TimePeriods
target = NPV (Chi1,0.12,Chi2
End CalculationBlock
```

NumToStr

Use this function to convert a numeric value to a character string.

You may find this function useful for comparison purposes. For example, suppose a complete account number in your system appears in the format A11234. You can use the NumToStr function to change any numeric values to character strings, so that they can be compared correctly.

The value provided by your system depends on the format of the source item being converted:

Format of Source	Value of TargetSymName	Example
Character string	Returns the source string	
Numeric and an integer	Character string representing the number	If Source is 1234, TargetSymName becomes the character string "1234".
Numeric but not an integer	Number	No specific number of decimals. No zero-padding.

Syntax example:

```
NUMTOSTR (SymName|Num)
```

where:

- SymName is a symbol name.
- Num is a value.

Syntax example:

```
NumToStr (1234)
```

Operators (arithmetic)

Use arithmetic operators to designate the operational relationships between expressions in an equation. For example, to add two numbers, use the "+" operator (as in "5+5").

Unlike most other functions, arithmetic operators do not use an actual function word within syntax — they use arithmetic symbols instead.

If an equation contains more than one arithmetic operator, your system evaluates the expressions in the following order of precedence:

1. functions and expressions in parentheses
2. multiplication and division (* , /)

   Caution: The use of the asterisk as a multiplication symbol in the application requires that it be enclosed in double quotation marks ("*"). An asterisk without double quotation marks is interpreted by the system as a wildcard search character, and operations including it will not give the expected results.

3. addition and subtraction (+ , -)

If the arithmetic operations have equal precedence, your system calculates them from left to right.

Syntax:

```
Expression1 OPERATOR Expression2
```

where:

- Each Expression is a mathematical expression consisting of symbols, constants, values, and functions. You can use parentheses to nest one expression within another.
- OPERATOR is an arithmetic operator. Although extra spacing is not required, you should make the expression easier to read and edit by typing a space before and after each operator:

Value	Description
+	Addition
-	Subtraction
/	Division
“*”	Multiplication

Syntax example:

```
Div = 0.1 "*" NetInc
```

Error messages caused by arithmetic calculations

If you use an invalid arithmetic operation in a statement, one of several error messages may appear:

Error	Description
#OVERFLOW	Result of a value greater than the maximum value the CPU can handle.
#UNDERFLOW	Result of a value less than the minimum value the CPU can handle.
#INFINITE	Result of an out-of-value range number, underflow, or overflow divided by 0.
-0	Result of 0 divided by 0.

These messages may appear as a result of unusual data or erroneous logic. For possible solutions to logic errors (for example, to limit calculations when the divisor would be equal to zero), see IRR and When....

Operators (relational)

Use relational operators to compare two mathematical expressions.

To display data under certain conditions in a Model or Procedure Document, use relation operator functions to specify a condition that must be true. You can use parentheses to change the order in which expressions are compared.

Syntax:

```
Expression1 OPERATOR Expression2
```

where:

- Each Expression is a mathematical combination of symbols, constants, and functions. If you use the EQ or NE functions, Expression can be a character string enclosed in double quotation marks.
- OPERATOR is a relational operator. Select one of the following:

Value	Description
EQ or ==	Equal to.
GE or >=	Greater than or equal to.
GT or >	Greater than.
LE or <=	Less than or equal to.
LT or <	Less than.
NE or !=	Not equal to.

Syntax example:

```
IF NetInc GT 0 AND Debt LT 0.2 "*" NetInc
```

Operators (conditional)

Use conditional operator functions to link two conditional expressions.

To instruct Longview to carry out certain instructions, use a conditional operator:

Conditional Operator	Description
And	When each of several conditions must be true.
Or	When one of several conditions must be true.

You can use parentheses to change or change the order in which conditions are evaluated; for example:

```
IF (Cash1 GT 0 OR Cash2 LT 100) AND (Cash3 LT 0 OR Cash4 NE 100)
```

Use conditional operators in Longview documents as follows:

Longview documents	Model Procedure
Must be used with...	If When While
Positioning	Can be used with If statement in all documents. Can be used with When statement in Model Documents.

Syntax:

```
Condition1 OPERATOR Condition2
```

where:

- Each Condition is a relational expression containing the EQ, GE, GT, NE, LT, or LE functions, or another expression containing the And or Or functions. For information on relational operators, see Operators (relational).
- OPERATOR is a conditional operator. Select one of the following:

Value	Description
And	To find an expression that matches both conditions.
Or	When one of several conditions must be true.

Syntax example:

```
IF NetInc GT 0 AND Debt LT 0.2 "*" NetInc
```

Or

See Operators (arithmetic).

PGrow

Use this function to calculate successive values as a percentage, growing the values from one instance to the next.

Each successive value is calculated as follows:

V1	V2	V3
Open	Open*(1 + rate)	Open*(1 + rate)*(1 + rate)

This function must be used with the BY function. The BY function determines the dimension in which the successive values are calculated.

Syntax:

```
PGROW(OpenValue,Rate[, Periods])
```

where:

- OpenValue is the opening value, which will be placed in the first symbol in the range.
- Rate is the growth rate used, which can be negative.
- Periods is optional and is the number of values calculated, if not specified, values are calculated for the entire range of the BY dimension.

Syntax example:

```
CdnSls = PGROW(CdnSls, 20, 4)
```

See also

- By

Pos

Use this function in an equation to find symbols with positive values. If a cell contains a negative value, the Pos function assigns a value of zero.

Syntax:

```
POS (SymName|Number)
```

where:

- SymName is a symbol whose positive values you want to specify.
- Number is a value.

Syntax example:

```
Account1 = POS (Account2)
```

See also

- Neg

Rollup

Use this function to perform a rollup on specific parent symbols or entire dimensions, according to changes made to child symbols. ROLLUP rolls up all specified symbols, unlike the ROLLUPACTIVE function which rolls up changes only. ROLLUP is limited to the area defined by a calculation block.

Note: A ROLLUP function cannot be contained within a ROLLUPACTIVE block.

Syntax:

```
ROLLUP ("FULL"|SymName)
```

where:

- "FULL" specifies to roll up every symbol in the specified dimension, up to the root symbols. This process can take some time if used on a large DataArea.
- SymName is the name of a parent symbol in the specified dimension.

Syntax example:

```
ROLLUP ("FULL")

ROLLUP (A1000)
```

RollupActive

Use this function to put the current calculation block into a state where changes to values are tracked for later rollup processing. The rollup is limited to the area defined by the calculation block and must be turned off before the end of the calculation block. ROLLUPACTIVE only rolls up the changes, unlike the ROLLUP function.

Note: A ROLLUP function cannot be contained within a ROLLUPACTIVE block.

Syntax:

```
ROLLUPACTIVE ("ON" | "OFF")
```

where:

- ROLLUPACTIVE provides two options. Select one of the following:

Value	Description
ON	Starts tracking changed cell values for rollup purposes.
OFF	Performs the rollup based on the changed cell values since the RollupActive was turned on.

Note: "ON" and "OFF" must be enclosed in double quotation marks.

Syntax example:

```
CalculationBlock (Accounts)
RollupActive("ON")
A1 = $UserInput$
A2 = A1 "*" .25
RollupActive("OFF")
A4 = A3 "*" .75
EndCalculationBlock()
```

This would calculate A1, A2 and then propagate the changes to A1, A2 to all affected parents in the DataArea, then calculate A4.

Round

Use this function in an equation to round values to the nearest decimal place.

Syntax:

```
ROUND (SymName[,NumPlaces])

ROUND (Value[,NumPlaces])
```

where:

- SymName is a symbol.
- Value is a number, or a valid mathematical expression.
- NumPlaces is a positive integer, or a symbol whose values are all positive integers, representing the number of decimal places to which each value must be rounded. The default is to round to the nearest whole number.

Syntax example:

```
ROUND (A41000 / 61100,2)
```

RuleExists

Use this function to determine whether the specified rule ID exists in the database. A return value of 1 means the rule ID exists; a value of 0 means it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
RULEEXISTS (RuleID)
```

where:

- RuleID is the number of the rule ID to search for.

Syntax example:

```
RuleExists("100")
```

ScheduleExists

Use this function to determine whether the specified schedule exists in the database. A return value of 1 means the schedule exists; a value of 0 means it does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
SCHEDULEEXISTS("SchedName")
```

where:

- SchedName is the name of the schedule to search for.

Syntax example:

```
SCHEDULEEXISTS("ICStandard")
```

Series

Use this function to populate a series of values into a range of cells. Must be used in conjunction with the BY function.

Syntax:

```
SERIES ([Num1 of] Value1 [, [Num2 of] Value2]...[, NumX of] ValueX])
```

where:

- Value specifies the number of times, starting at the first symbol and continuing until either the limit of the BY dimension range is reached or there are no more values to insert, to repeat to create the range. Used alone, Value may be any floating point or integer value.
- Num must be a positive integer, used with a Value that is a floating point number.

Syntax example:

```
Symbol = Series (2 of 1, 3 of 2)
```

See also

- By

Simult

Use this function in a calculation block to specify the convergence ratio and number of iterations to be used in simultaneous calculations. A simultaneous equation is a set of interdependent calculations in which no single calculation can be solved unless the others are solved simultaneously.

Syntax:

```
Simult(Convergence, MaxIterations)
```

where:

- Convergence is the convergence ratio (percentage) that you want to use. Type one of the following:

Value	Description
A value less than 1	The application treats it as a decimal form of the percentage. For example, if you enter 0.05, it assumes 5 percent.
a value greater than or equal to 1	The application treats it as the intended percentage value. For example, if you enter 5, it assumes 5 percent.

MaxIterations is an integer representing the maximum number of iterations for a simultaneous or iterative set of calculations. The default is 20.

Syntax example:

```
SIMULT(1,25)
```

SkipNoData

Use this function in a model document to skip intersections containing no data. This function enhances performance by speeding up the parsing of large data areas containing a small amount of data.

If the Collect Statistics command is run prior to this function, the collected statistics are used to maximize SkipNoData function performance.

Caution:
This function is not safe to use on LEAD/ LAG calculations. This function is also not safe to use when there are calculations where the right-hand side of the equation includes any term (elements separated by plus or minus) that resolve to a constant or string literal.

For example:

a. In the case below, the calculations for both A and C have terms on the right-hand side that are constants. It is not safe to use SkipNoData in this scenario:

   A = 2

   C = B + 10

b. In the case below, the calculation for A has a term on the right-hand side that effectively resolves to a constant (VALUE). It is not safe to use SkipNoData in this scenario:

   A = B + VALUE(…)

c. In the case below, the equation for Flag resolves to a string literal. It is not safe to use SkipNoData in this scenario:

   WHEN Flag EQ "0" OR Flag EQ ""

   Flag = “Initialized”

   END WHEN

Note: The WHEN terms in this example are irrelevant, it is the line assigning a string literal to the Flag symbol that makes this equation unsafe for SkipNoData.

The following calculations all have single terms on the right-hand side, and because they are multiplied by a symbol, these terms do not resolve to a constant and therefore SkipNoData is safe to use:

A = B * VALUE(…)

C = B * 10

Syntax:

```
SKIPNODATA "ON"|"OFF"
```

Select one of the following:

Value	Description
ON	The system ignores data intersections containing no data.
OFF	The system parses and submits all data.

Syntax example:

```
CalculationBlock (...)


Equation1
Equation2
SkipNoData "ON"
Equation3
SkipNoData "OFF"
Equation4
End CalculationBlock
```

See also

- Collect Statistics
- Delete Statistics

StrContains

Use this function to verify if the search string exists within a specific string. A result value of 1 is returned if the search string exists; a value of 0 is returned if the search string does not exist.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
STRCONTAINS (SymName|"String",SymName|"SearchString")
STRCONTAINS ("String","SearchString")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.
- SearchString is a character string, enclosed in double quotation marks, which is searched for inside String.

Syntax example:

```
CREATE VARIABLE NumTemp1 AS NUM

SET VARIABLE NumTemp1 = STRCONTAINS ("StringTest","Test")
NumTemp1 will equal 1
```

StrFind

This function returns a numeric value indicating the starting position within a string of the search string. For example, searching for “ad” within “Nevada” would return a value of 4.

Syntax:

```
STRFIND ("String", "SearchString")
```

where:

- String is a character string, enclosed in double quotation marks.
- SearchString is a character string, enclosed in double quotation marks, which is searched for inside String.

Syntax example:

```
CREATE VARIABLE NumTemp1 AS NUM
Variable = STRFIND ("SourceString","SearchString")
SET VARIABLE NumTemp1 = STRFIND ("StringTest","Test")
NumTemp1 will equal 7
```

StrLeft

Use this function to extract the leftmost character or characters from a symbol or character string. You may find the StrLeft function useful for finding individual values from a combined value. For example, suppose a complete account symbol in your system appears in the format A11234 — consisting of a prefix code (A), followed by two digits (11) representing an entity code, followed by three digits (234) representing an account code.

You can use the StrLeft function to find the prefix code (the leftmost character) from the complete account symbol.

Syntax:

```
STRLEFT (SymName|"String",NumChars)
```

where:

- SymName is the name of a symbol.
- "String" is a character string, enclosed in double quotation marks.
- NumChars is a positive integer representing the number of characters to extract from the beginning of the character string. NumChars cannot be greater than the number of characters in the character string.

Syntax example:

```
CodeLet = STRLEFT("A11234",1)
```

Example:

```
CREATE VARIABLE StrTemp1 AS STRING

SET VARIABLE StrTemp1 = STRLEFT ("StringTest",6)
StrTemp1 will equal "String" (without the double quotation marks)
```

StrLength

Use this function to calculate the length of a character string.

Syntax:

```
STRLENGTH ("String")
```

where:

- String is a character string, enclosed in double quotation marks.

Syntax example:

```
NumChars = STRLENGTH ("StringTest")
```

Example:

```
CREATE VARIABLE NumTemp1 AS NUM
SET VARIABLE NumTemp1 = STRLENGTH ("StringTest")
```

StrLower

Use this function to convert text in a symbol or character string to lowercase letters.

You can use the StrLower function to convert all uppercase letters in symbols to lowercase.

Syntax:

```
STRLOWER (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Syntax example:

```
LowCase = STRLOWER ("A11234")
```

Example:

```
CREATE VARIABLE LowCase AS STRING

LowCase = STRLOWER ("A11234")
A11234 will equal "a11234" (without the double quotation marks)
```

StrReplace

Use this function to replace a part of a character string with a different character string.

You may find the StrReplace function useful for comparison purposes. For example, suppose a complete symbol in your system appears in the format A11234.

You can use the StrReplace function to change all prefix codes from A to B.

Syntax:

```
STRREPLACE (SymName|"String","SearchString","NewString")
```

where:

- SymName is the name of a symbol.
- "String" is a character string enclosed in double quotation marks.
- SearchString is an existing character string, enclosed in double quotation marks. This parameter is case-sensitive and must be typed with the correct combination of uppercase and lowercase letters.
- NewString is a new character string, enclosed in double quotation marks. This parameter is case-sensitive and must be typed with the correct combination of uppercase and lowercase letters.

Syntax:

```
STRREPLACE (CodeLetB,"A","B")
```

Example:

```
CREATE VARIABLE StrTemp1 AS STRING
Variable = STRREPLACE ("SourceString","SearchFor","ReplaceWith")
SET VARIABLE StrTemp1 = STRREPLACE ("StringTest","Test","xxxx")
StrTemp1 will equal "Stringxxxx" (without the double quotation marks)
```

StrRight

Use this function to extract the rightmost character or characters from a symbol or character string. You may find the StrRight function useful for finding individual values from a combined value.

For example, suppose a complete account number in your system appears in the format A11234 — consisting of a prefix code (A), followed by two digits (11) representing an entity code, followed by three digits (234) representing an account code.

You can use the StrRight function to find the account code (the three rightmost characters) from the complete account number.

Syntax:

```
STRRIGHT (SymName[Structure] | "String",NumChars)
```

where:

- SymName is the name of a symbol.
- Structure is a range of symbols.

  Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

- String is a character string enclosed in double quotation marks.
- NumChars is a positive integer representing the number of characters to extract from the end of the character string. NumChars cannot be greater than the number of characters in the character string.

Syntax example:

```
STRRIGHT (AcctCode,"A11234",3)
```

Example:

```
CREATE VARIABLE StrTemp1 AS STRING

Variable = STRRIGHT ("SourceString",Number)
SET VARIABLE StrTemp1 = STRRIGHT ("StringTest",4)
StrTemp1 will equal "Test" (without the double quotation marks)
```

StrToNum

Use this function to convert a character string to a numeric value.

You may find this function useful for comparison purposes. For example, suppose a complete account number in your system appears in the format A11234. You can use the StrToNum function to change any character strings to numeric values, so they can be compared correctly.

The value provided by your system depends on the format of the source item being converted:

Format of Source	Value of TargetSymName	Example
Symbol with numeric value	An error message appears.	String is a valid number.
Empty character string	An error message appears.	Unable to convert string to number.
Character string that cannot change to numeric value	An error message appears.	Unable to convert string to number.

Syntax:

```
STRTONUM (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Syntax example:

```
AcctNum = STRTONUM ("11234")
```

Example:

```
CREATE VARIABLE NumTemp1 AS NUM
Variable = STRTONUM ("SourceString")
SET VARIABLE NumTemp1 = STRTONUM ("1234")
NumTemp1 will equal 1234 as a numeric value, not a string value.
```

StrTrim

Use this function to isolate the text of a string from all leading and trailing spaces.

Syntax:

```
STRTRIM (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Example:

```
CREATE VARIABLE StrTemp1 AS STRING
Variable = STRTRIM ("SourceString")
SET VARIABLE StrTemp1 = STRTRIM (" StringTest ")
StrTemp1 will equal "StringTest" (without the double quotation marks)
```

StrTrimL

Use this function to isolate the text of a string from all leading spaces.

Syntax:

```
STRTRIMR (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Syntax:

```
CREATE VARIABLE StrTemp1 AS STRING
Variable = STRTRIML ("SourceString")
SET VARIABLE StrTemp1 = STRTRIML (" StringTest ")
StrTemp1 will equal "StringTest " (without the double quotation marks)
```

StrTrimR

Use this function to isolate the text of a string from all trailing spaces.

Syntax:

```
STRTRIML (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Example:

```
CREATE VARIABLE StrTemp1 AS STRING
Variable = STRTRIMR ("SourceString")
SET VARIABLE StrTemp1 = STRTRIMR (" StringTest ")
StrTemp1 will equal " StringTest" (without the double quotation marks)
```

StrUpper

Use this function to convert text in a symbol or character string to uppercase letters.

You may find the StrUpper function useful for comparison purposes. For example, suppose a complete account number in your system appears in the format A11234, but with either an uppercase or lowercase A.

You can use the StrUpper function to convert all lowercase letters in symbols to uppercase.

Syntax:

```
STRUPPER (SymName|"String")
```

where:

- SymName is the name of a symbol.
- String is a character string, enclosed in double quotation marks.

Syntax example:

```
UppCase = STRUPPER ("a11234")
```

Example:

```
CREATE VARIABLE StrTemp1 AS STRING

Variable = STRUPPER ("SourceString")
SET VARIABLE StrTemp1 = STRUPPER ("StringTest")
StrTemp1 will equal "STRINGTEST" (without the double quotation marks)
```

StrValue

Use this function to retrieve a string value from a data intersection and store it to another data intersection, or a string variable. This function will only return a string value for a DataArea if the user has access to that DataArea.

Note: This function will only return string data. If the intersection contains a numeric value, then the function will return an empty string ("").

Syntax:

```
STRVALUE (""|"DataAreaName1", "Dim1Set", "Dim2Set", …, "Dim7Set")
```

where:

- "" is used in a model to reference the same data area that the model is executed against.
- DataAreaName is the name of the DataArea in which the symbols are found, and must be used in a procedure.
- DimNSet is the name of a symbol for each dimension in your system.

Note: Symbol names must be enclosed in double quotation marks.

Syntax example:

```
SALEST### = STRVALUE ("DataArea1", "CASHT###", "Dim2Set", …, "Dim7Set")
SALEST### = STRVALUE ("", "CASHT###", "Dim2Set", …, "Dim7Set")
SET VARIABLE myString = STRVALUE ("DataArea1", "CASHT", "Dim2Set", …,
"Dim7Set")
```

SubStr

Use this function to extract part of a string.

Syntax:

```
SUBSTR (SymName|"String", StartChar, NumChars)
```

where:

- SymName is the name of a symbol.
- String is a character string enclosed in double quotation marks.
- StartChar is the first character of the string that you would like to extract.
- NumChars is the number of characters that you would like to extract.

Syntax:

```
CREATE VARIABLE StrTemp1 AS STRING
Variable = SUBSTR ("SourceString",StartNum,Length)
SET VARIABLE StrTemp1 = SUBSTR ("StringTest",2,4)
StrTemp1 will equal "trin" (without the double quotation marks)
```

Sum

Use this function in an equation to calculate total values for a symbol or group of symbols that do not roll up to a specific parent.

In a hierarchy, it is easy to calculate the sum of all child symbols of a parent. However, you may find the Sum function useful to calculate the total of a series of symbols that do not roll up to a specific parent. All symbols must have the same weight (positive, negative, or zero).

Do not confuse the Sum function with the WSum command:

Function	Description
Sum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols have the same weight.
WSum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols do not have the same weight.

Syntax:

```
SUM (SymName1[Structure][,SymName2[Structure]]...)

SUM (Value1[,Value2][,Value3]...)
```

where:

- Each SymName is a symbol from the dimension corresponding to the calculation type currently in effect. You can also use pound signs (#) to specify the number of symbol levels in the hierarchy. Specify symbols containing only numeric values.
- Structure is a range of symbols.

Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Each Value is the name of a NUM list variable, or a numeric value or expression that evaluates to a number.

Syntax example:

```
CstSlsT = SUM (MatCst ###,OvHead ###)
```

SumArea

Use this function to sum an area in a DataArea to a single value. This function must be used within a calculation block. This function can be used in two ways: to assign a value to a calculation block, and to evaluate for multiple slices in a calculation block. This function works in conjunction with the Symbol function.

Syntax:

```
SumArea (SymName,[Structure], SymName)
```

where:

- SymName is a symbol name.
- Structure is a range of symbols.

Note: To specify a range of symbols, enter the first and last symbols in the range, separated by a colon. For example, A0601:A0612.

Syntax example:

```
SumArea(TrialBalance###, Period_01:Period_03, Total_Entities)

SumArea(TrialBalance###, Period_01:Period_03, Symbol())
```

Symbol

Use this function to convert a text value into a symbol reference, which can then be used in a function or formula. This function is utilized when parameters for a "formula" are stored as data in the database.

It is possible to nest arguments within the main symbol call (for example, SYMBOL (), TIMEPERIOD, SYMBOL()...). Symbol() can also be used for the other dimension in the DataArea in functions that require symbols for other dimensions (such as VALUE or STRVALUE).

Syntax:

```
SYMBOL ()
SYMBOL Expression
```

where:

- Expression is a data intersection to be resolved as a text string. If no expression is provided, the current symbol being calculated in the calculation block will be used.

Syntax example:

```
target = A1 + Symbol("A2")
or
Value (A1, Symbol(), Symbol(), Temp)
```

SymbolExists

Use this function to assess whether the specified symbol exists in a specified DataArea. A value of 1 means the symbol exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
SYMBOLEXISTS ("dataAreaName", "dimName", "symName")
```

where:

- dataAreaName is the name of the DataArea that contains the symbol to search for.
- dimName is the name of the dimension that contains the symbol to search for.
- symName is the name of the symbol to search for.

SymbolRange

Use this function to convert a text value into a list of symbols, which can then be used in a function or formula. This function is utilized when parameters for a "formula" are stored as data in the database. This function is used within a calculation block.

Syntax:

```
SymbolRange ()

SymbolRange Expression
```

where:

- Expression is a data intersection to be resolved to a list of symbols. If no expression is provided, the current symbol being calculated in the calculation block will be used.

Syntax example:

```
SymbolRange("Total_L_SE | TOTAL_E")
```

Transfer

This function is used to transfer data from one area to another. This function copies values of an n-dimensional block of cells, defined by the symbols written within the TRANSFER syntax, to an n-dimensional block of cells in the same or in a different DataArea.

When using this function, you must define at least one dimension using the CalculationBlock function. You must also define a symbol from each dimension in the Transfer statement syntax.

This function will transfer only those symbols to which the user has access.

```
A
|_A1 ==> Priority= 1
|_A2 ==> Priority= 2
|_A3 ==> Priority= 3
B
|_B1 ==> Priority= 1
|_B2 ==> Priority= 2
|_B3 ==> Priority= 3
```

Note: The TRANSFER function takes symbol priority into account. For example, in the following hierarchy structure example, data is transferred from symbol A1 to symbol B1 because of the corresponding priority levels.
For more information on symbol priority, see the "*Longview Application Administrator Guide*".

Syntax:

```
TRANSFER SymName0[Structure] | COMMON, SymName1[Structure] | COMMON,
...,SymNameN[Structure] | COMMON TO SymName0[Structure] | COMMON, SymName1
[Structure] | COMMON, ..., SymNameN[Structure] | COMMON FROM "DataAreaName"
```

where:

The first section of the Transfer statement defines the source DataArea. The second section (after the TO) defines the target DataArea.

- SymName0 is a symbol from the first dimension of the DataArea (i.e. ACCOUNTS).
- SymName1 is a symbol from the second dimension of the DataArea (i.e. TIMEPER).
- SymNameN is a symbol from the Nth dimension of the DataArea (all other dimension symbols by dimension order).
- [Structure] is an optional parameter that defines the level of symbols within the symbol parent structure. For more information, see Symbol hierarchies.

- COMMON is an optional parameter to be used instead of defining the dimension symbol and/or Structure. When the dimension DataArea is the same for both source and target, COMMON can be used to define all symbols for that dimension in the DataArea.

Note: COMMON must be used for both the dimension source and target areas.

Syntax using temporary symbols

If the temporary symbols belong to a different DataArea than the one associated with the Model, you must enclose them in double quotation marks, such as:

```
TRANSFER "TempSymGet" COMMON,COMMON,COMMON,COMMON,COMMON,COMMON,COMMON TO
DIM0SET,COMMON,COMMON,COMMON,COMMON,COMMON,COMMON,COMMON FROM "GetData"
```

where:

- TempSymGet is a temporary symbol in the GetData DataArea.

Note: If temporary symbols belong to the current DataArea (associated with the Model) the double quotation marks should not be used.

Caution: Database symbols should not be enclosed in double quotation marks; doing so will result in a syntax error.

Example (data Transferred Within A DataArea)

```
The following is executed in a Procedure Document:

RUN MODEL FORCPY.lvmod ON FORDATA
The following is executed in the FORCPY Model Document:

CalculationBlock (ACCOUNTS)

TRANSFER BALSHEET#99,F0701,COMMON,COMMON,CCAD,Dim5Set TO
BALSHEET#99,F0702,COMMON,COMMON,CCAD,Dim5Set

End CalculationBlock

In this example, the Transfer function moves F0701 to F0702 within the same
DataArea (FORDATA).
```

Example (data Transferred Between DataAreas)

```
The following is executed in a Procedure Document:


RUN MODEL ACTTOFOR.lvmod ON FORDATA

The following is executed in the ACTTOFOR Model Document:


CalculationBlock (ACCOUNTS)


TRANSFER BALSHEET#99,A0701,COMMON,COMMON,CCAD,Dim5Set TO
BALSHEET#99,F0701,COMMON,COMMON,CCAD,Dim5Set FROM "ACTDATA"

End CalculationBlock

In this example, the Transfer function moves A0701 from the ACTData area to
F0701 in the FORDATA area.
```

Example (data Transferred Within A Schedule Dimension)

```
The following is executed in a Procedure Document:


RUN MODEL FORCPY.lvmod ON FORDATA

The associated datalink Document would include the base dimensions and
reference to the schedule and schedule symbols:

SCHEDULE ICSTANDARD

OFFSET Total###


The following is executed in the FORCPY.lvmod file:

CalculationBlock (ACCOUNTS)


TRANSFER BALSHEET#99,F0701,COMMON,COMMON,CCAD,Dim5Set, Total###TO
BALSHEET#99,F0702,COMMON,COMMON,CCAD,Dim5Set, Total###


End CalculationBlock


In this example, the Transfer function moves schedule data from F0701 to
F0702 within the same DataArea (FORDATA).
```

UserExists

Use this function to determine whether the specified user exists in the database. A return value of 1 means the user exists in the system; a value of 0 means the user does not.

Note: If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
USEREXISTS ("UserID")
```

where:

- UserID is the user ID to search for.

Syntax example:

```
USEREXISTS ("jsmith")
```

Value

Use this function in an equation to get the value from an intersection of data, and to assign that value to an intersection of data, or to use it in a calculation.

Note: This function will only return numeric data. If the intersection contains a string value, then the function will return a zero (0).

Syntax:

```
VALUE ("DataAreaName","SymName0","SymName1", ..., "SymNameN")
```

where:

- DataAreaName is the name of the DataArea in which the symbols are found.
- SymName is the name of a symbol.

Note: Symbol names must be enclosed in double quotation marks.

Syntax:

```
VALUE ("daMaster","Acct1", "A1201", "Entity100", …, "WorkingVersion")
```

VariableExists

Use this function to determine whether the specified variable exists in the database. A return value of 1 means the variable exists; a value of 0 means it does not.

> **Note:** If you create a variable to store the return value for this function, you must create the variable with the NUM type.

Syntax:

```
VARIABLEEXISTS("Variable")
```

where:

- Variable is the name of the variable to verify the existence of.

Syntax example:

```
VariableExists("sEmployeeType")
```

When...

Use this function to carry out a set of instructions each time the contents of a cell match a condition.

Do not confuse the When function with the If function:

Function	Description
When	To carry out a set of instructions each time the contents of an intersection in a DataArea match a condition.
If	To carry out a set of instructions if any of the intersections in a DataArea matches a condition. In this case, the IF function carries out the instructions for all the intersections in that DataArea. (The IF function looks for a match of the condition and then applies the instructions to each and every intersection of the DataArea regardless of whether the remaining intersections in the DataArea match the condition.)

Syntax:

```
WHEN Condition
Instructions
[ELSE
Instructions2]
END WHEN
```

where:

- Condition is a test to determine if a relationship is true or false. Your system tests the condition for every value of that symbol. When Condition is true, your system carries out Instructions.
- Instructions is an instruction that your system carries out if Condition is true.
- Instructions2 is an instruction that your system carries out if Condition is false.

Syntax example:

```
WHEN NetInc GE 1000
Invest = 0.2 "*" NetInc
END WHEN
```

WSum

Use this function to calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols do not have the same weight.

With parent/child relationships, it is easy to calculate the sum of all child symbols of a parent. However, you may find the WSum function useful for calculating the total of a series of symbols that do not roll up to a specific parent.

The process of calculating the total of symbols that do not have the same weight is called weighted summation. If the symbol's weight is negative, the symbol is subtracted from the total; if it is positive, the symbol is added to the total; and if it is zero, the symbol does not affect the total.

Do not confuse the WSum function with the Sum command:

Function	Description
WSum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols do not have the same weight.
Sum	To calculate total values for a symbol or group of symbols that do not roll up to a specific parent, when all symbols have the same weight.

Syntax:

```
WSUM (SymName1[Structure], [SymName2, ... SymNameN])
```

where:

- Each SymName is a symbol whose child symbols you want to add. You can also use pound signs (#) to specify the number of symbol levels in the hierarchy.
- Each Structure is the symbol structure that will define a limited or restricted set of symbols.

Syntax example:

```
NetInv = WSUM (InvT ###)
```

Developing Longview ImportSpecs, ExportSpecs, And External Maps

You can use an ImportSpec or ExportSpec document in Longview Application Framework to specify instructions for importing or exporting data to or from an external file in ASCII format. You can use Maps to specify instructions when the symbol names in your file do not match the symbol names in your Longview database.

Creating ImportSpecs

ImportSpecs stored as ASCII file and can be created and edited in any text editor. ImportSpecs support importing values up to nine decimal places (i.e. 1000.123456789).

The recommended file extension for ImportSpecs is .lvimp. For information on the functions used in an ExportSpec, see ImportSpec and ExportSpec Functions.

Sample ImportSpec for DataAreas

Example Code

```
DataSource TEXT, "DemoBuild\Data\Deferred Tax Setup\DeferredTaxSetup.txt",
EXTERNAL, "{"


Set Acc, 1

Set Tim, 2

Set Ent, 3

Set Det, 4

Set Cur, 5

Set Seg, 6

Set Ele, 7

Set Con, 8

Set Val, 9

Define ACCOUNTS, Match, Acc

Define TIMEPER, Match, Tim

Define ENTITIES, Match, Ent

Define DETAILS, Match, Det

Define CURRENCY, Match, Cur

Define SEGMENTS, Match, Seg

Define ELEMENTS, Match, Ele
```

```
Define CONTROLS, Match, Con

ValueField Val
```

Sample text sour

ce file and ImportSpec for DataTables

The following is an example of a source file that could be specified in an ImportSpec.

```
228468{John{Smith{CEO{Full Time{2,475,424.12
394851{Anita{Lane{CFO{Full Time{1,654,310.32
254876{Andrew{Croft{COO{Full Time{645,934.43
352456{Brad{Randall{CIO{Full Time{358,903.23
```

An ImportSpec for the above source file could look like this:

Example: Code

```
DataSource TEXT, "Salary Reporting\ExecutiveSalaries.txt", EXTERNAL, "{"
Set ID, 1
Set FNM, 2
Set LNM, 3
Set Job, 4
Set Amount, 6
Define USERID, MATCH, ID
Define FirstName, MATCH, FNM
Define LastName, MATCH, LNM
Define Role, MATCH, Job
Define Salary, MATCH, Amount
DecimalCharacter "REGIONAL"
```

Note: In this example, the developer chose to omit the fifth column (Employment Type) from the ImportSpec for the DataTable.

Sample JSON source files and ImportSpec for DataArea Import.

The following is an example of a JSON source filed that could be used in an ImportSpec.

Verbose


```
[{
            "Acc":  "A11101",
            "Tim":  "A1801",
            "Ent":  "TORONTO",
            "Det":  "SUMMDET",
            "Cur":  "CCAD",
            "Seg":  "SUMMSEG",
            "Ele":  "DIM6SET",
            "Con":  "DIM7SEt",
            "Val":  14999,
      }, {
            "Acc":  "A11102",
            "Tim":  "A1801",
            "Ent":  "TORONTO",
            "Det":  "SUMMDET",
            "Cur":  "CCAD",
            "Seg":  "SUMMSEG",
            "Ele":  "DIM6SET",
            "Con":  "DIM7SEt",
            "Val":  84024,
      }]
```


Concise


```
{"metadata":["Acc","Tim","Ent","Det","Cur","Seg","Ele","Con","Val"],
"data":[
    ["A11101","A1801","TORONTO","SUMMDET","CCAD","SUMMSEG","DIM6SET","DIM7SET",14999],
    ["A11102","A1801","TORONTO","SUMMDET","CCAD","SUMMSEG","DIM6SET","DIM7SET",84024],
    ]}
```


An ImportSpec for either of the above source file could look like this:

Example: Code

```
DataSource JSON, "DemoBuild\Data\Trialbal.json", EXTERNAL


Set Acc, 1

Set Tim, 2

Set Ent, 3

Set Det, 4

Set Cur, 5

Set Seg, 6

Set Ele, 7

Set Con, 8

Set Val, 9

Define ACCOUNTS, Match, Acc

Define TIMEPER, Match, Tim

Define ENTITIES, Match, Ent

Define DETAILS, Match, Det

Define CURRENCY, Match, Cur

Define SEGMENTS, Match, Seg

Define ELEMENTS, Match, Ele

Define CONTROLS, Match, Con

ValueField Val
```

Creating ExportSpecs

ExportSpecs are stored as ASCII files and can be created and edited in any text editor. ExportSpecs support exporting values up to nine decimal places (i.e. 1000.123456789).

The recommended file extension for ExportSpecs is .lvexp. For information on the functions used in an ExportSpec, see ImportSpec and ExportSpec Functions.

Sample ExportSpec for DataAreas

Example: Code

```
DataTarget TEXT, "DemoBuild\Data\Deferred Tax Setup\DeferredTaxSetup.txt",
EXTERNAL, "{"

Set Acc, 1
Set Tim, 2
Set Ent, 3
Set Det, 4
Set Cur, 5
Set Seg, 6
Set Ele, 7
Set Con, 8
Set Val, 9

Define ACCOUNTS, Match, Acc
Define TIMEPER, Match, Tim
Define ENTITIES, Match, Ent
Define DETAILS, Match, Det
Define CURRENCY, Match, Cur
Define SEGMENTS, Match, Seg
Define ELEMENTS, Match, Ele
Define CONTROLS, Match, Con

ValueField Val
```

Sample ExportSpec and target file for DataTables

Example: Code

```
DataTarget TEXT, "Salary Reporting\MiddleMgmtSalaries.txt", EXTERNAL, "{"
```

```
Set First, 1
Set Last, 2
Set ID, 3
Set Amount, 4
Set Job, 5
Define FirstName, MATCH, First
Define LastName, MATCH, Last
Define USERID, MATCH, ID
Define Salary, MATCH, Amount
Define Role, MATCH, Job
DecimalCharacter "."
```

The resulting target file could look like this:

```
Alan{Jenner{jen354{134,700.00{Support Manager
Alona{Strahovski{str124{126,934.25{Lead Engineer
Devon{Burton{bur245{146,793.00{Director of Marketing
Maria{Ribeiro{rib267{164,543.00{Logistics Manager
```

Sample JSON ExportSpec for DataAreas

Example: Code

```
DataTarget JSON, "DemoBuild\Data\Trialbal.json", EXTERNAL, VERBOSE


Set Acc, 1
Set Tim, 2
Set Ent, 3
Set Det, 4
Set Cur, 5
Set Seg, 6
Set Ele, 7
Set Con, 8
Set Val, 9

Define ACCOUNTS, Match, Acc
Define TIMEPER, Match, Tim
```

```
Define ENTITIES, Match, Ent
Define DETAILS, Match, Det
Define CURRENCY, Match, Cur
Define SEGMENTS, Match, Seg
Define ELEMENTS, Match, Ele
Define CONTROLS, Match, Con

ValueField Val
```

The resulting target file could look like this:


```
[{
            "Acc":  "A11101",
            "Tim":  "A1801",
            "Ent":  "TORONTO",
            "Det":  "SUMMDET",
            "Cur":  "CCAD",
            "Seg":  "SUMMSEG",
            "Ele":  "DIM6SET",
            "Con":  "DIM7SEt",
            "Val":  14999,
      }, {
            "Acc":  "A11102",
            "Tim":  "A1801",
            "Ent":  "TORONTO",
            "Det":  "SUMMDET",
            "Cur":  "CCAD",
            "Seg":  "SUMMSEG",
            "Ele":  "DIM6SET",
            "Con":  "DIM7SEt",
            "Val":  84024,
      }]
```


Example:

```
DataSource TEXT, "Salary Reporting\ExecutiveSalaries.txt", EXTERNAL, "{"

Set ID, 1
Set FNM, 2
Set LNM, 3
Set Job, 4
Set Amount, 6
Define USERID, MATCH, ID
Define FirstName, MATCH, FNM
Define LastName, MATCH, LNM
```

```
Define Role, MATCH, Job
Define Salary, MATCH, Amount
DecimalCharacter "REGIONAL"
```

Note: In this example, the developer chose to omit the fifth column (Employment Type) from the ImportSpec for the DataTable.

Creating external Maps

You can use Maps to specify instructions when the symbol names in your file do not match the symbol names in your Longview database.

External maps are maps that are not created in your Longview database. If you want to create an internal map, you can do so using the Mappings editor. For more information, see the “Mappings Editor Help”.

The recommended file extension for external Maps is .lvmap.

Understanding mappings methods and expressions

Maps contain instructions for importing data when the symbol names in your source file do not match the symbol names in the Longview database. A map contains a number of mappings, which specify methods for the way in which a source symbol should be mapped to a Longview database symbol using an expression.

You can create internal maps using the Mappings editor and external maps in Longview Application Framework.

For information on a creating a sample external map, see Sample External Map in the *Longview Developer’s Guide*.

You can use one of four methods in a mapping:

- Exact
- Wildcard
- Range
- In

You can use all alphanumeric and ASCII characters in an expression except for double quotation marks (”). The following are the rules for each method:

Exact

Use this method to match the expression value exactly. For example, use the EXACT method to map a single source symbol name to a Longview database symbol name exactly as indicated in the expression.

Note: Special characters are matched exactly when used in an expression for the Exact method.

- Internal syntax

Symbol	Method	Expression
0100	Exact	0100
0105	Exact	0105

- External syntax

```
Map Cash, EXACT, 10000
Map AccountsReceivable, EXACT, 20000
```

Wildcard

Use this method to match the expression field partially by including wildcards in an expression. This method is case sensitive. Use the question mark (?) to specify any one character as wildcard. Use the asterisk (*) to specify zero or more characters as wildcards. For example, use the WILDCARD method to import source symbols beginning with certain characters to a Longview database symbol name.

- Internal syntax

Symbol	Method	Expression
0300	Wildcard	0300???
1300	Wildcard	130*

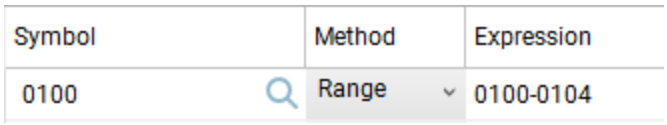
- External syntax

```
Map Cash, WILDCARD, 1000?
Map AccountsReceivable, WILDCARD, 200*
Map AccountsPayable, WILDCARD, 211*0
```

Range

Use this method to match a range of values, separated by a hyphen. This method is case sensitive. For example, use the RANGE method when you have a range of source symbol names to import to a Longview database symbol name.

- Internal syntax



Symbol	Method	Expression
0100	Range	0100-0104

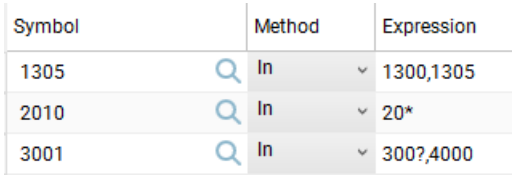
- External syntax

```
Map Cash, RANGE, 10000-10009
```

In

Use this method to match the expression value to one of several values. For example, use the IN method when you have a set of source symbol names to import to a Longview database symbol name. This method is case sensitive. You may also use the wildcard characters question mark (?) and asterisk (*) in the IN clause.

- Internal syntax



Symbol	Method	Expression
1305	In	1300,1305
2010	In	20*
3001	In	300?,4000

- External syntax

```
Map E12130, IN, “E12130,E11130, E11140, E12000”
Map E13100, IN, “E13*,E00000”
Map E11110, IN, “E11??0,E11111”
```

Sample External Map

Example code:

```
Format STANDARD
DefineDimension ACCOUNTS
Map 10010, RANGE, "10011-10013"
Map 10021, EXACT, "A100"
Map 10030, WILDCARD, "10030??"
Map 10040, WILDCARD, "1004*"
Map 10040, WILDCARD, "*100"
Map 10000,IN, “10000,10005
```

ImportSpec and ExportSpec Functions

A function is an instruction word representing an action you want to perform in Longview Application Framework. You can carry out the action by typing a function statement in an ImportSpec or ExportSpec Document in Longview Application Framework.

Note: Unless otherwise specified, the following functions are available for use in both ImportSpecs and ExportSpecs.

The following functions apply to DataAreas:

Assign	ErrorFile	QuotedStrings
DataSource (ImportSpec only)	IgnoreHeaderRecords	ReverseSign
DataTarget (ExportSpec only)	IgnoreFooterRecords	ScientificNotation (ImportSpec only)
DecimalCharacter	IncludeHeader (ExportSpec only)	Search
v26.2 DecimalPlaces (ExportSpec only)	LogFile	Set
Define (for DataAreas)	MaxError	ValueField
DuplicateMappings	Put	
DuplicateRecords		

The following functions apply to DataTables:

DataSource (ImportSpec only)	IgnoreHeaderRecords	Search
DataTarget (ExportSpec only)	IgnoreFooterRecords	Set
DecimalCharacter	IncludeHeader (ExportSpec only)	StringEncodedSymbolNames
v26.2 DecimalPlaces (ExportSpec only)	LogFile	
Define (for DataTables)	QuotedStrings	
ErrorFile	RegionalDates	

Assign

This function does not apply to ImportSpecs and ExportSpecs for DataTables.

Use this Longview Application Framework function to designate dimension names when each external record contains, or will get, multiple value fields, providing a way to associate each value field with specific symbols.

The ASSIGN function can only be used in a single dimension.

Syntax:

```
ASSIGN DimName
```

where:

- DimName is the dimension across which the multiple values will be associated.

The ASSIGN function must be used with the PUT function, and can only be used if the dimension has not been set by DEFINE, and if the VALUEFIELD function has not been used.

See also

- Define (for DataAreas)
- Put
- ValueField

DataSource

Use this function with import specification to define the source of the data to be imported. This keyword is available in a ImportSpecs only, is mandatory, and specifies the type and source of data for the import operation. The parameters for the DataSource function depend on the specified type of connection.

Syntax For Text

```
DataSource  SourceType, "FileName", EXTERNAL[, "Delimiter|REGIONAL"]
```

where:

- SourceType is one of the following:

Parameter	Description
TEXT	Imports the data from a delimited text file.
JSON	Imports the data from a JSON formatted file. The JSON file can be either verbose or concise format. The import will automatically determine which one it is.
ODBC	Imports the data directly from a database using ODBC connection.

- FileName is the name of the source file and it must be enclosed in double quotation marks.

> **Note:** By default, ODBC connections through Application Framework are allowed. To restrict all ODBC connections through Application Framework, open the lv_af.cfg file found in the working directory of the lv_af.exe and add the following statement to the end of the document: `PERMIT_DATASOURCE_ODBC=FALSE`

- Delimiter is the character that is used to separate the fields in the source file and must be enclosed in double quotation marks. If not specified, the default delimiter is {. Alternatively, you use the keyword REGIONAL, which will use the client machine's regional settings to determine the delimiter character to use. Use this parameter when using the TEXT.

Example:

```
DataSource TEXT, "MyData", EXTERNAL, "{"

DataSource JSON, "MyData", EXTERNAL
DataSource ODBC, "Driver={SQL Server Native
Client10.0};Server=localhost;Database=mydb ;Trusted_Connection=Yes;"
```

See also

- SQLBatchSize
- SQLStatement

DataTarget

Use this function with export specification to define the target of the data to be exported. This keyword is available in a Data Export Specification only, is mandatory, and specifies the target and type of data for the export operation.

Syntax:

```
DataTarget TargetType, "FileName", EXTERNAL, ["Delimiter|REGIONAL"]
[,VERBOSE|CONCISE]
```

where:

- TargetType is one of the following:

  Note: DataTables do not support ODS or JSON.

Parameter	Description
TEXT	Exports the data to a text file. You must use this keyword if you are exporting from a DataTable..
JSON	Exports the data in JSON format. The JSON file can be either verbose or concise format.
ODBC	Exports the data to an ODS file. For more information on ODS, see the “Longview Integration Guide”.

- "FileName" is the name of the source file and must be enclosed in double quotation marks.

  Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

- Delimiter is the character used to separate the fields in the ouput file and must be enclosed in double quotation marks. If not specified, the default delimiter is {. Alternatively, you use the keyword REGIONAL, which will use the client machine’s regional settings to determine the delimiter character to use. Use this parameter when using the TEXT.

  VERBOSE|CONCISE is the json format the data will be output as. This is an optional parameter and will default to VERBOSE if not specified. For more information on the output type VERBOSE and CONCISE see Sample JSON source files and ImportSpec for DataArea Import.

Syntax example:

```
DataTarget TEXT, "MyData", EXTERNAL

DataTarget TEXT, "MyDataTable", EXTERNAL, "REGIONAL"

DataTarget JSON, “MyData”, EXTERNAL, CONCISE

DataTarget JSON, “MyData”, EXTERNAL
```

DecimalCharacter

This function does not apply when JSON is used for DataSource or DataTarget.

Use this function to specify whether the character that is used as a decimal character, in the source or target file, is a comma or a period.

Syntax:

```
DecimalCharacter "Character|REGIONAL"
```

where:

- Character is the decimal character for the import or export file and can be either a period (.) or a comma (,). The default decimal character is a period (.).
- REGIONAL specifies to use the client machine's regional settings to determine the decimal character for import and export. This keyword must be enclosed in double quotation marks.

Syntax example:

```
DecimalCharacter ","

DecimalCharacter "REGIONAL"
```

v26.2 DecimalPlaces

Use this function to control how many decimal places are displayed for numeric values in export output. It applies to TEXT and JSON data targets, preserves trailing zeros, and defaults to 9 decimal places if not specified.

Syntax:

```
DecimalPlaces "FieldName","DecimalCount"
```

where:

- FieldName is the alphanumeric identifier defined in the set function. Set the FieldName to specify which numeric value should have decimal formatting applied. It can be either a literal value or a variable (for example value, $sFieldName$).
- DecimalCount is the number of decimal places to export (range 0-9). It can be either a literal number or a variable (for example 2, $nDecimalPlaces$).

Syntax example:

```
DecimalPlaces "value", "2"
```

See Also

- Put
- Set
- ValueField

Define (for DataAreas)

Use this function to specify how a field in the source file will map to the dimensions.

Syntax:

```
DEFINE DimName, MapMethod, FieldName|SymName[, MapName, INTERNAL|EXTERNAL]
```

where:

- DimName is the dimension being defined.
- MapMethod is a mapping method and is one of the following:

Value	Description
MATCH	The values encountered in field FieldName must match the dimension symbols exactly.
MAP	The values encountered in field FieldName will be mapped between dimension symbols via the Map specified by MapName.
UNIQUE	Import operations: all values encountered will be placed in the symbol SymName (symbol names are not derived from any source field). Export operations: only values in the source DataArea belonging to the symbol SymName will be exported.
CONSECUTIVE	Values will be placed consecutively (in the order they are encountered) in the dimension starting at symbol SymName, moving in priority order through that symbol's siblings. CONSECUTIVE is designed to work with a symbol existing in only one hierarchy - if the symbol belongs to more than one hierarchy, the behavior used to determine the siblings is undefined and may be affected by hierarchy reorganizations and import/exports. If the dimension is a schedule dimension, there is no concept of hierarchy or priority, therefore symbols will be used in defined order. If the end of the sibling or symbol list is reached, the next value will generate an error. **Note:** The CONSECUTIVE parameter is available for use in ImportSpecs only.

- FieldName|SymName is one of the following:

Value	Description
FieldName	This parameter applies only when MapMethod is MATCH or MAP. This can be any alphanumeric identifier, and is used only within the scope of the import action. This parameter must match exactly the FieldName specified in your Set function.
SymName	This parameter applies only when MapMethod is UNIQUE or CONSECUTIVE. This is the name of a valid symbol for the specified dimension.

- MapName is required when MapMethod is MAP and is one of the following:
  - the name of a map that exists inside the database, created using the Mappings editor. You must specify INTERNAL as the map location for this type of map.

- the path and file name, including the relevant extension, of a map that exists as a file, such as a text file on your local machine or network location. If you do not specify the path, the system assumes the path is `C:\...\MyDocuments\Longview`. You must specify EXTERNAL as the map location for this type of map.

A valid ImportSpec or ExportSpec must contain a DEFINE or ASSIGN for each base dimension in the DataArea. If the DataArea contains schedule data, each extra dimension must also relate to a DEFINE or ASSIGN.

Syntax example:

```
DEFINE Accounts, MATCH, Acct


DEFINE TimePeriods, MAP, TimePer, TimePeriodsMap, INTERNAL
DEFINE Entities, MAP, Ent, MyMaps\EntitiesMap.txt, EXTERNAL

DEFINE Currencies, UNIQUE, USD

DEFINE Details, CONSECUTIVE, AgingT

It is possible to use a single named field in an Import Spec with one or
more keywords (Define, Put, ValueField) to avoid outside transformation of
the source file.

To achieve this, you can use multiple define statements which refer to the
same fieldname but different map files. For example:
```

Sample Data File

```
72740000, Bike, 123.45

72740001, Skateboard, 987.65

        73740000, Bike, 12.34

73740001, Skateboard, 98.76
```

Sample ImportSpec

```
Set AccEnt, 1

Set Tim, 2

Set Det, 4

Set Cur, 5

Set Seg, 6

Set Ele, 7

Set Con, 8

Set Val, 9

Define ACCOUNTS, Map, AccEnt, AccountMapping, INTERNAL
```

```
Define TIMEPER, Match, Tim
Define ENTITIES, Map, AccEnt, EntityMapping, INTERNAL
Define DETAILS, Match, Det
Define CURRENCY, Match, Cur
Define SEGMENTS, Match, Seg
Define ELEMENTS, Match, Ele
Define CONTROLS, Match, Con
ValueField Val
```

Sample AccountMapping

```
40000, Wildcard, ???40000
40001, Wildcard, ???40001
```

Sample Entity Mapping

```
727, Wildcard, 727*

737, Wildcard, 737*
```

See also

- Assign
- Set
- ValueField

Define (for DataTables)

Use this function to specify how a field in the source file maps to a DataTable field.

Syntax:

```
DEFINE TableFieldName, MapMethod, FieldName|SymName[, MapName,
INTERNAL|EXTERNAL]
```

where:

- TableFieldName is the name of the DataTable field to which to map the field in the source file.
- MapMethod is a mapping method and is one of the following:

Value	Description
MATCH	The values encountered in field FieldName must match the dimension symbols exactly.
MAP	The values encountered in field FieldName will be mapped between dimension symbols via the Map specified by MapName.
UNIQUE	Import operations: The SymName will be placed in the corresponding TableFieldName.

- FieldName|SymName is one of the following:

Value	Description
FieldName	This parameter applies only when MapMethod is MATCH or MAP. This can be any alphanumeric identifier, and is used only within the scope of the import action. This parameter must match exactly the FieldName specified in your Set function.
SymName	This parameter applies only when MapMethod is UNIQUE. This is the name of a valid symbol for the specified dimension.

- MapName is required when MapMethod is MAP and is one of the following:
  - the name of a map that exists inside the database, created using the Mappings editor. You must specify INTERNAL as the map location for this type of map.
  - the path and file name, including the relevant extension, of a map that exists as a file, such as a text file on your local machine or network location. If you do not specify the path, the system assumes the path is `C:\...\MyDocuments\Longview`. You must specify EXTERNAL as the map location for this type of map.

Syntax example:

```
Define Account, Map, Acc, MyAccountMap, INTERNAL
Define CostCentre, Match, CostCentre
Define Company, Map, CompCode, MyCompanyMap, INTERNAL
Define Currency, Match, Curr
Define PostPeriod, Unique, [[ SYSTEM, SGPCurrentPeriod, DBDEFAULT ]]
Define Amount, Match, Val
Define DebitOrCredit, Match, DRCR
Define PostDate, Match, PostDate
Define MyColumnName, Match, FieldX
```

It is possible to use a single named field in an Import Spec multiple times to avoid certain external transformations of the source file. To achieve this, you can use multiple define statements which refer to the same FieldName but different map files.

For example, if Field1 in the source file is similar to Account.CostCenter.CompanyCode.Currency (e.g.: 11000.007.123.CAD), you can use Define statements similar to the following:

```
Define Account, Map, Field1, MyAccountMap, INTERNAL
Define CostCentre, Map, Field1, MyCCMap, INTERNAL
Define Company, Map, Field1, MyCompanyMap, INTERNAL
Define Currency, Map, Field1, MyCurrMap, INTERNAL
```

DuplicateMappings

Use this Longview Application Framework function to specify how your process handles duplicate mappings. Duplicate mappings can occur when a dimension's DEFINE function is set to MAP, and the specified field's value (for imports) or symbol name (for exports) for a given record matches multiple maps in the Symbol Map.

Syntax:

```
DUPLICATEMAPPINGS Action
```

where:

- Action is the action to take when the system encounters duplicate mappings in a source file. Select one of the following:

Value	Description
ALLOW	Specifies that duplicate maps are permitted. In this case, if multiple EXACT maps are found, all mappings will be used so that the same value is imported to multiple data cells. Import operations only: If no EXACT maps are found, the value will get imported to only one location, the one corresponding to the first MAP listed.
DISALLOW	Specifies that duplicate maps should not be allowed. The first entry is submitted. If a duplicate map is encountered, the system reports an error. **Note:** This is the default setting.

Syntax example:

```
DUPLICATEMAPPINGS ALLOW
```

DuplicateRecords

This function does not apply to ImportSpecs and ExportSpecs for DataTables. Use this Longview Application Framework function to specify how your process handles duplicate records.

Caution: Using the ADD option adds values in the file to the existing DataArea value. Using all other options overwrites existing DataArea values. Longview recommends clearing the DataArea first before using the ADD option.

Syntax:

```
DUPLICATERECORDS Action
```

where:

- Action is the action to take when the system encounters duplicate records in a source file. Select one of the following:
  - For use in both ImportSpecs and ExportSpecs:

Value	Description
ADD	Specifies that duplicate records should be allowed, with the value of all such records being summed. In this mode, if a duplicate record is encountered that contains text (versus numeric) data, it will be treated as an invalid/error record.
DISALLOW	Specifies that duplicate records should not be allowed. The first entry is submitted. If a duplicate record is encountered, the system reports an error. **Note:** This is the default setting.

  - For use in ImportSpecs only:

Value	Description
USEFIRST	Specifies that the first entry of a duplicate record should be used. If a duplicate record is encountered, the system does not report an error.
USELAST	Specifies that the last entry of a duplicate record should be used. If a duplicate record is encountered, the system does not report an error.

  - For use in ExportSpecs only:

Value	Description
ALLOW	Specifies that duplicate records are allowed, and are all exported to the target. **Note:** This option is not allowed if the Assign or Put functions are used.

Syntax example:

```
DUPLICATERECORDS USELAST
```

See also

- Reconcile Upload

ErrorFile

Use this Longview Application Framework function to specify the name of the error file to contain data on invalid records.

This will create a text file in the default folder. All rejected records will be captured in this file.

Syntax:

```
ERRORFILE FileName [,EXTERNAL]
```

where:

- FileName is the name of the error file to contain data on invalid records. It can include a complete or partial folder path in the format `C:\...\FileName`. If FileName includes spaces, enclose it in double quotation marks; for example:

  `"C:\My Documents\My Data.txt"`

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

EXTERNAL is an optional parameter that specifies the file will be externally created. If this parameter is not specified, still the file will be considered as external.

Syntax example:

```
ERRORFILE error.txt
```

IgnoreHeaderRecords

This function does not apply to:

- ExportSpecs
- ImportSpecs using DataSource JSON

Use this Longview Application Framework function to specify the number of header rows to ignore on import. If you do not include this function in your import specification, the system assumes your import file starts with data records with no header rows.

Syntax

```
IGNOREHEADERRECORDS Num
```

where:

- Num is the number of header records to ignore. Blank lines at the top of the import file are treated as records.

Syntax example:

```
IGNOREHEADERRECORDS 2
```

IgnoreFooterRecords

This function does not apply to .

- ExportSpecs
- ImportSpecs using DataSource JSON

Use this Longview Application Framework function

to specify the number of footer rows to ignore on import. If you do not include this function in your import specification, the system assumes your import file ends with data records with no footer rows.

Syntax:

```
IGNOREFOOTERRECORDS Num
```

where:

- Num is the number of header records to ignore. Blank lines at the bottom of the import file are treated as records.

Syntax example:

```
IGNOREFOOTERRECORDS 2
```

IncludeHeader

This function does not apply when DataTarget uses JSON.

Use this Longview Application Framework function to export a default or specific header rows when exporting from a DataArea or DataTable.

Syntax:

```
IncludeHeader AUTO|SPECIFY[, "line 1", "line 2", ...]
```

where:

- AUTO auto-populates the header with the dimension descriptions for each dimension and VALUE for the value field. If there are multiple value fields, the symbol description will be output instead of VALUE.
- SPECIFY allows you to specify a custom header. The header can be a single line or multiple lines. Each line must be enclosed in quotes and separated by a comma.

Syntax example:

```
INCLUDEHEADER AUTOINCLUDEHEADER SPECIFY, "Account, Entity, Time Period,
Value"
INCLUDEHEADER SPECIFY, "File created on May 31 2017", "****"
```

LogFile

Use this Longview Application Framework function to specify the name of the log file to contain information of the processing of the source or target file. This will create a text file in the default folder. All log information will be captured in this file.

Syntax:

```
ERRORFILE FileName[,EXTERNAL]
```

where:

- FileName is the name of the error file to contain data on invalid records. It can include a complete or partial folder path in the format C:\...\FileName. If FileName includes spaces, enclose it in double quotation marks; for example:

  ```
  "C:\My Documents\My Data.txt"
  ```

Note: If the document is in the same location as lv_af.exe, you do not need to specify the drive or path. If the path specified does not exist, it will be created. The folder path can have a maximum length of 260 characters.

EXTERNAL is an optional parameter that specifies the file will be externally created. If this parameter is not specified, still the file will be considered as external.

Syntax Examples

```
LOGFILE log.txt
```

MaxError

This function does not apply to ImportSpecs and ExportSpecs for DataTables. For DataTables, this function is always set to 1.

Use this Longview Application Framework function to specify the maximum number of error records to permit before stopping a process. This function is optional.

Syntax:

```
MAXERROR NumErrorRecords
```

where:

- NumErrorRecords is the maximum number of error records to permit before stopping an import. The default value for this function is 0.

Syntax Examples

```
MAXERROR 1
```

Put

This function does not apply to ImportSpecs and ExportSpecs for DataTables.

Use this Longview Application Framework function in both ImportSpecs and ExportSpecs to designate symbol and field names when each external record contains (or will get) multiple value fields, providing a way to associate each value field with specific symbols.

Syntax:

```
PUT SymName, FieldName
```

where:

- SymName is a symbol name in this dimension.
- FieldName is the field with which this symbol is to be associated.

The PUT function can be used only if the ASSIGN dimension has been specified, and multiple PUT functions should be specified (targeting multiple fields to multiple symbols). This function cannot be used in conjunction with the ValueField function.

See also

- Assign

QuotedStrings

If you're importing from or exporting to a .csv file, you may need to enclose string values in double quotation marks. Use this function in an ImportSpec to remove the double quotation marks from string values on import. Similarly, use this function in an ExportSpec to enclose string values in double quotation marks on export to .csv.

Syntax:

```
QuotedStrings ON|OFF
```

where:

- ON encodes/decodes strings with double quotation marks.
- OFF indicates that string values should be exported or imported as-is.

Syntax Examples

```
QuotedStrings ON
```

RegionalDates

This function is available only for DataTables.

The system stores dates for DataTable objects and App tables in ISO 8601 format (yyyy-mm-dd). Use this function in an ExportSpec to convert dates from ISO 8601 format to the format specified by the client machine's regional settings.

Similarly, you can also use this function in an ImportSpec to convert dates from the format specified by the client machine's regional settings to ISO 8601.

This can be useful when working with DataTable data in a .csv file that you plan to import into your Longview system.

Syntax:

```
RegionalDates ON|OFF
```

where:

- ON converts dates from ISO 8601 to regional format and vice-versa.
- OFF imports and exports date values without any conversion.

Syntax Examples

```
RegionalDates ON
```

ReverseSign

This function does not apply to ImportSpecs and ExportSpecs for DataTables.

Use this Longview Application Framework function to toggle logic dealing with sign reversal of numeric values.

When this logic is enabled (ON), the system considers the Account symbol's "balance type" property. If this property indicates a "credit" Account, the numeric value being imported or exported should have its sign reversed.

The default value for ReverseSign is OFF.

Syntax:

```
REVERSESIGN ON|OFF
```

Syntax Examples

```
REVERSESIGN ON
```

ScientificNotation

This function does not apply to ExportSpecs for DataAreas and DataTables and ImportSpecs for DataTables.

Use this function in an ImportSpec if the data in your import is in scientific notation format.

Syntax:

```
ScientificNotation ON|OFF
```

where:

- ON enables importing of data in scientific notation format.
- OFF disables importing of data in scientific notation format.

Syntax Examples

```
ScientificNotation ON
```

Search

Use this Longview Application Framework function to filter source records based on values of specified fields.

Syntax:

```
SEARCH SearchExpression
```

where:

- SearchExpression consists of one or more relational expressions, linked by AND and OR and grouped by brackets, as appropriate:

  ```
  [(] RelationalExpression [AND|OR RelationalExpression [)] [AND|OR …]]
  ```

  Each RelationalExpression is created using field names, operators and values, as follows:

```
RelationalExpression = FieldName EQ|NE|LE|LT|GE|GT "Value"
```

The value parameter may optionally contain wildcards (? and *), and symbols for operators, such as ==, <=, >=, and so on.

Multiple SEARCH expressions may appear within a single specification object.

Syntax Examples

```
SEARCH "ACCT==??1 OR ACCT==???2"
```

Set

Use this function to assign unique names identifying the structural elements of the source data file. You must use either the Define or ValueField functions for each Set function.

Syntax:

```
SET FieldName, Position
```

where:

- FieldName can be any alphanumeric identifier, and is only used within the scope of the import action.
- Position is a parameter that can be a field number (for delimited text files) or an element [.subelement] name (for XML files). Multiple SET functions are used to refer to various fields in the data object, but not all fields require the SET function.

Syntax Examples

```
SET Acct, 1

SET TimePer, 2
SET Ent, 3
SET Cur, 4
```

StringEncodedSymbolNames

This function is available only for DataTables.

If you're importing from or exporting to a .csv file, you may need to enclose symbol names in square brackets so that numeric symbol names are not interpreted as numbers (and the leading zeros are retained). Use this function in an ExportSpec to enclose symbol names in square brackets on export to .csv. Similarly, use this function in an ImportSpec to remove the square brackets from symbol names on import.

Note: This logic applies only to symbol names in columns that have the symbol data type.

Syntax:

```
StringEncodedSymbolNames ON|OFF
```

where:

- ON encodes/decodes symbol names with square brackets.
- OFF indicates that symbol names should be exported or imported as-is.

Syntax Examples

```
StringEncodedSymbolNames ON
```

SQLBatchSize

Use this function to specify the number of records to retrieve with each fetch against the ODBC data source. The default if this function is not specified is 1000 records. Specifying a larger number of records to retrieve may improve performance.

Note: This function applies only when DataSource is ODBC.

Syntax:

```
SQLBatchSize Size
```

where:

- Size specifies the number of records to retrieve with each fetch against the ODBC data source.

Syntax Examples

```
SQLBatchSize 500
```

SQLStatement

Use this function to specify the SQL statement to pass along to the ODBC data source.

Note: This function applies only when DataSource is ODBC.

Syntax:

```
SQLStatement Statement[, SQLFile]
```

where:

- Statement is the SQL statement to pass along to the ODBC datasource and can be one of the following:

Value	Description
“statement”	Specifies the SQL statement in text, enclosed in double quotation marks.
FILE	Specifies that the SQL statement is specified in a file.

- SQLFile is the file containing the SQL statement, including the extension, and applies only when Statement is FILE. The file name must be relative to the current directory.

Syntax Examples

```
SQLStatement SELECT Currency, Rate FROM Rates

SQLStatement FILE, Sample.sql
```

ValueField

This function does not apply to ImportSpecs and ExportSpecs for DataTables.

Use this Longview Application Framework function to specify which field will contain the actual data value being imported or exported.

Syntax:

```
VALUEFIELD FieldName
```

where:

- FieldName can be any alphanumeric identifier, and is only used within the scope of the import action.

See also

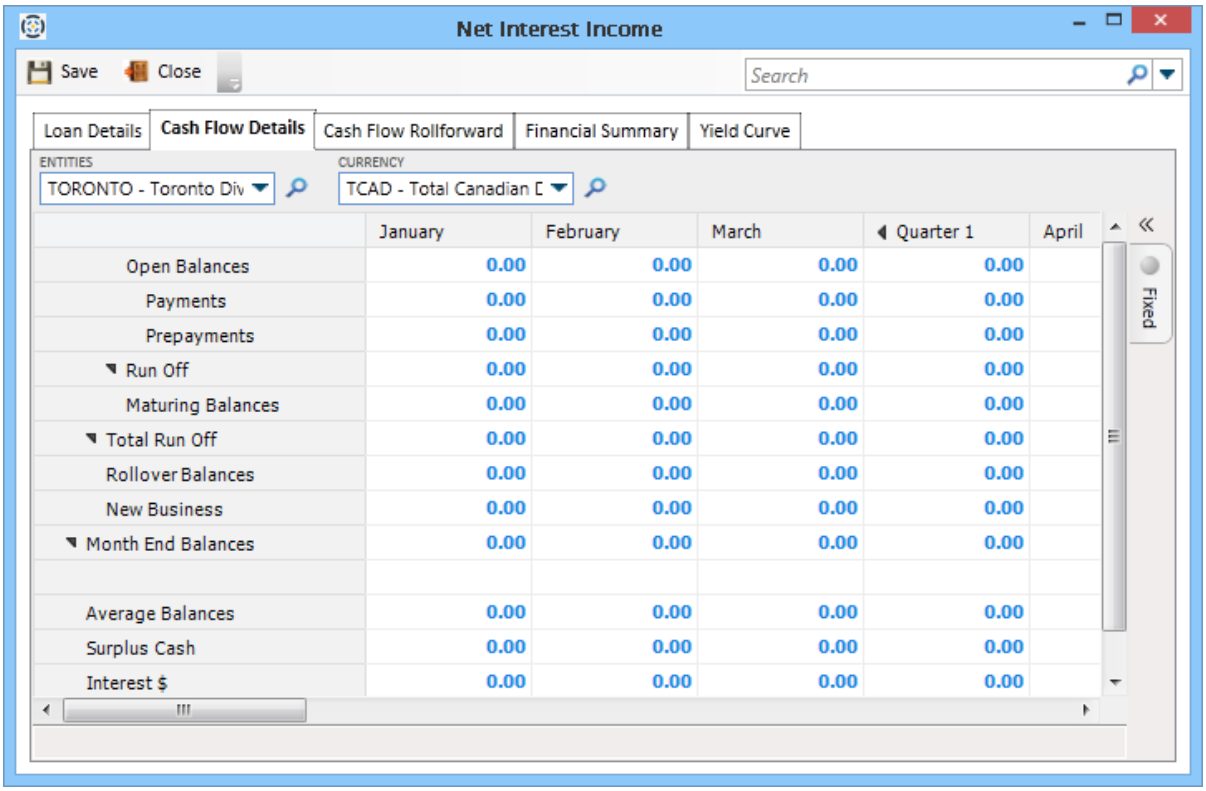
- Assign
- Define (for DataAreas)

Developing Longview Data Grids

You can create interactive Data Grids that enable users to work with data in the Longview database. The data displayed by a Data Grid is determined by the Data Area that the grid specifies in the Show DataArea function. For more information on this function, see Show DataArea.

Data View definition files define the layout, format, and behavior of data in a Data Grid. Use Data View functions in a Longview Data View definition (.lvdvw) file. The Data View definition file is also referenced by the Show DataArea command used in a procedure file.

The following graphic shows a typical Data Grid:



A Data Grid, such as the one above, could be defined in a Data View definition file similar to the following:

Example:

```
//Orientation

Rows Accounts
Columns TimePeriods

Fix Details, Local_Statutory
Fix Departments, Departments_Default
Fix Products, Products_Default
```

```
Fix DIMENSION8, DIMENSION8_Default
Fix DIMENSION10, DIMENSION10_Default
Fix DIMENSION11, DIMENSION11_Default
Fix DIMENSION12, DIMENSION12_Default
Fix DIMENSION13, DIMENSION13_Default
Fix ReportingSets, ReportingSets_Default
Fix ScalingFactors, ScalingFactors_Default

//Slice Entities and Currency
Slice Entities, TORONTO
Slice Currency, TCAD

//Appearance
DefaultDecimals 2
Parentheses ON
ThousandsSeparator ON

InsertSeparator BLANK, Accounts, Month_End_Balances, AFTER, 1

//Add Save and Close buttons
Action "Save", "NII\images\save.ico", "Upload NII", "Save Data",
toolbar|cellcontext, "parent & Leaf"
Action "Close", "NII\images\close.ico", "Show Return", "Close Data Grid",
toolbar|cellcontext, "parent & Leaf"

//End of data view
```

This section contains the Longview Application Framework functions you can use in a Data Grid. Click the links in the following table for detailed usage and syntax information.

Columns	MessageBar	SystemAction	LIDInputOnly
ColumnWidth	Rows	WindowTitle	NumericInputOnly
Expand	Search	ConditionalCellStyle	Parentheses
Fix	Slice	DefaultDecimals	Protect
Hide	SymbolWidth	DimensionFreeze	SymbolDecimals
Include	TitleWidth	DimensionTitles	SymbolSuppress
InsertSeparator	Action	DynamicRollup	TextInputOnly

Instructions	DynamicModel	Filter	ThousandsSeparator
LeftTitleText			

This section contains information on these main topics:

- Working with area specs
- Working with line item details
- Working with comments
- Working with attachments
- Defining Data Grid layouts
- Adding actions to Data Grids
- Specifying the order of toolbar actions
- Specifying title bar descriptions
- Formatting data in Data Grids
- Working with drop-down lists

Working With Area Specs

When you are designing your Data Grid, you must often define the area of the database to apply certain functions to. For instance, you must specify an area spec for the following functions:

- ConditionalCellStyle
- Filter
- LIDInputOnly
- NumericInputOnly
- Protect
- TextInputOnly

The area spec for a Data View function must be a subset of the data area as defined by the corresponding SHOW DATAAREA command.

You can specify symbols for each base and schedule dimension. Each dimension must be separated by a semicolon (;). A basic area spec has the following syntax:

```
Dimension1Symbol; Dimension2Symbol; Dimension3Symbol; ...DimensionNSymbol;
ScheduleDimension1Symbol; ScheduleDimension2Symbol
```

If you want to include all symbols in a dimension that are inside the relevant data area, you can leave the symbol specification empty. For example:

```
Dimension1Symbol;;;;;;;
```

You can also add multiple symbols for each dimension. Multiple symbols within a dimension must be separated by a pipe (|). For example:

```
Dimension1Symbol1|Dimension1Symbol2; Dimension2Symbol1|Dimension2Symbol2;
Dimension3Symbol; ...DimensionNSymbol1|DimensionNSymbol2|DimensionNSymbol3
```

In addition, you can use symbol specifications to specify the symbols to include in the area spec. The following table lists supported symbol specifications:

Specification	Meaning
symbol###	all leaf symbols under symbol
symbol#n	all symbols under symbol, n levels down, including symbol
symbol##n	all parent symbols under symbol, n levels down
symbol##+n	all parent symbols under symbol, n levels down, including symbol
symbol#*	all roots of symbol

Specification	Meaning
symbol#-n	all symbols exactly n levels down from symbol
symbol##^n	all ancestors of symbol that are n levels above symbol, including symbol
symbol##^n:root	all symbols above symbol, n levels up under root, including symbol, where “root” is the root symbol (for example, BALSHEET##^2:TRIALBAL)

Working With Line Item Details

If your organization requires line item details to provide more detailed accounting for data, you can create Data Grids that allow and enforce line item details.

Users can enter line item details under the following conditions:

- USEINPUTRELATEDSCHEDULES is set to ON or LineItemDetails in the related procedure. For more information on this function, see UseInputRelatedSchedules.
- The Data Grid has the time periods dimension as the only dimension in the across orientation. For more information, see Columns.
- The ZGPLineItemDetailReference attribute is set for the appropriate time periods in the Data Grid. For more information, see Enabling line item details.
- If the symbol set for the ZGPLineItemDetailReference attribute is different than the time period, you must include that symbol in the area spec for the Data Grid. For more information, see Working with area specs.
- Cells with line item details must not be set to text input only. For more information, see TextInputOnly.
- The cell with the line item details must not intersect with a drop-down list. For more information, see Working with drop-down lists.
- Rows with line item details must be a leaf or static parent row.

A green triangle in the bottom-right corner of a cell indicates that there are line item details associated with that data row.

Enabling line item details

You must set the ZGPLineItemDetailReference attribute for each time period in the Longview Data Grid that you expect to contain line item details. If you set this attribute at a parent or root time period, users can enter line item details for all the leaf symbols below that parent or root.

The purpose of this attribute is to identify the symbol in which to store the textual comment and description entered by contributors for each line of line item details. It typically points a plan time period root back to itself (for example, for PYR10, ZGPLineItemDetailReference should be set to PYR10).

The only exception to this rule would typically occur if a company has alternate time hierarchies that intersect and are both used in the Data Grid (for example, fiscal year vs. seasons). In this case, the ZGPLineItemDetailReference attributes for the alternate time hierarchy roots should all point to the same time period symbol.

Enforcing line item details

In certain cases, your company may want to enforce line item details for certain symbols. For example, you may want users to always provide detailed information for a travel expense account that breaks up the total into various categories, such as airfare, hotel, and meals. In this case, you can enforce line item details so that users enter the value for the appropriate category, such as Airfare and Hotel.

To enforce line item details, use the LIDInputOnly Data View function.

For more information, see LIDInputOnly.

Working With Comments

If your organization wants to allow users to provide more detailed information about data that they input, you can create Data Grids that allow comments. Comments are always optional and cannot be enforced. A red triangle in the top-right corner of a cell indicates that there is at least one comment associated with that data intersection.

In order for the Data Grid to support comments, you must set the USEINPUTRELATEDSCHEDULES function to ON or Comments in the related procedure. For more information on this function, see UseInputRelatedSchedules.

Note: To allow users to delete any existing comment, you must assign them Delete Comments authorization. Users without Delete Comments authorization can delete only their own comments before they are submitted to the database. For more information on authorizations, see the Longview Application Administrator Guide.

Working With Attachments

If your organization wants to allow users to provide supplemental information to a cell in a Data Grid, you can create Data Grids that support attachments. Attachments are always optional and cannot be enforced. Users with appropriate write access can attach files only to locked base data cells.

Users can add attachments under the following conditions:

- The File Attachment setting in Longview Server Manager is selected. For more information, see the Longview Server Manager Guide.
- USEINPUTRELATEDSCHEDULES is set to ON or Attachments in the related procedure. For more information on this function, see UseInputRelatedSchedules.

A paperclip icon in a cell indicates there is an attachment for that cell.

Defining Data Grid Layouts

Layout functions define the way dimensions are displayed in the Data Grid. When you create a Data Grid, you must choose the dimensions that appear in the rows and columns, as well as the dimensions that are fixed or sliced. You can specify a dimension as either a row, column, fixed, or sliced; you cannot specify a dimension as more than one of these categories.

Columns

Use this function to define the dimensions displayed as columns in the data view. This function is mandatory in a Data View definition file.

Syntax:

```
Columns Dimension1, Dimension2, ...
```

where:

- Dimension1, Dimension2, etc, are the dimensions to display as columns. If you specify multiple dimensions, they are nested. The first dimension specified is the outermost dimension, and any other dimensions appear in order from outer to inner, with the last dimension being the innermost dimension.

Example:

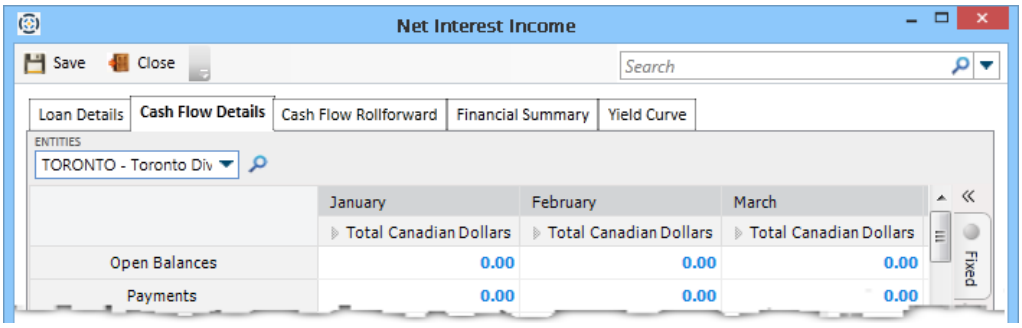
```
Columns TIMEPER
```

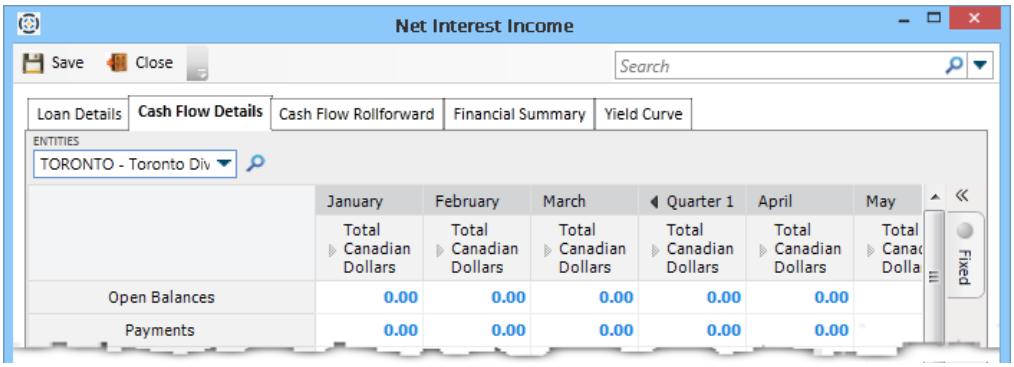
Related functions

- Fix
- Rows
- Slice

ColumnWidth

Use this function to specify the column width for the innermost dimension in the columns orientation and optionally wrap text in title cells. If you use the ColumnWidth function for any dimension other than the innermost across dimension, the width of columns remains unaltered.





Syntax:

```
ColumnWidth Dimension, WidthSpecification[, WrapTitle:ON|OFF]
```

where:

- Dimension is the name of the innermost dimension in the columns orientation to which the column width applies.
- WidthSpecification is one of the following:

Value	Description
FitToTitle	Automatically adjusts the width of each column individually to fit the text of the title cell for the specified dimension.
Width	Specifies the column width of the title cells in pixels for the specified dimension. If you specify Width as 0 or do not use the ColumnWidth function in your data view definition file, the default column width is 200 pixels.

- WrapTitle is optional and can be one of the following:

Value	Description
ON	Wraps title text to the next line in title cells, if necessary, in the specified dimension. If both FitToTitle and WrapTitle are specified, WrapTitle behavior is not noticeable until users manually adjust the width of columns.
OFF	Specifies the column width of the title cells in pixels for the specified dimension. If you specify Width as 0 or do not use the ColumnWidth function in your data view definition file, the default column width is 200 pixels.

Note: WrapTitle applies to titles only, not data.

Example:

```
ColumnWidth Currency, FitToTitle
```

```
ColumnWidth Currency, 75, WrapTitle:ON
```

Related functions

- SymbolWidth
- TitleWidth

Expand

Use this function to specify the initial expansion state of symbols in a Data Grid.

> **Note:** If you want an expanded symbol to appear in the initial view of the Data Grid, you must expand all ancestors of that symbol.

Syntax:

```
Expand Dimension, Symbol, ExpansionLevel[,
OuterDimension1Symbol1|OuterDimension1Symbol2, OuterDimension2Symbol1, ...]
```

where:

- Dimension is the dimension that contains the parent symbol you want to expand.
- Symbol is the parent symbol that you want to expand.
- ExpansionLevel is the number of levels to expand the parent symbol.
- OuterDimension1Symbol1 is the outer dimension symbol in which to expand the parent symbol. Separate symbols in the same outer dimension with a pipe (|). Separate symbols in different outer dimensions with a comma. Outer dimension symbols are optional. If you do not specify outer dimension symbols, all occurrences of the specified symbol are expanded.

Example:

```
Expand TIMEPER, PYr_Total, 2
 Expand ACCOUNTS, Trialbal, 1, Toronto|Montreal|Halifax, TCAD
```

Fix

Use this function to specify the dimension and symbol in your data view that are fixed. Fixed dimensions are listed on the Fixed tab. You can have more than one fixed dimension; however, each dimension can have only one fixed symbol.

Syntax:

```
Fix Dimension, Symbol
```

where:

- Dimension is the dimension to fix.
- Symbol is the symbol to fix.

Syntax:

```
Fix CURRENCIES, CAD
```

Related functions

- Columns
- Rows
- Slice

Hide

Use this function to hide specified symbols from a Data Grid. You may want to, for example, include a certain hierarchy in your Data Grid but not particular symbols within that hierarchy. The Hide function does not apply to Slice or Fix dimensions or to line item details.

Note: Data for hidden symbols is still affected by the DynamicModel and DynamicRollup functions.

Syntax:

```
Hide Dimension, Symbol[|Symbol2|...|SymbolN]
```

where:

- Dimension is the dimension that includes the symbols you want to hide.
- Symbol is the symbol to hide. Separate multiple symbols with a pipe (|).

Syntax:

```
Hide ENTITIES, Toronto
Hide ENTITIES, Toronto|Montreal|Ottawa
```

Include

Use this function to specify the symbols that are available in a Data Grid for the dimensions in the rows, columns, or slices. If you do not use this function for a dimension, all symbols in the underlying data area are included. You can specify multiple Include statements for a dimension. Symbols appear in the order of the statements.

Syntax:

```
Include Dimension, SymbolSpec [,ExpandLevels]
```

where:

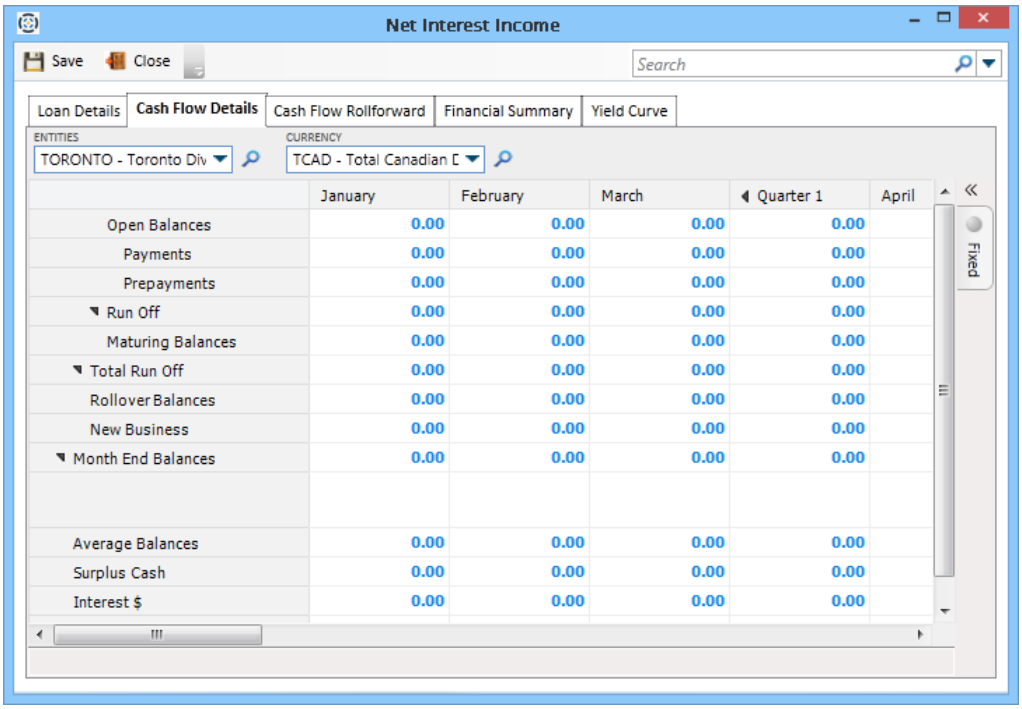
- Dimension is the dimension that contains the symbol to include.
- SymbolSpec contains the name of the symbols with any valid hierarchical syntax.
- ExpandLevels is optional and specifies the number of levels to expand the symbol when the Data Grid opens. ExpandLevels can have a value from 1 to 99. The default value is 1.

Example:

```
Include ACCOUNTS, Net_FA#99
```

InsertSeparator

Use this function to insert a blank row or column in a Data Grid. For example, in the grid below, there is a blank row after Month End Balances.



Syntax:

```
InsertSeparator BLANK, Dimension, Symbol, Position[, Number[, Size]]
```

where:

- Dimension is the dimension that contains the symbol to insert a blank row or column before or after.
- Symbol is the symbol that you want to insert a blank row or column before or after.

  Note: If you specify a symbol that is suppressed using the SymbolSuppress function and Apply Symbol Suppression is selected in the Options dialog, the separator is also suppressed.

- Position can be either BEFORE or AFTER.
- Number is the number of blank rows or columns to insert in the specified position. This parameter is optional. The default value for this parameter is 1.
- Size specifies the row height or column width in pixels. Use this parameter only if Number is specified. If Number is specified, this parameter is optional and has the following default values:

Value	Description
Rows	20 pixels
Columns	ColumnWidth specification

Note: If ColumnWidth is also not specified, the default is 200 pixels.

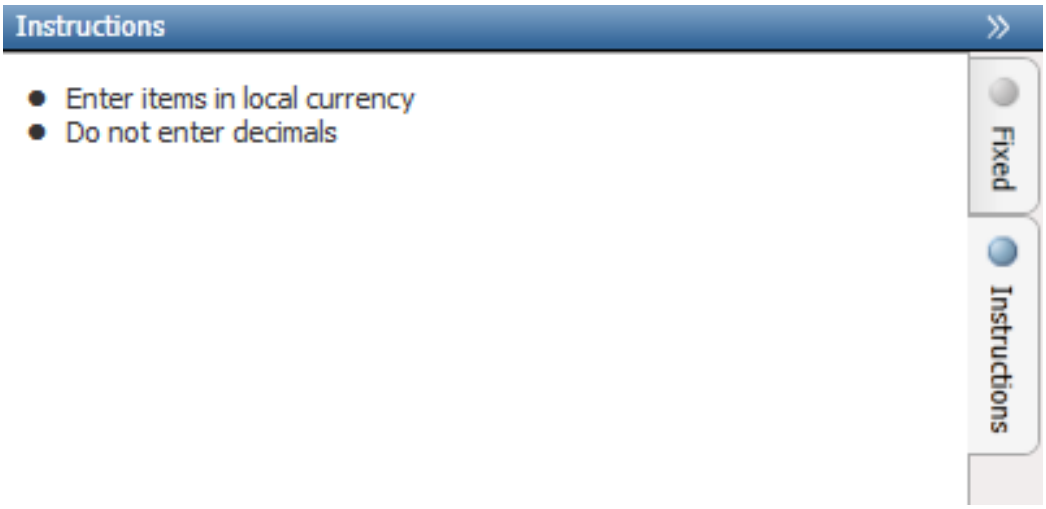
Example:

```
InsertSeparator BLANK, ACCOUNTS, Month_End_Balances, AFTER, 1, 40
```

Instructions

Use this function to include the Instructions tab in a Data Grid. The Instructions tab appears below the Fixed tab on the right-most edge of a Data Grid. You can write your own custom instructions using HTML.

Note: Instructions do not support HTML5..



Syntax:

```
Instructions Option, FileName
```

where:

- Option can be one of the following:

Value	Description
ON	Includes the Instructions tab in the Data Grid. If you specify Instructions ON, you must include FileName.
OFF	Hides the Instruction tab in the Data Grid. If you specify Instructions OFF, you do not need to include FileName. This is the default if the Instructions function is not included in the Data View definition file.

- FileName is the name of the .html file to display. The path for this file must be relative to the applications folder on your Data Server. If you specify Instructions OFF, you do not need to specify FileName.

Example:

```
Instructions ON, SampleInstructions.html
```

The following is an example of the HTML code you could write for a custom instructions file:

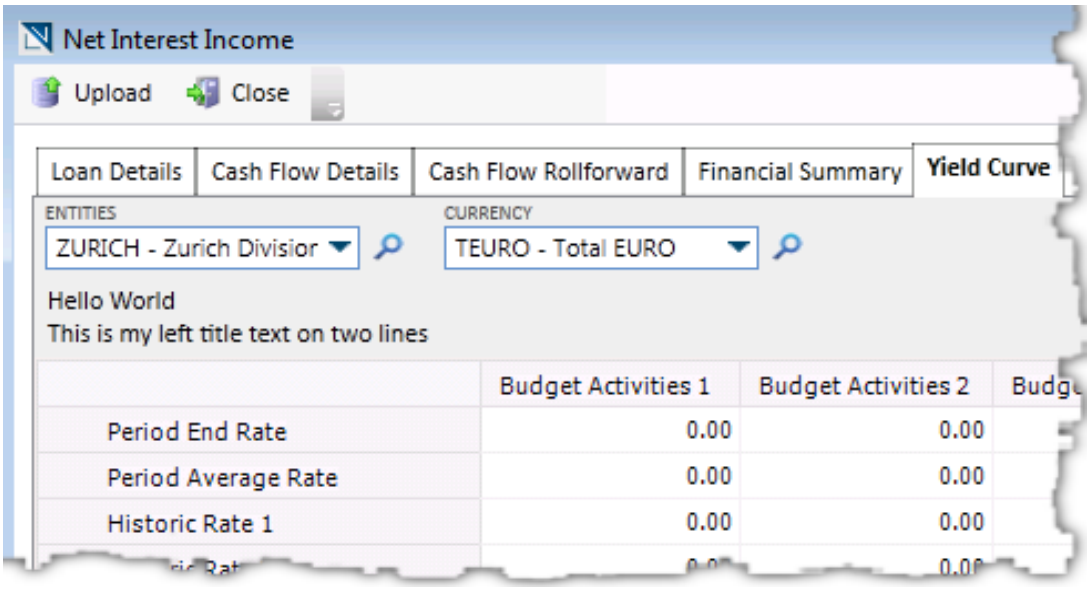
Example:

```
<html>
<head>
```

```
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title></title>
<style type="text/css">

body {color:#333333; font-family: Tahoma, Geneva, sans-serif; font-size:
11px; margin-top: 10px;}

ul {margin-left: 15px;}
</style>
</head>
<body>

<ul>

<li>Enter items in local currency</li>

<li>Do not enter decimals</li>

</ul>
<body>
</html>
```

LeftTitleText

Use this function to include additional information in the top left section of your Data Grid. Left title text appears just below slice dimensions or at the top of the Data Grid if you do not have slice dimensions. You can include attribute tokens and string variables in the text.



Syntax:

```
LeftTitleText "Text"
```

where:

- Text is the text to include, enclosed in double quotation marks. To include text on multiple lines, use \n as the line break character.

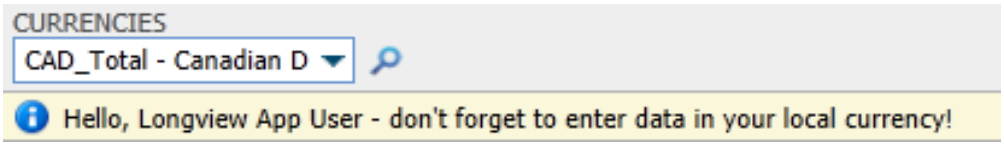
> **Note:** If you use variables or attribute tokens in the LeftTitleText function, variables and attribute tokens are not updated with toolbar button actions.

Example:

```
LeftTitleText "Hello World\nThis is my left title text on two lines"
```

MessageBar

Use this function to add text and an icon in a highlighted message bar at the top of a Longview App.



Syntax:

```
MessageBar MessageType, "Message"
```

where:

- MessageType determines the icon to display and can be one of the following values:
  - Error
  - Information
  - Warning
  - None
- Message is the message to include on a single line only, enclosed in double quotation marks. If you include the MessageBar function in your Data View document but do not specify a message by using double quotation marks (""), the message bar does not display.

> **Note:** If you use variables in MessageType or Message, variables are updated with toolbar button actions and the message bar is refreshed.

Example:

```
MessageBar Information, "Hello, Longview App User - don't forget to enter
data in your local currency!"
```

Rows

Use this function to define the dimensions displayed as rows in the Data Grid. This function is mandatory in a Data View definition file.

Syntax:

```
Rows Dimension1, Dimension2, ...
```

where:

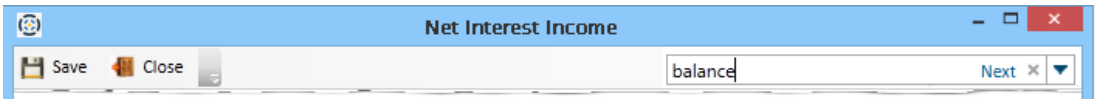
- Dimension1, Dimension2, and so on, are the dimensions to display in the rows. If you specify multiple dimensions, they are nested. The first dimension specified is the outermost dimension, and any other dimensions appear in order from outer to inner, with the last dimension as the innermost dimension.

Example:

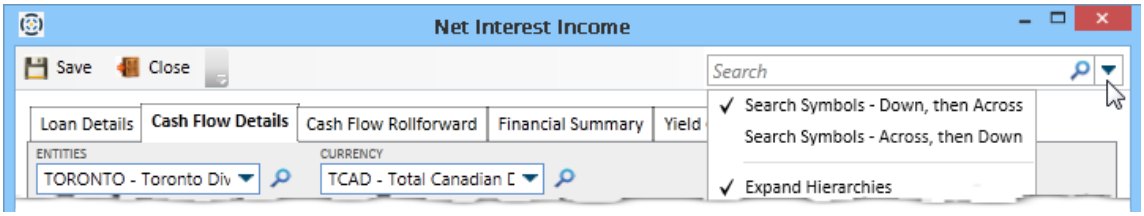
```
Rows ENTITIES, ACCOUNTS
```

Search

Use this function in your Data View definition file to customize the Data Grid Search box. You can use the Search function to specify the defaults for the Search options or to hide the Search box from a Data Grid. Unless you use this function to hide the Search box, it appears on the right-most edge of the toolbar of all Data Grids.



If the Search box is available in the Data Grid, Longview App users can modify the Search options within the Data Grid using the drop-down list. Longview App users cannot show or hide the Search box. Longview App users can use the Search box to search symbols only, including temporary symbols.



Note: If you have created a tabbed Data Grid, you can customize the Search box for each tab using the appropriate Data View definition file.

Syntax:

```
Search ON|OFF[, Order:OrderOption][, Expand:ExpandOption]
```

where:

- ON displays the Search box in the toolbar. This is the default if the Search function is not used in a Data View definition file.
- OFF hides the Search box in the toolbar.
- OrderOption can be one of the following

Value	Description
DOWNACROSS	Searches the down dimension symbols first, then the across dimension symbols. This is the default if Order is not specified.
ACROSSDOWN	Searches the across dimension symbols first, then the down dimension symbols.

- ExpandOption can be one of the following

Value	Description
ON	Expands hierarchies when the user navigates search results using the Next button. This is the default if Expand is not specified.
OFF	Maintains the expansion states of the Data Grid when the user navigates search results using the Next button.

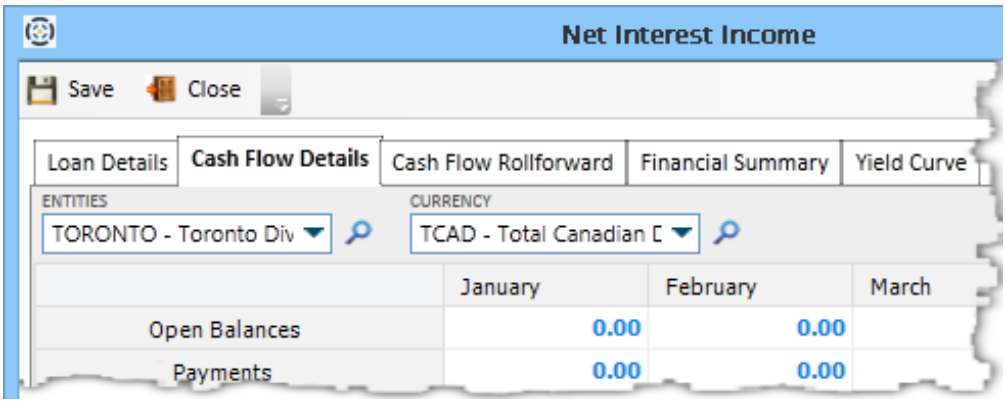
Note: For more information on using the Search box in a Data Grid, see the "*Longview Dashboard User's Help*".

Syntax:

```
Search ON, Order:ACROSSDOWN, Expand:OFF
```

Slice

Use this function to specify the dimensions that are available in your Data Grid as slice dimensions. Slice dimensions are dimensions that users can change the symbol selection for using the Slice list in the top left corner of the Data Grid. For example, in the Data Grid below, the Slice lists contain Entities and Currency, and the TORONTO - Toronto Division and TCAD - Total Canadian Dollars symbols are the default selections.



Note: If you do not specify a particular dimension in the Rows or Columns statements in the Data View definition file, that dimension is automatically available as a Slice dimension.

Syntax:

```
Slice Dimension[, Symbol]
```

where:

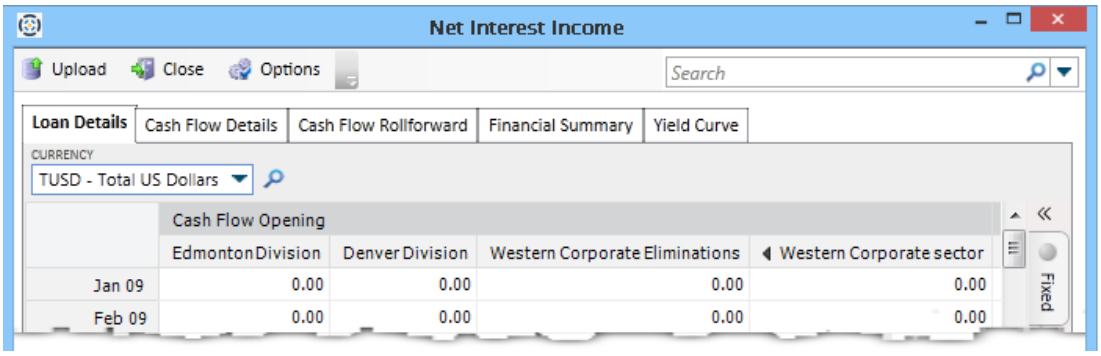
- Dimension is the dimension to make available as a slice dimension. Slice dimensions appear in the Slice list in the order specified in the Data View document.
- Symbol is optional and specifies the default symbol selected for the dimension. If you do not specify a symbol, the first symbol in the Data Area for that dimension is selected.

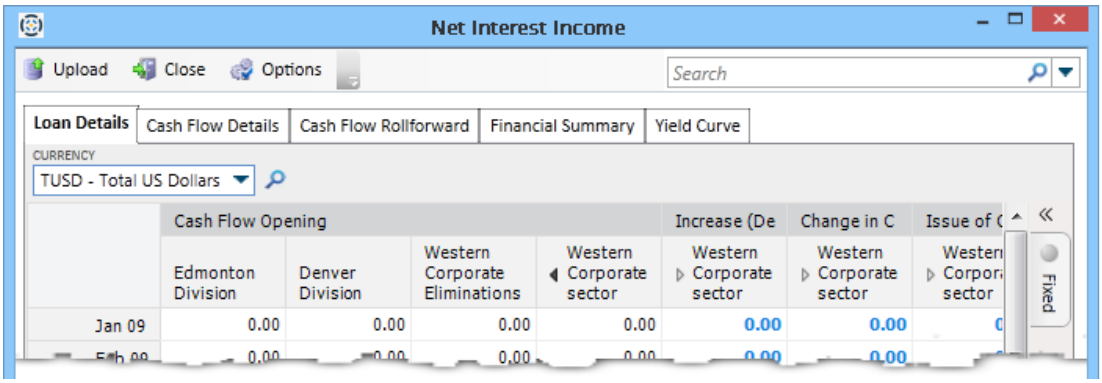
Example:

```
Slice Entities, Toronto
```

SymbolWidth

Use this function to specify the column widths for specific symbols in the innermost dimension in the columns orientation and optionally wrap text in title cells. If you use the SymbolWidth function for any dimension other than the innermost across dimension, the width of columns remains unaltered.





Syntax:

```
SymbolWidth Dimension, SymbolSpec, WidthSpecification[, WrapTitle:ON|OFF]
```

where:

- Dimension is the name of the innermost dimension in the columns orientation that contains the symbol to which the width applies.
- SymbolSpec contains the name of the symbols to apply the width to, with any valid hierarchical syntax.
- WidthSpecification is one of the following:

Value	Description
FitToTitle	Automatically adjusts the width of each column individually to fit the text of the title cell for the specified symbol or symbols.
Width	Specifies the column width of the title cells in pixels for the specified symbol or symbols. If you specify Width as 0, the default column width is 200 pixels.

- WrapTitle is optional and can be one of the following:

Note: WrapTitle applies to titles only, not data.

Value	Description
ON	Wraps text to the next line in title cells, if necessary, in the specified dimension. If both FitToTitle and WrapTitle are specified, WrapTitle behavior is not noticeable until users manually adjust the width of columns.
OFF	Does not wrap text to the next line in title cells. This is the default if WrapTitle is not specified. If WrapTitle is OFF and the text in title cells is longer than the width of the title cell, the text is truncated.

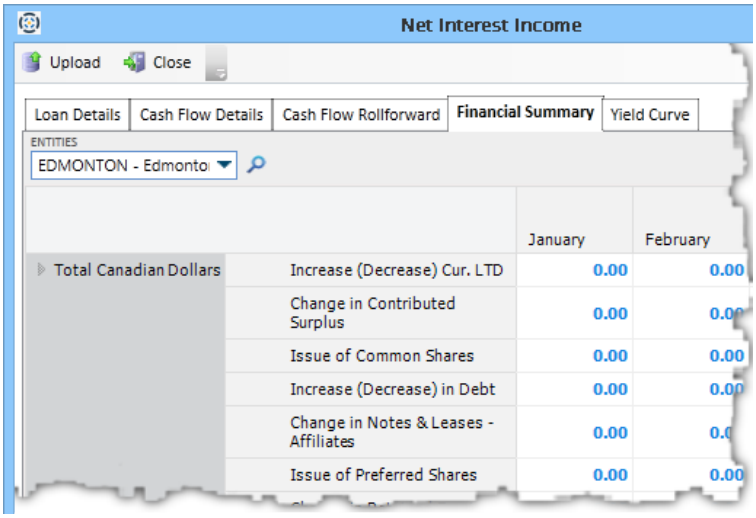
Note: The SymbolWidth function applies only to the innermost across dimension; outer dimension symbol widths are dictated by the width of the innermost dimension symbols.

Example:

```
SymbolWidth Entities, CORPW#99, FitToTitle

SymbolWidth Entities, CORPW#99, 80, WrapTitle:ON
```

TitleWidth

Use this function to specify the width of titles for a dimension in the rows orientation and optionally wrap text in title cells.



Syntax:

```
TitleWidth Dimension, WidthSpecification[, WrapTitle:ON|OFF]
```

where:

- Dimension is the name of a dimension in the rows orientation to which the width of the title applies.

- WidthSpecification is one of the following:

Value	Description
FitToTitle	Automatically adjusts the width of the row title column to fit the widest title text for the specified dimension.
Width	Specifies the column width of the row title cells in pixels for the specified dimension. If you specify Width as 0 or do not use the TitleWidth function in your Data View definition file, the default column width is 200 pixels.

- WrapTitle is optional and can be one of the following:

Value	Description
ON	Wraps text to the next line in title cells, if necessary, in the specified dimension. If both FitToTitle and WrapTitle are specified, WrapTitle behavior is not noticeable until users manually adjust the width of columns.
OFF	Does not wrap text to the next line in title cells. This is the default if WrapTitle is not specified. If WrapTitle is OFF and the text in title cells is longer than the width of the title cell, the text is truncated.

Note: WrapTitle applies to titles only, not data.

Example:

```
TitleWidth Currency, FitToTitle

TitleWidth Accounts, 200, WrapTitle:ON
```

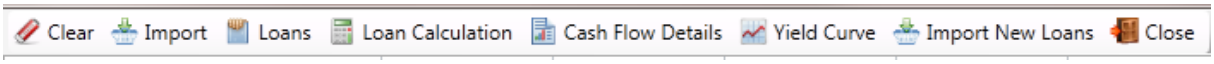
Related functions

- ColumnWidth
- SymbolWidth

Adding Actions To Data Grids

To make it easy for users to access certain functionality, you can create a toolbar or add a context menu to a cell within your Data Grid. You can also add inactive buttons to Data Grid toolbars, allowing for a unified look between tabs.

For more information, see the "Command" parameter in Syntax.



Action

Use this function to add a toolbar icon or a context menu with an associated action in a Data Grid. For example, you can use actions to run a procedure, open a new Data Grid, or run a model.

When a user right-clicks a cell or clicks a toolbar button, an action is triggered and specific context variables are populated. The following table lists the variables that are associated with actions in a Data Grid.

Variable	Description
C_CONTEXT	If a user clicks a cell in the Data Grid, the return value for this variable is CELL. If they click a toolbar action, the return value is DATAGRID. For example, $C_CONTEXT$ resolves to CELL or $C_CONTEXT$ resolves to DATAGRID.
C_ DATAAREA	This variable returns the name of the data area. For example $C_DATAAREA$ resolves to SampleDA.
C_ CELLVALUE	This variable returns the value of the intersection that the user right-clicked. This variable is not populated for toolbar actions. For example, $C_CELLVALUE$ resolves to 200.
C_ROWS	This variable returns the dimension in the rows orientation of the Data Grid. If the Data Grid has nested dimensions in the rows orientation, the variable returns the dimensions in order from outer to inner. For example, C_ROWS resolves to ENTITIES, ACCOUNTS.
C_ COLUMNS	This variable returns the dimension in the columns orientation of the Data Grid. If the Data Grid has nested dimensions in the columns orientation, the variable returns the dimensions in order from outer to inner. For example, $C_COLUMNS$ resolves to TIMEPER.

Variable	Description
C_Dimension	This variable returns the name of the symbol for each dimension at the intersection that the user right-clicked. For toolbar actions, these variables are populated only for dimensions that are not in the rows or columns orientation. For example, $C_CURRENCIES$ resolves to "CAD" or $C_ENTITIES$ resolves to "TORONTO". The names of variables in Longview Application Framework to be used in these context variables are limited to 31 characters. The convention is C_Dimension (where Dimension is the first 26 characters of the dimension name). In the case where the first 26 characters of the dimension names are the same, the variables are named C_Dimension_N (where Dimension is the first 26 characters of the dimension, and N is the dimension number).

You can reference any of the above context variables in Longview Application Framework documents.

Syntax

```
Action "Text", "Image", "Command", "Helptext"[, Location[,
"cellTypeExpression"]]
```

where:

- Text is the text to display in the toolbar. To display an image-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.
- Image is the path to the image to display as the toolbar or context menu icon. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. To display a text-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.

  Note: Longview recommends using 16 x 16 pixel images to optimize the appearance of your Data Grid toolbar.

- Command is the Longview Application Framework command to execute when a user clicks the toolbar icon or context menu item. To display a disabled icon, omit this parameter and use two double quotation marks (""). You can embed the Launch commands in the Command parameter.
  For more information, see Launch App, Launch Component, Launch ReportViewer and Launch URL.
- Helptext is the tooltip text that appears when a user hovers over the toolbar icon. To omit tooltip text, use two double quotation marks ("").
- Location is optional and defines the location of the action in the user interface. To add multiple locations, separate the parameters with a pipe (|). If you do not specify a location, the action item is added to the toolbar. Location can be any of the following:

Parameter	Description
TOOLBAR	Adds the action to the Data Grid toolbar.
CELLCONTEXT	Adds the action as part of the context menu for a cell.

- cellTypeExpression can be specified only if Location is also specified. This parameter is optional and is an expression that resolves to TRUE or FALSE and, when the expression resolves to TRUE for a cell, makes the context menu available. Cell type expressions can use NOT, AND, and OR with any of the following cell types:

Cell type	Description
PARENT	Applies the context menu action to all relevant parent cells.
LEAF	Applies the context menu action to all relevant leaf cells.
STATIC	Applies the context menu action to all relevant static cells.
WRITABLE	Applies the context menu action to all relevant writable cells.
READONLY	Applies the context menu action to all relevant read-only cells.

For example, to enable a context menu action on static parents only, the expression is PARENT AND STATIC. A cell type expression is optional. If you do not specify an expression, the context menu action is available for all cells.

Note: If you require double quotation marks (“) in the parameters for the Action, precede them with a backslash.

```
Action "Clear", "Common/images/toolbar/Eraser.png", "Run Procedure NII\NII_
View_Zero_CashFlow.lvpro", "Clear Cashflow", TOOLBAR


Action "Calculation", "Common/images/toolbar/calculator.gif", "Run Procedure
\"NII\NII_View_CDCalc.lvpro\"", "Deposit Calculator", CELLCONTEXT, "LEAF"

Action "", "Images/copy.ico", "Run"

Action “Open”, “Common/images/toolbar/Open.png”, “Launch App
\"HelloWorldApp\"”, “Open App”, TOOLBAR
```

DynamicModel

Use this function to specify a model to run when a user enters data manually or uses a drop-down list in a Data Grid.

Note: Models run before any data rolls up.

Before the model runs, the following variables are populated and are available for use in the model:

- C_CONTEXT
- C_DATAAREA
- C_CELLVALUE
- C_ROWS
- C_COLUMNS
- C_DIMENSION

For more information on these variables, see Working with context variables.

Syntax:

```
DynamicModel "Model"
```

where:

- Model is the model to run, enclosed in double quotation marks. The model must exist in the ...\DataServers\<LID>\applications directory, where LID is the Longview Identifier for your system. You can specify multiple DynamicModel statements. Models run in the order in which they appear in the Data View definition (.lvdvw) file.

Example:

```
DynamicModel "RecalcForecast.lvmod"
```

SystemAction

Use this function to add a button with a predefined system action to the toolbar. For example, you can use system actions to include toolbar buttons that allow users to export to Microsoft Excel, specify whether they view symbol names, descriptions, or an attribute value in a Data Grid, preview printing options, or close the Data Grid.

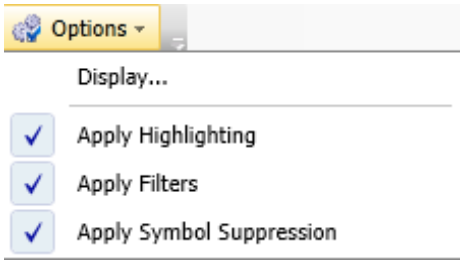
Syntax:

```
SystemAction Action, "Text", "Image", "Helptext"
```

where:

- Action is the system action to add to the toolbar and can be one of the following values:

Value	Description
ExporttoExcel	Adds a button that allows users to export the current view to Microsoft Excel.
PrintPreview	Adds a button that allows users to open a Print Preview dialog.
Close	Adds a button that allows users to signal that they are finished with the current Data Grid. This action does not trigger OnClose behavior, but is equivalent to using the SHOW RETURN command. For more information, see OnClose and Show Return.
Options	Adds a button that allows users to access the following:



Value	Description
Display dialog box	Allows users to specify whether to view symbol names, descriptions, and/or a symbol's attribute value for each dimension. You can specify the default selection for display options using the "DimensionTitles" function. For more information, see DimensionTitles.
Apply Highlighting toggle	Allows users to remove or apply highlighting, defined by the "ConditionalCellStyle" function. For more information, see ConditionalCellStyle.
Apply Filters toggle	Allows users to remove or apply filters, defined by the Filter function. For more information, see Filter.
Apply Symbol Suppression toggle	Allows users to remove or apply symbol suppression, defined by the SymbolSuppress function. For more information, see SymbolSuppress. The Apply Symbol Suppression toggle is available to users only if you use the SymbolSuppress function on the innermost down and/or across dimensions in your Data View definition file.

- Text is the text to display on the toolbar button. If you want to use the default text, use two double quotation marks (""). For ExporttoExcel, the default text is Export to Excel, and for Option, the default text is Options. For PrintPreview, the default text is Print Preview.

- Image is the filename of the to display on the toolbar button. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. If you want to use a default icon, use two double quotation marks ("").

- Helptext is the tooltip text that appears when a user hovers over the toolbar button. If you do not want tooltip text, use two double quotation marks ("").

Example:

```
SystemAction ExporttoExcel, "Export",
 "Common/images/toolbar/Excel.png", "Export current view to Excel"

SystemAction Options, "", "", "Change display options"

SystemAction Close, "","",""
```

Specifying the order of toolbar actions

You can specify the order in which you want actions to appear on the toolbar by controlling the order of the corresponding Action and SystemAction functions in your Data View file. If you want to, for example, always have your custom Close button as the last icon on the toolbar, place the Close Action at the bottom of the list of functions.

Example:

```
Action "Instructions", "users.ico", "RUN PROCEDURE ShowGreeting.lvpro",
"Show Instructions"

Action "Import Data", "import.gif", "RUN PROCEDURE ImportEmps.lvpro",
"Import Existing Salaries"

SystemAction ExporttoExcel, "", "", ""

SystemAction PrintPreview, "", "", ""

Action "Close", "Common/images/toolbar/exit.gif", "Show RETURN", "Close"
```

The above example displays the Close button as the last icon on the toolbar.

Specifying The Order Of Toolbar Actions

You can specify the order in which you want actions to appear on the toolbar by controlling the order of the corresponding Action and SystemAction functions in your Data View file. If you want to, for example, always have your custom Close button as the last icon on the toolbar, place the Close Action at the bottom of the list of functions.

Example:

```
Action "Instructions", "users.ico", "RUN PROCEDURE ShowGreeting.lvpro",
"Show Instructions"

Action "Import Data", "import.gif", "RUN PROCEDURE ImportEmps.lvpro",
"Import Existing Salaries"

SystemAction ExporttoExcel, "", "", ""

SystemAction PrintPreview, "", "", ""

Action "Close", "Common/images/toolbar/exit.gif", "Show RETURN", "Close"
```

The above example displays the Close button as the last icon on the toolbar.

For more information on using the Action and SystemAction functions, see Adding actions to Data Grids.

Specifying Title Bar Descriptions

You can use a data view function to customize the title bar description for Data Grids. If you do not include the WindowTitle function in your .lvdvw or .lvui file, the Data Grid opens with the title bar description specified by the description parameter in the Longview App configuration (.lvapp) file.

For more information, see “Configuring Longview Apps”.

Note: Title bar descriptions apply to standalone mode only.

WindowTitle

Use this function to customize the title bar description for Data Grids. For non-tabbed Data Grids, include this function in the .lvdvw file. If your Data Grid has tabs, you must specify this function in the .lvui file — window titles in the .lvdvw file do not apply to tabbed Data Grids.

Syntax:

```
WindowTitle "Title"
```

where:

- Title is the title bar description for a Data Grid, enclosed in double quotation marks.

Example:

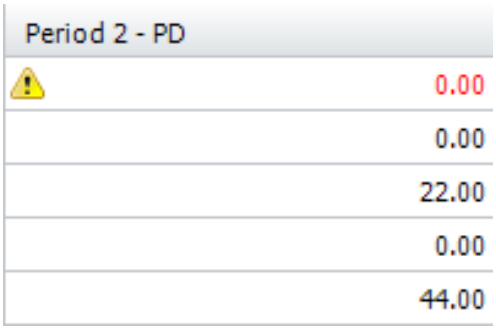
```
WindowTitle "Company Salary Planning"
```

Formatting Data In Data Grids

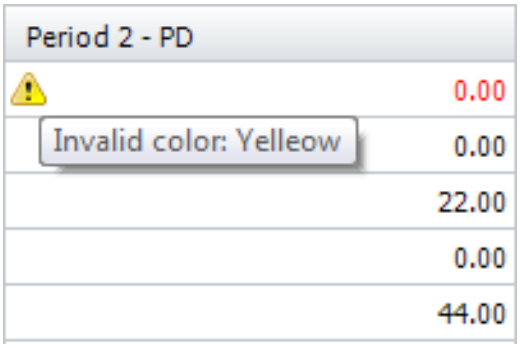
You can use the functions in this section to format your Data Grid to comply with locale-specific formatting standards or to provide data in a familiar format to your users.

ConditionalCellStyle

Use this function to apply a conditional style to an area of Data Grid cells. If you use an unsupported value for a style, a warning icon appears in the cell.



When you hover over the cell, a tooltip indicates the error.



Syntax:

```
ConditionalCellStyle Areaspec, Condition, Style1[, Style2]...
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension inherit the conditional style. You can have multiple conditional statements with overlapping area specifications; the order of the statements in the Data View definition file determines the priority from lowest to highest. For more information, see Working with area specs.

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension inherit the conditional style. You can have multiple conditional statements with overlapping area specifications; the order of the statements in the Data View definition file

determines the priority from lowest to highest. For more information, see Working with area specs.

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension inherit the conditional style. You can have multiple conditional statements with overlapping area specifications; the order of the statements in the Data View definition file determines the priority from lowest to highest. For more information, see Working with area specs.
- Condition is a condition statement. Use %v to indicate the cell value with AND and OR and the following arithmetic or text operators:

Arithmetic operator	Text operator	Definition
==	EQ	Equal to
!=	NE	Not equal to
>	GT	Greater than
>=	GE	Great than or equal to
<	LT	Less than
<=	LE	Less than or equal to

Note: Enclose string values with \" to apply a style to a string. For example, \"sample string\".

- Style1, Style2, etc can be any of the following:

Style	Description	Example
BACKGROUND	Specifies the cell background color to display. The values for this parameter can be system color names and are case sensitive. By default, the cell background color is white. For more information on system colors, refer to the MSDN web site section on colors by name.	BACKGROUND:Cyan
FOREGROUND	Specifies the text color for the cell contents. The values for this parameter can be system color names and are case sensitive. By default, the cell foreground color is black. For more information on system colors, refer to the MSDN web site section on colors by name. Because the text in protected cells in a Data Grid appears in blue, you should choose another color for conditional cell formatting to avoid any confusion to users. Using this setting overrides the default color for protected cells.	FOREGROUND:Gray

Example:

```
ConditionalCellStyle Trial_Balance;;;;;;;, "%v!=0", FOREGROUND:Red,
BACKGROUND:Yellow

ConditionalCellStyle SALEST#99;QAPYR#99;;;;;;;;;;;;;;, "%v==\"test3\"",
BACKGROUND:Yellow
```

Related functions

- Protect

DefaultDecimals

Use this function to specify the default number of decimals to display for cells containing numeric values in a Data Grid.

Note: The Data Grid displays decimals based on the following priority of functions, if specified: NumericInputOnly, SymbolDecimals, DefaultDecimals. For more information, see NumericInputOnly or SymbolDecimals.

Syntax:

```
DefaultDecimals Number
```

where:

- Number is the number of decimals to display. If you do not specify this parameter, the Data Grid displays two decimals.

Example:

```
DefaultDecimals 3
```

DimensionFreeze

Use this function to specify the number of columns in a dimension that should be frozen when scrolling horizontally. If a parent symbol is frozen, all of its children are frozen when it is expanded.

Note: If the Data Grid contains nested columns, the DimensionFreeze function is ignored.

Syntax:

```
DimensionFreeze Dimension, Number
```

where:

- Dimension is the dimension for which to freeze.
- Number is the number of columns in the specified dimension to freeze.

Example:

```
DimensionFreeze TIMEPER, 1
```

DimensionLabel

Use this function to replace generic dimension names in a Data Grid with terms more familiar to users. The label will appear in the following areas:

- Above Slice Dimensions
- In Symbol Selectors for Slice Dimensions
- In the Fixed Dimensions tab
- In the Line Item Details, Comments and Attachments dialog
- In the Print Preview for Data Grids

If you do not specify the dimension labels to use, the system displays the dimension name.

Syntax:

```
DimensionLabel Dimension, "Label"
```

where:

- Dimension is the dimension for which to specify the dimension label.
- Label is the label to use.

Example:

```
DimensionLabel ENTITIES, "COST CENTER"
```

DimensionTitles

Use this function to specify whether to display symbol names, descriptions, or a symbol's attribute value for a specified dimension in the Data Grid. If you do not specify the dimension titles to use, the system displays symbol descriptions. If you specify the name or description and the attribute value for a dimension, the name or description displays.

Syntax:

```
DimensionTitles Dimension, NAME|DESCRIPTION|ATTRIBUTE:attribute
```

where:

- Dimension is the dimension for which to specify dimension titles.
- NAME displays symbol names in the Data Grid.
- DESCRIPTION displays symbol descriptions in the Data Grid.
- ATTRIBUTE displays the value of the specified symbol attribute in the Data Grid.
- attribute is the name of the symbol attribute to use.

Note: You cannot specify NAME, DESCRIPTION, or ATTRIBUTE in the same command statement. To display symbol names and descriptions for a dimension, write separate statements for the dimension.

Example:

```
DimensionTitles ACCOUNTS, NAME

DimensionTitles TIMEPER, DESCRIPTION
DimensionTitles ENTITIES, DESCRIPTION
DimensionTitles PRODUCTS, ATTRIBUTE:AZReportDescription
```

DynamicRollup

Use this function to dynamically update the totals in the Data Grid as users enter data. If this function is ON or not specified, the parent values update automatically to reflect manual changes users make to the values in the Data Grid.

Syntax:

```
DynamicRollup ON|OFF
```

where:

- ON dynamically rolls up data in the grid as users make changes.
- OFF does not update parent data as users make changes to the grid until the users submit their changes and then refresh the grid.

Example:

```
DynamicRollup OFF
```

Filter

Use this function to filter out numeric or string values in a Data Grid based on a condition.

Syntax:

```
Filter Areaspec, Condition
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension that are part of the area spec are filtered if they meet the condition. For more information, see Working with area specs.
- Condition is a condition statement. Use %v to indicate the cell value with AND and OR and the following arithmetic or text operators:

Arithemtic operator	Text operator	Definition
==	EQ	Equal to
!=	NE	Not equal to
>	GT	Greater than
>=	GE	Great than or equal to
<	LT	Less than
<=	LE	Less than or equal to

Note: Enclose string values with \" to apply a style to a string. For example, \"sample string\".

Example:

```
Filter Trial_Balance###;;;;;;;, "%v==0"
Filter SALEST#99;QAPYR#99;;;;;;;;;;;;;;, "%v==\"test3\""
```

LIDInputOnly

Use this function to enforce line item details for the specified symbols in a Data Grid, and optionally, to prepopulate line item details comments. For example, you can prepopulate line item details comments for travel expenses with Airfare, Hotel, Car rental, and so on. For more information, see Working with line item details.

Caution: If you select symbols for line item details, make sure you select the related time period in the Across dimension. Selecting the wrong time period symbol could result in mismatched data. For more information, see Columns.

Syntax:

```
LIDInputOnly Areaspec[, "Comment1|Comment2|Comment3|...Comment20"]
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension enforce line item details. For more information, see Working with area specs.

  Note: For the time periods dimension, you must specify a symbol for which the ZGPLineItemDetailReference attribute is set.

- Comment1... Comment20 are optional and define the prepopulated line item details comments to appear in the Line Item Details dialog. Separate comment lines with a pipe (|). You can add up to 20 prepopulated lines of comments.

Example:

```
LIDInputOnly Travel;;;;;;;;, "Hotel|Airfare|Car rental"
```

NumericInputOnly

Use this function to restrict any input by users to numeric values for the specified data area. You can also restrict the number of decimals.

This restriction applies only when users manually input or copy/paste a value into the cell. If a model, procedure, or import process populates a cell with a numeric only restriction, the restriction is ignored.

If you designate a cell as numeric only and the current value is not a numeric value, a warning appears, but the data is still submitted to the database. All numeric input cells are right aligned in the Data Grid.

Note: The Data Grid displays decimals based on the following priority of functions, if specified: NumericInputOnly, SymbolDecimals, DefaultDecimals. For more information, see SymbolDecimals or DefaultDecimals.

Syntax:

```
NumericInputOnly Areaspec[, Decimals]
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. For more information, see Working with area specs.

- Decimals is the number of decimals to restrict numeric input to. This parameter is optional. If you do not specify this parameter, the system restricts decimal input based on the Number parameter of the DefaultDecimals function. For more information, see DefaultDecimals.

Example:

```
NumericInputOnly Sales1;;;;;;;;, 6
```

Parentheses

Use this function to specify whether or not negative numbers display with parentheses instead of a negative sign.

Syntax:

```
Parentheses ON|OFF
```

where:

- ON displays negative numbers with parentheses.
- OFF displays negative numbers with a negative sign. If you do not specify this parameter, the grid displays negative numbers with a negative sign.

Example:

```
Parentheses OFF
```

Protect

Use this function to prevent users from manually entering data to specific cells in a Data Grid. The contents of protected cells appear in blue in the Data Grid. If you are using conditional cell styles in the Data Grid, you should set the foreground color to something other than blue to avoid any confusion to users. If you specify both ConditionalCellStyle and Protect for a cell, the ConditionalCellStyle takes priority.

Note: Non-static parent cells are automatically protected.

Syntax:

```
Protect Areaspec
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. If you leave the symbol specification empty, all the symbols for that dimension that are part of the area spec are protected from manual input by users. For more information, see Working with area specs.

Example:

```
Protect sales6;;;;;;;

Protect sales1|sales2|sales5|cashT#99;Jan_YTD;Canada###;;CCAD|CUSD;;
```

Related functions

- CondtionalCellStyle

SymbolDecimals

Use this function to specify the number of decimals to display for a specific symbol. The priorities for decimals are (from highest to lowest priority):

- Inner row
- Outer row
- Inner column
- Outer column

Note: The Data Grid displays decimals based on the following priority of functions, if specified: NumericInputOnly, SymbolDecimals, DefaultDecimals. For more information, see NumericInputOnly or DefaultDecimals.

Syntax:

```
SymbolDecimals Dimension, Symbol, Decimals
```

where:

- Dimension is the dimension that contains the symbol to apply the number of decimals to.
- Symbol is the symbol to apply the number of decimals to.
- Decimals is the number of decimals to display.

Example:

```
SymbolDecimals ACCOUNTS, Balance_Sheet, 2
```

SymbolSuppress

Use this function to suppress symbols in the specified dimension that have values equal to zero. Suppression behavior applies only to the innermost dimension.

Syntax:

```
SymbolSuppress Dimension, Suppress
```

where:

- Dimension is the dimension name.
- Suppress is the symbol level to suppress and can be one of the following values:

Value	Description
LEAF	Suppresses any leaf symbols in the specified dimension that have a value of zero.
ALL	Suppresses all symbols in the specified dimension that have a value of zero.

Example:

```
SymbolSuppress ACCOUNTS, LEAF
```

TextInputOnly

Use this function to restrict any input by users to text for the specified data area. This restriction applies only when users manually input or copy/paste a value into the cell. If a model, procedure, or import process populates a cell with a text only restriction, the restriction is ignored.

If you designate a cell as text only and the current value is not a text value, a warning appears, but the data is still submitted to the database.

All text input cells are left aligned in the Data Grid.

Note: If a user enters a numeric value into a text input cell, the value is rendered and stored as text.

Syntax:

```
TextInputOnly Areaspec
```

where:

- Areaspec is a semicolon (;) delimited list in order of dimension that can contain a list of pipe (|) delimited symbol specifications. For more information, see Working with area specs.

Example:

```
TextInputOnly Employee;;;;;;;
```

ThousandsSeparator

Use this function to specify whether numeric values display with a locale-specific thousands separator.

Syntax:

```
ThousandsSeparator ON|OFF
```

where:

- ON displays a locale-specific thousands separator based on the operating system setting.
- OFF does not display a thousands separator. If this function is not specified, OFF is used.

Example:

```
ThousandsSeparator ON
```

Working With Drop-Down Lists

You can add drop-down lists to Data Grid cells to make it easier for users to submit the appropriate value and to reduce input error. Users cannot type in a cell that contains a drop-down list; however, they can paste values into the cell. Likewise, users can run a model that affects a cell with a drop-down list.

- Drop-down lists are not enforced in any of the following circumstances:
- line item details intersect with the drop-down list
- the cell has been protected using the Protect function
- the cell is unlocked or read-only

Before you can add a drop-down list, you must decide whether to use the default attribute or create a custom attribute.

For more information, see one of the following topics:

- Using the default drop-down list attribute
- Creating a custom drop-down list attribute

Using the default drop-down list attribute

By default, the system uses the ZValidValues attribute. This attribute already exists in your system. If ZValidValues contains a value and the relevant Data View does not specify a custom attribute using the ValidValuesAttribute function, the system uses this attribute by default for any drop-down lists for that symbol. For information on adding drop-down list values for this attribute, see Adding drop-down list values.

If you want to use your own custom drop-down list attribute, see Creating a custom drop-down list attribute.

Creating a custom drop-down list attribute

Longview allows you to create custom drop-down list attributes. If you use a custom attribute, you must also set the ValidValuesAttribute function in the relevant Data View definition. For more information, see ValidValuesAttribute.

To create a drop-down list attribute:

1. In Longview Application Administrator, click New, and select Attribute in the list. The New Attribute Definition dialog opens.
2. Complete these fields:

Field	Description
Class	Select SYMBOL.
Name	Specify a name for the drop-down list attribute.
Description	Specify a description for the attribute.

Field	Description
Type	Select List, and then select one of the following: ■ Date ■ String ■ Symbol

3. When you are finished, click OK.
4. Continue to “Adding drop-down list values”.

Adding drop-down list values

To add a drop-down list to a cell, you must set the attribute to the required values for the appropriate data intersection.

Caution: This feature can be used in either the across dimension or the down dimension of a Data Grid, but not both. Any intersection where both symbols include a drop-down list results in an error for that data intersection.

To set the drop-down list attribute:

1. In Longview Application Administrator, right-click the symbol to associate with a drop-down list, and select Properties on the context menu. The symbol Properties dialog opens.
2. Click the Attributes tab.
3. In the Attributes list, locate the ZValidValues attribute, or the attribute that you created for the drop-down list, and click in the Value column. For more information, see Using the default drop-down list attribute or Creating a custom drop-down list attribute.
4. Type the items to include in the drop-down list, separated by a pipe (|) symbol. Use a semicolon (;) within each list item to specify both a display value and a return value for that item. The string to the left of the semicolon is the display value and the string to the right is the return value. For more information, see “Example”.

   Note: Drop-down list items are case sensitive. Any input by a user as well as any references in Longview Application Framework documents must take this case sensitivity into account.

5. If you are using a custom attribute, specify the ValidValuesAttribute function in the required data view. For more information, see ValidValuesAttribute.

Example:

To add a drop-down list that allows users to select an employee type of Full Time, Part Time, or Contract, type Full Time|Part Time|Contract.

To add a drop-down list that displays Jan, Feb, and Mar, but returns the values p1, p2, and p3 respectively, type Jan;p1|Feb;p2|Mar;p3.

ValidValuesAttribute

After you create a custom drop-down list attribute, you must use the ValidValuesAttribute Data View function to specify the drop-down list attribute to associate with a Data Grid. You can specify only one ValidValuesAttribute function per Data View document. The system uses the ZValidValues attribute unless you specify a custom attribute using this function.

Syntax:

```
ValidValuesAttribute "AZAttribute"
```

where:

- Attribute is the appropriate Symbol attribute. For more information, see Creating a custom drop-down list attribute".

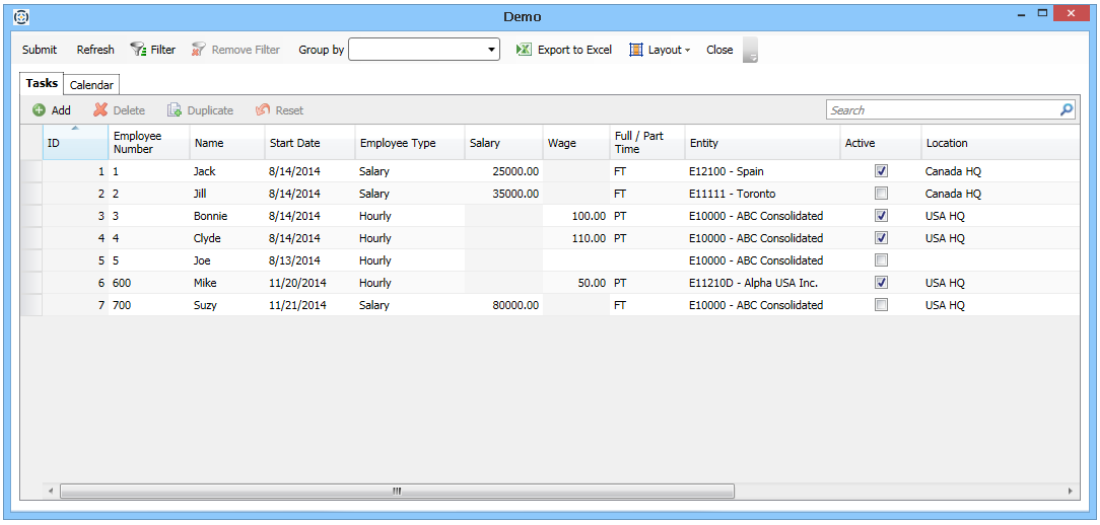
Example:

```
ValidValuesAttribute "AZDropDownAttribute"
```

Developing Longview Tables

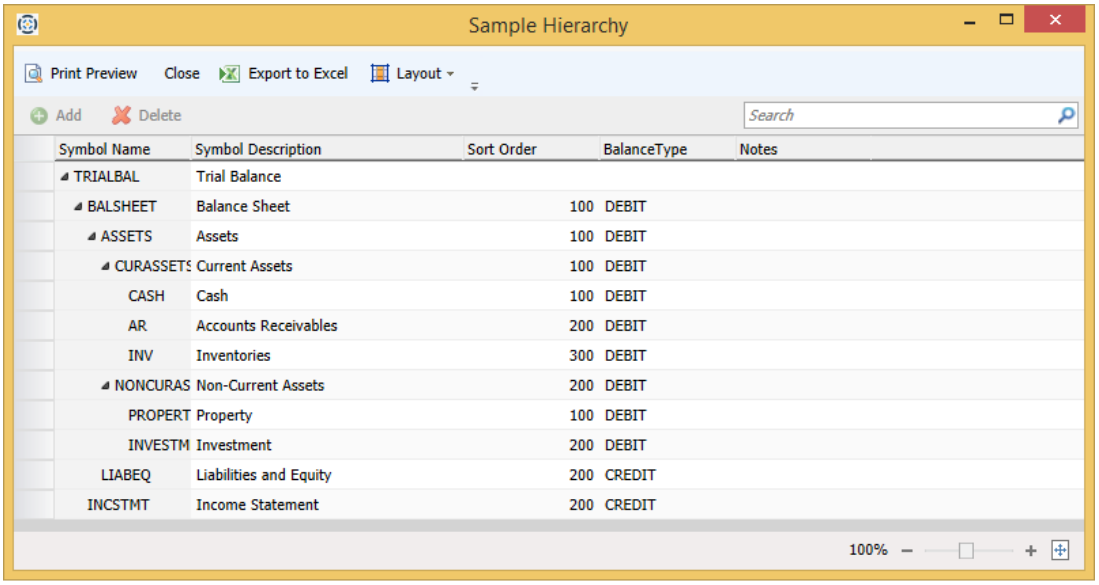
You can create interactive Tables that allow users to work with a flat list of data from the database. You can also generate Hierarchies to help users manage items that are related hierarchically. Additionally, you can display a Calendar that allows users to visualize items in calendar format.

For example, you could create a Table that displays salary information.



ID	Employee Number	Name	Start Date	Employee Type	Salary	Wage	Full / Part Time	Entity	Active	Location
1	1	Jack	8/14/2014	Salary	25000.00		FT	E12100 - Spain	☑	Canada HQ
2	2	Jill	8/14/2014	Salary	35000.00		FT	E11111 - Toronto	☐	Canada HQ
3	3	Bonnie	8/14/2014	Hourly		100.00	PT	E10000 - ABC Consolidated	☑	USA HQ
4	4	Clyde	8/14/2014	Hourly		110.00	PT	E10000 - ABC Consolidated	☑	USA HQ
5	5	Joe	8/13/2014	Hourly				E10000 - ABC Consolidated	☐	
6	600	Mike	11/20/2014	Hourly		50.00	PT	E11210D - Alpha USA Inc.	☑	USA HQ
7	700	Suzy	11/21/2014	Salary	80000.00		FT	E10000 - ABC Consolidated	☐	USA HQ

On the other hand, you could create a Hierarchy that displays symbol information.



Symbol Name	Symbol Description	Sort Order	BalanceType	Notes
TRIALBAL	Trial Balance			
BALSHEET	Balance Sheet	100	DEBIT	
ASSETS	Assets	100	DEBIT	
CURASSETS	Current Assets	100	DEBIT	
CASH	Cash	100	DEBIT	
AR	Accounts Receivables	200	DEBIT	
INV	Inventories	300	DEBIT	
NONCURAS	Non-Current Assets	200	DEBIT	
PROPERT	Property	100	DEBIT	
INVESTM	Investment	200	DEBIT	
LIABEQ	Liabilities and Equity	200	CREDIT	
INCSTMT	Income Statement	200	CREDIT	

Creating a Table, Hierarchy or Calendar consists of the following steps:

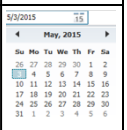
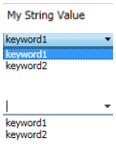
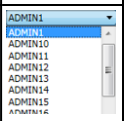
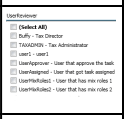
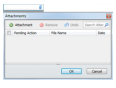
No.	Step...	Document/command to use
1	Create an App table in your database. For more information, see Creating App tables. **Note:** This step is not necessary if you plan to create a virtual DataTable in step 2.	
2	Define the DataTable object (for Tables, Hierarchies or Calendars) based on the App table or by defining a virtual DataTable (for Tables and Hierarchies only). For more information, see Defining DataTable objects.	Data Table definition (.lvdtd)
3	Create an in-memory DataTable object based on the DataTable definition using the Create DataTable command.	Procedure (.lvpro)
4	If you want to present a pre-populated Table, Hierarchy or Calendar to users, use the Download (for DataTables) command to populate the DataTable object with data from the database. Alternatively, you can use an ImportSpec or the Insert DataTableRow command.	Procedure (.lvpro) ImportSpec (.lvimp)
5	Define how the information in the DataTable object will render in the Table, Hierarchy or Calendar user interface. For more information, see Creating Table View definition files, see Creating Hierarchy View definition files, or Creating Calendar View definition files.	Table View definition (.lvtvw) Hierarchy View definition (.lvhvw) Calendar View definition (.lvcvw)
6	Use the Show DataTable command in the relevant procedure to display the DataTable object as a Table, Hierarchy or Calendar. Alternatively, you can use the Show UI command to include it in a tabbed App.	Procedure (.lvpro) App UI file (.lvui)
7	Test your Longview Table and make changes as needed. For more information, see Testing your Longview Table.	All supporting files (.lvapp, .lvpro, .lvdtd, .lvtvw, .lvhvw, .lvcvw, .lvui)
8	Deploy your Longview App. For more information, see Deploying Longview Apps.	All supporting files (.lvapp, .lvpro, .lvdtd, .lvtvw, .lvhvw, .lvcvw, .lvui)

Creating App Tables

App tables are the database tables in which data related to DataTables is stored. Unless you plan to create a virtual DataTable using the DataTable definition, you must create an App table to store information for Data Tables that is not related to your Longview multidimensional database. Currently, you must create these tables manually in the supported databases.

For more information, see the “Understanding meta tables related to App tables” section in the Longview Installation Guide.

The SQL or Oracle column data types that you choose when you create the table determine the Longview data type and, ultimately, the user controls in the user interfaces. For information on how the Longview data types correspond to SQL and Oracle data types, see the Longview Installation Guide.

User Control	Longview Data Type	Example
check box	Boolean	
calendar control	date	
text box (valid numbers only)	number	
text box, drop-down list, or a combo box (depending on whether the Values and AllowOverride keywords are used)	string	
drop-down list populated with users in the system	user	
multiple-selection list populated with users in the system	userlist	
drop-down list populated with the dimensions in the system	dimension	
Longview Symbol Selector for the dimension or linked dimension column specified by DATAVALUES (for more information on DATAVALUES syntax, see the Longview Installation Guide)	symbol	
button that opens the Longview File Attachments dialog	file	
read-only text box that the system automatically increments by 1 starting at the number specified for the DATAVALUES (for more information on DATAVALUES syntax, see the Longview Installation Guide)	autointeger	

User Control	Longview Data Type	Example
read-only text box with the username of the currently connected user	autouser	

Defining DataTable Objects

Before you can create a DataTable object using the Create DataTable command (or the VIRTUAL option), you must define the DataTable object in a DataTable definition. The DataTable definition defines the structure of the in-memory DataTable object. DataTable definitions are stored as ASCII files and can be created and edited in any text editor. The recommended file extension for DataTable definitions is .lvdtd.

The basic syntax for the DataTable definition document is:

```
DATATABLE AppTableName |VIRTUAL [,WRITABLE]
```

where:

AppTableName is the name of the App table to use to create the in-memory DataTable object. If you are using Oracle, the AppTableName must match the case that is in the Oracle database.

Note: If you do not specify any filtering options, the entirety of the AppTableName App table will be downloaded to the DataTable object, which may use an undesirable amount of memory. Longview recommends that you refine your DataTable object to include only the necessary elements of the App table using the relevant functions.

You can use the following functions to filter a DataTable definition:

- Columns (for DataTable definitions)
- RowLimit (for DataTable definitions)
- SymbolFilter
- UserFilter
- Where

VIRTUAL defines the DataTable object as a virtual DataTable object. Use this parameter when you want to define a DataTable object structure on-the-fly without creating an App table in your database. If you use this parameter, you must use the AddColumn function to define the appropriate columns.

WRITABLE is an optional parameter used in conjunction with VIRTUAL Tables to define the Table as writable.

Note: Longview Tables and Hierarchies that are based on a virtual DataTable are read-only when the WRITABLE option is not defined.

To view a sample DataTable definition, see Sample documents for Tables.

AddColumn

Use this function to define a column in a virtual DataTable. This function applies only to virtual DataTables.

Syntax:

```
AddColumn DataType, "ColumnName",
"Dimension:SymbolDimension""DimensionColumn:DimColumn" "Values:MyValues"
"Default:BooleanDefault"
```

where:

- DataType is the data type for the column, and can be one of the following:
  - boolean — Use this data type to add a column with a check box.
  - number — Use this data type to add a column with an editable text box that accepts numeric values.
  - string — Use this data type to add a column with an editable text box that accepts alphanumeric entries. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.
  - date — Use this data type to add a column with a calendar control.
  - dimension — Use this data type to add a column with the Longview Dimension Selector.
  - symbol — Use this data type to add a column with the Longview Symbol Selector. If you use this data type, you must specify the Values:MyValues keyword-value pair.
  - user — Use this data type to add a column with a combo box that is populated with users in the system. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.
  - userlist — Use this data type to add a column with a multiple-selection list that is populated with users in the system. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.

  > **Note:** Virtual DataTables without the WRITABLE option defined are read-only. In these cases, you cannot access interactive controls, such as the calendar control, symbol selector, drop-down user list, or multiple-selection user list when viewing the Table; however, data types are taken into consideration when importing data into a read-only virtual DataTable.

- ColumnName is the name for the new column.
- SymbolDimension applies only to symbol data types and is the dimension containing the symbols to include in the Symbol Selector. For symbol data types, specify either the Dimension keyword or the DimensionColumn keyword, but not both.
- DimensionColumn applies only to symbol data types and is the column containing the dimension which will control the symbols to include in the Symbol Selector. For symbol data types, specify either the Dimension keyword or the DimensionColumn keyword, but not both.

- MyValues syntax depends on the column's data type.

Column data type	Syntax
string	"ReturnValue1|ReturnValue2...|ReturnValueN[;DisplayValue1|DisplayValue2... |DisplayValueN]" where: ReturnValue1|ReturnValue2...|ReturnValueN are the values to include in the drop-down list as stored in the database. For example, FT. DisplayValue1|DisplayValue2...|DisplayValueN... are the values to display in the drop-down list. For example, Full Time. You can not specify a DisplayValue when AllowOverride is TRUE. If you do not specify a corresponding DisplayValue for a ReturnValue, the ReturnValue displays in the drop-down list.
symbol	Valid symbol specification. For more information, see Working with symbol specifications.
user	A pipe-delimited list of the usernames to include in the drop-down list.
userlist	A pipe-delimited list of the usernames or user groups to include in the multiple-selection list.

- BooleanDefault is the default value for a boolean column and can be either TRUE (selected) or FALSE (cleared).

Example:

```
DATATABLE VIRTUAL

ADDCOLUMN SYMBOL, "Dimension:ENTITIES", "Values:USNorthEast###"

ADDCOLUMN USER,
"UserName","Values:CChase|BSummers|WRosenberg|XHarris|RGiles"

ADDCOLUMN DATE, "StartDate"

ADDCOLUMN BOOLEAN, "Contract" "Default:FALSE"

ADDCOLUMN STRING, "EMAIL"

ADDCOLUMN DIMENSION, "MyDimensionColumn"
ADDCOLUMN SYMBOL, "MySymbolColumn", "DimensionColumn:MyDimensionColumn"
```

Columns (for DataTable definitions)

Use this DataTable function to specify a subset of the columns from the App table to include in the DataTable object. This function does not apply to virtual DataTables.

If you want to include all columns from the App table in the DataTable object, omit this function.

Note: You can further filter the columns displayed in a Table or Hierarchy using the Columns function. For more information, see Columns (for Table Views) and Columns (for Hierarchy Views).

Caution: If you restrict the columns and do not specify the column that is designated as the primary key, the entire Table or Hierarchy will be read-only. If you omit any non-nullable columns, the Add and Duplicate buttons will be disabled and users will not be able to add new rows.

Syntax:

```
Columns ColumnName1[, ColumnName2]...
```

where:

- ColumnName is the name of the App table column to include in the DataTable object. Separate multiple column names with a comma.

Note: The order of columns in this function determines the order that you must use for the String list variable in the Insert DataTableRow command.

Example:

```
Columns ID,EmployeeNumber,Name,Start_Date,EmployeeType,Salary,
Wage,FullPartTime,Entity,Active,Location,Manager,Attachments,CreatedBy
```

OrderBy (for DataTable definitions)

Use this DataTable function to order the rows retrieved from the App table. Ordering of the rows can be done by one or more columns in a table.

Syntax:

```
OrderBy "ColumnName1 [ASC|DESC][, ColumnName2 ASC|DESC, …]"
```

where:

- ColumnName is the column to sort by. If the column name contains spaces, you must enclose it in square brackets. For example, [Employee Type].

Note: OrderBy does not apply to columns of type Symbol, File, User or Userlist. Using OrderBy with these types of columns will return unexpected results.

Example:

```
OrderBy "[Employee Type] ASC"

OrderBy "Revenue DESC"
OrderBy "HireDate DESC, LastName ASC"
```

RowLimit (for DataTable definitions)

Use this DataTable function to restrict the number of rows retrieved from the App table to include in the DataTable object. This function does not apply to virtual DataTables. If you do not want to restrict the number of rows from the App table in the DataTable object, omit this function.

Note: You can use this function in conjunction with the OrderBy function to retrieve a meaningful but restricted subset of rows from the App table.

Syntax:

```
RowLimit numRows
```

where:

- numRows specifies the number of rows to retrieve from the App table into the DataTables object.

Example:

```
RowLimit 1000
```

SetColumn (for DataTable definitions)

Use this DataTable function to specify the options for columns in the DataTable object. Options depend on the data type for the column.

For more information on data types, see the Longview Installation Guide.

Note: Longview recommends that you include all necessary SetColumn parameters for each column in a single statement to avoid undesirable results.

Use the following table to determine whether a keyword in this function is appropriate for the column's data type.

Keyword	Column data types
Values	string
	symbol
	user
	userlist
Min Max	date
Default	boolean
AllowOverride	string

In addition, certain keywords determine the cell control that appears in the Table based on the column's data type. Use the following table to determine the keywords to use to generate the appropriate cell control.

Column data type	Keywords used	Resulting cell control
string	none	Text box
	Values	Drop-down list
	Values + AllowOverride:TRUE	Combo box
symbol	none	Drop-down list with all the symbols for the dimension or linked dimension column specified for the DATAVALUES column in the LV_APPTABLE_COLUMNS table.
	Values	Drop-down list with the symbols as restricted by the symbol spec specified for the Values keyword. **Note:** Specifying Values for symbol data type is only supported when specifying a dimension for the DATAVALUES
user	none	Drop-down list with all users in the system.
	Values	Drop-down list with the users as restricted by the Values keyword.
userlist	Values	Multiple-selection list populated with users as restricted by the Values keyword.

Syntax:

```
SetColumn "ColumnName", "Values:MyValues", "AllowOverride:TRUE|FALSE",
"Min:Date", "Max:Date", "Default:DefaultValue"
```

where:

- ColumnName is the name of the column for which to set properties in the DataTable object. For Oracle systems, this must be in the same case as in the Oracle database.
- MyValues syntax depends on the column's data type.

Column data type	Syntax
string	"ReturnValue1|ReturnValue2...|ReturnValueN[;DisplayValue1|DisplayValue2...|DisplayValueN]" where: ■ ReturnValue1|ReturnValue2...|ReturnValueN are the values to include in the drop-down list as stored in the database. For example, FT. ■ DisplayValue1|DisplayValue2...|DisplayValueN... are the values to display in the drop-down list. For example, Full Time. You can not specify a DisplayValue when AllowOverride is TRUE. If you do not specify a corresponding DisplayValue for a ReturnValue, the ReturnValue displays in the drop-down list.
symbol	Valid symbol specification. For more information, see Working with symbol specifications.
user	A pipe-delimited list of the usernames to include in the drop-down list.
userlist	A pipe-delimited list of the usernames or user groups to include in the multiple-selection list.

- AllowOverride can have one of the following values:
  - TRUE — Converts the drop-down list to a combo box, which allows users to type their own value in the cell control or select the values defined by MyValues.
  - FALSE — Restricts the cell control to a drop-down list, which allows users to select only the values defined by MyValues. This is the default.

> **Note:** You must specify AllowOverride in the same SetColumn statement as the Values keyword; otherwise, it produces a blank field.

- Date is the date range for a calendar control, defined using the Min and Max keywords. The format is yyyy-mm-dd. Date can be expressed only as equal to, and not greater than or less than.

> **Note:** You must specify Min and Max in the same SetColumn statement. If you use separate statements, only the last instance of either Min or Max is valid.

- DefaultValue is the default value for the column. Supported types are 1) boolean column and can be either TRUE (selected) or FALSE (cleared), and 2) dimension column, which can be any dimension name within the system. For boolean columns, Longview recommends that you

specify a default for NULLABLE boolean columns, otherwise, the value for new rows will be NULL, which may cause unpredictable behavior.

Example:

```
SetColumn "EmployeeType", "Values:FT;PT"


SetColumn "EmployeeType", "Values:FT;PT|$FT$;$PT$"

SetColumn "EmployeeType", "Values:FT;PT|Full Time;Part Time"

SetColumn "Method", "Values:Check/Cheque|Electronic Funds Transfer",
"AllowOverride:True"

SetColumn "Tax_Type", "Values:[[Symbol,AZTaskTaxTypes,
{{COLUMN,Jurisdiction}}]]"

SetColumn "Processed_Date", "Min:2015-01-01", "Max:2015-01-31"
```

SymbolFilter

Use this DataTable function to specify a subset of symbols from the App table to include in the DataTable object. This function does not apply to virtual DataTables and applies only to fields with the Symbol data type. If you do not specify a subset of symbols, all symbols from the App table are included in the DataTable object.

Syntax:

```
SymbolFilter ColumnName, "SymbolSpec1[|SymbolSpec2|…SymbolSpecN]"
[,"attributefilters"]
```

where:

- ColumnName is the column of type Symbol in the App table from which to filter the symbols.
- SymbolSpec specifies the subset of symbols from the App table to include in the DataTable object. You can specify multiple symbol specifications, separated by pipes (|). For information on symbol specifications, see Working with symbol specifications.
- attributefilters is optional and can include up to two attribute filters linked by AND or OR, enclosed in double quotation marks, using the syntax:

  ```
  FilterType{AttdrName{Operation{Expression
  ```

  where:

Field	Description
FilterType	specifies the method to use to search the hierarchy for symbols matching the filter criteria and can be one of the following: ■ ALL ■ PARENT ■ LEAF ■ ROOT If you specify two attribute filters, FilterType must be the same for both filters.
AttrName	is the name of the attribute for which to filter.
Operation	is one of the following: ■ EQ — Exactly equal to. ■ NE — Not equal to.
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For Non-List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

Example:

```
SymbolFilter entityName, "E1000###"
SymbolFilter entityName, "E1000###",
 "(ALL{ZGPNativeCurrency{EQ{CAD)"
SymbolFilter entityName, "E1000###|G1000###", "(ALL{ZGPNativeCurrency{EQ
{CAD) OR (ALL{ZGPNativeCurrency{EQ{USD)"
```

TempColumn

Use this function to define a temporary column in a persistent DataTable. This function applies only to persistent DataTables. You may find this feature useful if you would like to add a temporary column to display calculated values such as percentages or ratios that are not required to be stored in the database.

Syntax:

```
TempColumn DataType, "ColumnName", "Dimension:SymbolDimension",
"DimensionColumn:DimColumn","Values:MyValues", "Default:BooleanDefault"
```

where:

- DataType is the data type for the column, and can be one of the following:
  - boolean — Use this data type to add a temporary column with a check box.
  - number — Use this data type to add a temporary column with an editable text box that accepts numeric values.
  - string — Use this data type to add a temporary column with an editable text box that accepts alphanumeric entries. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.
  - date — Use this data type to add a temporary column with a calendar control.
  - dimension — Use this data type to add a temporary column with the Longview Dimension Selector.
  - symbol — Use this data type to add a temporary column with the Longview Symbol Selector. If you use this data type, you must specify the Values:MyValues keyword-value pair.
  - symbol — Use this data type to add a temporary column with the Longview Symbol Selector. If you use this data type, you must specify the Values:MyValues keyword-value pair.
  - user — Use this data type to add a temporary column with a combo box that is populated with users in the system. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.
  - userlist — Use this data type to add a temporary column with a multiple-selection list that is populated with users in the system. If you specify this value, you can optionally specify the Values:MyValues keyword-value pair.
- ColumnName is the name for the temporary column.
- SymbolDimension applies only to symbol data types and is the dimension containing the symbols to include in the Symbol Selector. For symbol data types, specify either the Dimension keyword or the DimensionColumn keyword, but not both.
- DimensionColumn applies only to symbol data types and is the column containing the dimension which will control the symbols to include in the Symbol Selector. For symbol data types, specify either the Dimension keyword or the DimensionColumn keyword, but not both.

- MyValues syntax depends on the column's data type.

Column data type	Syntax
string	"ReturnValue1|ReturnValue2...|ReturnValueN[;DisplayValue1|DisplayValue2... |DisplayValueN]" where: ■ ReturnValue1|ReturnValue2...|ReturnValueN are the values to include in the drop-down list as stored in the database. For example, FT. ■ DisplayValue1|DisplayValue2...|DisplayValueN... are the values to display in the drop-down list. For example, Full Time. You can not specify a DisplayValue when AllowOverride is TRUE. If you do not specify a corresponding DisplayValue for a ReturnValue, the ReturnValue displays in the drop-down list.
symbol	Valid symbol specification. For more information, see Working with symbol specifications.
user	A pipe-delimited list of the usernames to include in the drop-down list.
userlist	A pipe-delimited list of the usernames or user groups to include in the multiple-selection list.

- DefaultValue is the default value for the column. This parameter is valid for the following data types:
  - boolean, and the value can be either TRUE (selected) or FALSE (cleared)
  - dimension, and the value can be any dimension name within the system.

  For boolean columns, Longview recommends that you specify a default for NULLABLE boolean columns, otherwise, the value for new rows will be NULL, which may cause unpredictable behavior.

Example:

```
DATATABLE MyTable

TEMPCOLUMN NUMBER, "Percentage Increase"
TEMPCOLUMN STRING, "Favourite Color"
TEMPCOLUMN DIMENSION, "MyDimensionColumn"
TEMPCOLUMN SYMBOL, "MySymbolColumn", "DimensionColumn:MyDimensionColumn"
```

UserFilter

Use this DataTable function to specify a subset of users from the App table to include in the DataTable object. This function does not apply to virtual DataTables and applies only to fields with the user or

userlist data type. If you do not specify a subset of users, all users from the App table are included in the DataTable object.

Syntax:

```
UserFilter ColumnName, "UserName[|UserName2|…UserNameN]", "GroupName1
[|GroupName2|…GroupNameN]" [,"AttributeFilters"]
```

where:

- ColumnName is the column in the App table from which to filter the users. This column must have a data type of user or userlist.
- UserName is the name of the user subset to include in the DataTable object. Separate multiple usernames with a pipe (|). You can also filter for instances where the column is blank by using the NULL keyword.
- Group is the name of the group subset to include in the DataTable object. Separate multiple groups with a pipe (|).
- AttributeFilters is optional and can be up to two attribute filters linked by AND or OR, enclosed in double quotation marks, using the syntax:

  ```
  {AttrName{Operation{Expression
  ```

  where:

AttrName	is the name of the attribute for which to filter.
Operation	is one of the following: ■ EQ — Exactly equal to. ■ NE — Not equal to.

AttrName	is the name of the attribute for which to filter.
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For Non-List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

Example:

```
UserFilter ownerName, "Cordelia", ""
UserFilter ownerName, "", "GRP2"
UserFilter ownerName, "Cordelia|NULL", ""
UserFilter ownerName, "Xander", "GRP2"
UserFilter ownerName, "Cordelia|Xander|NULL", "GRP2" ,"(AUPlanner{EQ{TRUE)"
UserFilter ownerName, "Xander", "GRP2" ,"(AUPlanner{EQ{TRUE) AND
(AUInputUser{EQ{TRUE)"
```

Where

Use this DataTable function to define a subset of data to include in the DataTable object using conditions supported by the underlying database. This function does not apply to virtual DataTables. If you do not specify a subset of data, all data from the App table is included in the DataTable object.

Note: You must use the SymbolFilter or UserFilter functions to create a subset of data for symbols and users. For more information, see SymbolFilter and UserFilter.

Syntax:

```
Where "ConditionStatement [AND|OR ConditionStatement]"
```

where:

- ConditionStatement can be condition statements linked by AND or OR, enclosed in double quotation marks, using the following syntax:

```
ColumnName Operator Value
```

  where:
  - ColumnName is the name of the column to filter on.
  - Operator is one of the following:

Operator	Description	Additional Syntax
=	Equal to	
<>	Not equal to	
>	Greater than	
<	Less Than	
>=	Greater than or equal to	
<=	Less than or equal to	
BETWEEN	Specifies numeric values for a column within a specified range.	"Column BETWEEN x AND y"
LIKE	Specifies a search condition that must match a string of characters and wildcards (represented by percent signs).	"Column LIKE (%Value%)"
IN	Specifies multiple possible values for a column.	"Column IN ('Value1', 'Value2', 'Value3')"

- Value is the value for the condition. If you specify a column with the date data type, the Value syntax is based your database type:

Database	Syntax
Oracle	TO_DATE('DateValue,'DateFormat where: ■ DateValue is the date to filter on. ■ DateFormat is the format of the DateValue.
SQL Server	'yyyy/mm/dd'

  If you use column names that are reserved keywords for either SQL or Oracle, WHERE statements must use quotes. These quotes must then be escaped. For example, Where "\"number\" > 500".

Example:

```
Where "Year >= 2002 AND Year <=2012"
```

```
Where "Amount BETWEEN 150 AND 200"
Where "Title LIKE (%Manager%) OR Title LIKE (%Team Lead%)"
Where "Make IN ('Acura', 'Aston Martin', 'Chevrolet')"
Where "\"number\" > 500"
For Oracle: Where "StartDate > TO_DATE('2012/01/01','2012/01/01')"
For SQL: Where "StartDate > '2012/01/01'"
```

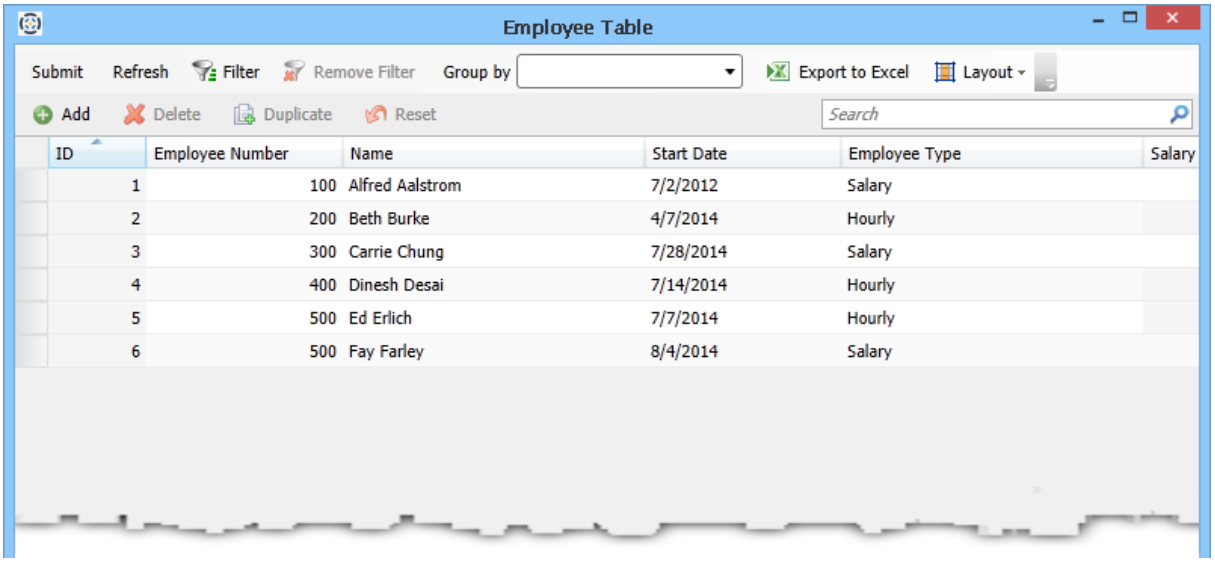
Creating Table View Definition Files

Table View definition files define the layout, format, and behavior of data in a Table. The Table View definition file is also referenced by the Show DataTable command used in a procedure file.

Table View definitions are stored as ASCII files and can be created and edited in any text editor. The recommended file extension for Table View definitions is .lvtvw.

> **Note:** The data displayed in a Table is determined by the DataTable object specified in the Show DataTable command. For more information, see Defining DataTable objects and Show DataTable

The following graphic shows a typical Table.



A Table, such as the one above, could be defined in a Table View definition similar to the following:

```
Columns ID,
EmployeeNuLongviewmber,Name,Start_Date,EmployeeType,Salary,Wage,
FullPartTime,Entity,Active,Location,Manager,Attachments,CreatedBy

Action "Submit", "Resources/Longview Icons/icons_ap/16x16/submit.png",
"Upload dtSalary", "", TOOLBAR

Action "Refresh", "Resources/Longview Icons/icons_ap/16x16/refresh.png",
"Download dtSalary", "", TOOLBAR

SystemAction Filter, "", "", ""

SystemAction GroupBy, "GroupBy", "", ""

SystemAction ExporttoExcel, "Export to Excel", "", ""

SystemAction ViewLayout, "Layout", "", ""
```

You can use the following functions in a Table View definition:

Action (for Table Views)	DefaultDecimals	SortOrder
AllowMultiPaste	DuplicateExclusions	SummaryRow
ApplyUserPreferences	Parentheses	SummaryRowLabel
ColorPicker	Protect	SystemAction (for Table Views)
Columns (for Table Views)	ReadOnly	ThousandsSeparator
ColumnDecimals	SetColumn (for Table Views)	VisibleRows
ColumnWidth	ShowToolbarButtons	

To view a sample Table View definition, see Sample documents for Tables.

Action (for Table Views)

Use this function to add a toolbar button or a context menu with an associated action in a Table

For example, you can use actions to run a procedure, open a new Table or Data Grid, or run a model.

Syntax:

```
Action "Text", "Image", "Command", "Helptext"[, Location[,
"ConditionalExpression"[, AND|OR "ConditionalExpression"]]]
```

where:

- Text is the text to display in the toolbar. To display an image-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.
- Image is the path to the image to display as the toolbar or context menu icon. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. To display a text-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.

  Note: Longview recommends using 16 x 16 pixel images to optimize the appearance of your Table toolbar.

- Command is the Application Framework command to execute when a user clicks the toolbar icon or context menu item. To display a disabled icon, omit this parameter and use two double quotation marks (""). For example, you can add a button that executes the Launch App command.
- Helptext is the tooltip text that appears when a user hovers over the toolbar icon. To omit tooltip text, use two double quotation marks ("").
- Location is optional and defines the location of the action in the user interface. To add multiple locations, separate the parameters with a pipe (|). If you do not specify a location, the action item is added to the toolbar. Location can be one of the following:

Parameter	Description
ROWCONTEXT	Add the action as part of the context menu for a row. **Note:** Changes applied as the result of an action using the ROWCONTEXT location are row-specific only. Using this option updates the specified row and does not refresh or reorder the entire table.
TOOLBAR	Adds the action to the Table toolbar.

- CondtionalExpression can be specified only if Location is also specified as ROWCONTEXT and must be enclosed in double quotation marks (""). This parameter is optional and is the expressions under which the context menu is available on the row header. You must enclose column tokens in escaped quotes (\"). ConditionalExpression uses the following syntax:

```
{{Column, ColumnName}} Operation Condition
```

  where:
  - ColumnName is the name of the column to evaluate for the expression.
  - Operation is one of the following operators:

Symbol	Meaning
NE or !=	is not equal to
EQ or ==	is exactly equal to
GT or >	is greater than
GE or >=	is greater than or equal to
LT or <	is less than
LE or <=	is less than or equal to

  - Condition is the condition to evaluate.

Note: If you require double quotation marks (“) in the parameters for the Action, precede them with a backslash

Example:

```
Action "Clear", "Common/images/toolbar/Eraser.png", "Run Procedure
ClearTable.lvpro", "Clear Table", TOOLBAR


Action "Expenses", "Resources/app images/new.png", "Launch App Expenses"
"Expenses", TOOLBAR

Action "Product Search", "MagnifyingGlass.jpg", "Run Procedure
LaunchProductPage.lvpro" "some tooltip text", ROWCONTEXT
```

```
Action "Update Balance (Active And Owner=TAXADMIN)", "Resources/Longview
Icons/icons_ap/16x16/flash_edit.png", "Run Procedure Update.lvpro" "Increase
balance by 10%", ROWCONTEXT, "{{Column, Active}}" AND \"{{Column, Owner}}\"
EQ \"TAXADMIN\""
```

AllowMultiPaste

Use this Table View function to specify whether a user can paste multiple cell values into contiguous cells within the Data Table.

If you set this function to OFF, users will be unable to paste multiple values into the Data Tables. You may wish to set this function to OFF in cases where you want to prevent this behavior because there are dynamic procedures defined that will not work well when modifying several values at once. For example, dynamic procedures that invoke user interfaces (messages, forms) or that run long calculations.

Note: Regardless of the AllowMultiPaste setting, there are some restrictions on when pasting is allowed. For example, users cannot paste into File type or Protected columns even if AllowMultiPaste is set to ON.

Syntax:

```
AllowMultiPaste ON|OFF
```

where:

- ON — Allows the end-user to paste multiple values into contiguous cells within the Data Table at runtime.
- OFF — Prevents the end-user from pasting multiple values into contiguous cells within the Data Table at runtime. The default is OFF if this function is not used.

Example:

```
AllowMultiPaste ON
```

ApplyUserPreferences

Use this Table View function to specify whether to display the user's last view when it is first opened (the view is automatically saved when the user closes the Table) or to always display the default view as defined in the Table View definition.

If you set this function to FALSE, Reset will be disabled in the Layout menu. If you do not use this function, the Table displays the user's last view when the user opens the Table.

Note: Specifying a SortOrder when applying user preferences may cause unpredictable sorting. If you specify a SortOrder, Longview recommends that you set ApplyUserPreferences to FALSE. If you do not specify a SortOrder and ApplyUserPreferences is FALSE, the columns are sorted based on the order in which they are returned from the database.

Syntax:

```
ApplyUserPreferences TRUE|FALSE
```

where:

- TRUE — Opens the Table with the user's last view. The default is TRUE if this function is not used.
- FALSE — Opens the Table with the default view as defined in the Table View definition.

Example:

```
ApplyUserPreferences FALSE
```

ColorPicker

Use this Table View function to add a color picker control that the user can invoke to edit a cell. The return value will be the HEX color code for the selected color.

Note: ColorPicker can only be specified for columns of type STRING.

Syntax:

```
ColorPicker ColumnName1[, ColumnName2... ] [,ShowText:TRUE|FALSE]
```

where:

- ColumnName is the name of the DataTable column that will contain the color picker control.
- ShowText specifies whether the HEX color code is displayed with the color indicator. By default, this option is TRUE.
  - TRUE displays the HEX color code beside the color indicator. This is the default if this option is not specified.
  - FALSE does not display the HEX color code beside the color indicator.

Example:

```
ColorPicker “FavoriteColor”
```

Example:

```
ColorPicker “FavoriteColor”, ShowText:FALSE
```

Columns (for Table Views)

Use this Table View function to specify the columns in the DataTable object that you want to display in the Table, along with the order in which they display. If you do not use this function, all columns in the DataTable object display in the Table.

Note: For information on including columns in the DataTable object, see Columns (for DataTable definitions).

Caution: If you restrict the columns and do not specify the column that is designated as the primary key or any non-nullable columns, the Add, Delete, and Duplicate buttons will be disabled in the Table and users will not be able to add new rows.

Syntax:

```
Columns ColumnName1[, ColumnName2... ]
```

where:

- ColumnName is the name of the DataTable column to display in the Table. Separate multiple column names with a comma. Columns display in the Table in the order specified in this function.

Example:

```
Columns
ID,Name,EmployeeType,Salary,Wage,FullPartTime,Entity,Location,Manager,

Attachments
```

ColumnDecimals

Use this function to specify the number of decimals for numeric values in specific column(s). This function overrides the DefaultDecimals function for the specified columns in the Table. The default number of decimal places for numeric values in a column is 2.

Syntax:

```
ColumnDecimals DecimalPlaces, "ColumnName1[|ColumnName2... |ColumnN]"
```

where:

- DecimalPlaces is the number of decimal places to use for numeric values in the specified columns.
- ColumnName1[|ColumnName2... |ColumnN] are the columns for which to specify the number of decimal places.

Example:

```
ColumnDecimals 0, "EmployeeID|DivisionNumber|SalaryLevel|Overtime Hours"

ColumnDecimals 2, "Salary|Bonus|Commissions"
ColumnDecimals 1, "Percentage"
```

ColumnWidth

Use this function to specify the column width for all or for particular columns in the Table.

Syntax:

```
ColumnWidth Width, [ColumnName1|ColumnName2...|ColumnN]
```

where:

- Width is the width for the specified columns, in pixels. The minimum column width that will be displayed in the Table's user interface is 20 pixels. The default column width ranges from 100 pixels to 200 pixels depending on the data type of a column.
- ColumnName1, ColumnName2... are the names of the DataTable column for which to set the width. If you do not specify at least one column, the width applies to all columns in the Table. To specify different widths for different columns, use multiple ColumnWidth statements.

Example:

```
ColumnWidth 100, EmpNames|Salary
```

DefaultDecimals

Use this function to specify the default number of decimal places for numeric values in a Table. If you do not specify this function, the default number of decimal places for numeric values is 2.

If you want to override this value for a specific column, you can use the ColumnDecimals function.

Syntax:

```
DefaultDecimals DecimalPlaces
```

where:

- DecimalPlaces is the default number of decimal places for numeric values in the Table.

Example:

```
DefaultDecimals 0
```

DuplicateExclusions

Use this function to specify the columns to exclude from duplication when a user duplicates a row. Columns specified as excluded from duplication appear blank and require user input. This function does not apply to columns with file, autointeger, autouser, or boolean data types.

Syntax:

```
DuplicateExclusions ColumnName|...|ColumnName
```

where:

- ColumnName is a pipe-delimited list of the columns for which to exclude from duplication.

Example:

```
DuplicateExclusions EmployeeNumber|Name|StartDate
```

DynamicProcedure

Use this function to specify a procedure to run when a user enters data manually or edits data using a control within a Table. You may also specify a procedure to run when a user invokes a standard row operation such as Add, Duplicate, Delete, Reset, and ResetAll.

Before the procedure runs, the following variables are populated and are available for use in the procedure:

- C_CELLVALUE
- C_CELLVALUEPRIOR
- C_COLUMNINDEX
- C_ROWINDEX
- C_DATATABLE
- C_CONTEXT

For more information on these variables, see Working with context variables.

You can use the global variable LVG_TableRedraw to set the table redraw indicator after the procedure is run. For more information, see Working with global variables.

Syntax:

```
DYNAMICPROCEDURE Type, "Procedure" [, ColName1|ColName2|…|ColNameN][,
RunOnMultiPaste:TRUE|FALSE]
```

where:

- Type can be one of the following:

Field	Description
CELL	Used to define the procedure to run when a user manually inputs data or edits data using a control
ROW	Used to define the procedure to run when a user invokes one of the following standard row operations: Add, Duplicate, Delete, Reset.
TABLE	Used to define the procedure to run when a user invokes the standard row operation ResetAll.

- Procedure is the procedure to run, enclosed in double quotation marks. The procedure must exist. You can specify multiple DynamicProcedure statements. Procedures will run in the order in which they appear in the Table View definition (.lvtvw) file.

- ColName1, ColName2... are the names of the DataTable column for which the dynamic procedure applies. This parameter allows you to optionally restrict which columns trigger the specified dynamic procedure. If you do not specify at least one column, the dynamic procedure applies to all columns in the Table.

- RunOnMultiPaste specifies whether the dynamic procedure will run when multiple values are pasted into the Data Table. By default, this option is FALSE.
  - TRUE The dynamic procedure will run when multiple values are pasted into the Data Table.
  - FALSE The dynamic procedure will not run when multiple values are pasted into the Data Table.

Example:

```
DYNAMICPROCEDURE CELL, "RecalcForecast.lvpro"

DYNAMICPROCEDURE ROW, "CalculateTotalPrice.lvpro", UNITPRICE|VOLUME

DYNAMICPROCEDURE ROW, "CalculateTotal.lvpro"

DYNAMICPROCEDURE TABLE, "Validate.lvpro"

DYNAMICPROCEDURE CELL, "CalculateTotal.lvpro", PRICE|VOLUME,
RunOnMultiPaste:TRUE
```

EditProcedure

Use this DataTable function to add a custom procedure that the user can invoke to edit a cell. For example, you can invoke a form to prompt the user for input and calculate a value for the cell. To set the value back to the cell, you can use the SetDataTableCell function.

Note: EditProcedure can only be specified for columns of type STRING and NUMBER.

Syntax:

```
EditProcedure ColumnName, "Procedure" [,AllowManualInput:TRUE|FALSE]
```

where:

- ColumnName is the name of the DataTable column for which the procedure will be run when the user clicks on the edit control.
- Procedure is the procedure to run, enclosed in double quotation marks. The procedure must exist.
- AllowManualInput specifies whether the user can type in the textbox. By default, this option is FALSE.
  - TRUE User can type in the textbox or click a button to invoke the procedure
  - FALSE User cannot type in the textbox and must click the button to invoke the procedure

Example:

```
EditProcedure "Salary", "EditSalary.lvpro", AllowManualInput:TRUE
```

LeftTitleText

Use this function to include additional information in the top left section of your Table. Left title text appears just below the toolbar(s) of the Table. You can include attribute tokens and string variables in the text.

Syntax:

```
LeftTitleText "Text"
```

where:

- Text is the text to include, enclosed in double quotation marks. To include text on multiple lines, use \n as the line break character.

Note: If you use variables in Text, variables are updated with toolbar button actions and the left title text is refreshed.

Example:

```
LeftTitleText "Hello World\nThis is my left title text on two lines"
```

PageSize

Use this DataTable function to render the table in a paged format that displays a specified number of rows per page. Users use navigation controls in the table to traverse the pages. The use of the PageSize function is recommended for tables when there are a large number of rows to display. By rendering the table into separate pages, the UI will perform optimally.

Because PageSize displays a subset of the total rowset, the following functions have either modified behavior or do not work in conjunction with the PageSize keyword:

- The SummaryRows totals applies to the rowset in the current view only.
- The secondary toolbar actions Add, Duplicate, and Delete are disabled when using the PageSize keyword.
- The secondary toolbar action Reset All applies to the rowset in the current view only.
- The Filter toolbar action will filter according to the current filter settings. It is possible that as a user pages through the table, they may encounter a page with all records filtered. In order to reset the filter settings, use the Remove Filter system action and then invoke the Filter system action again.
- The toolbar action Save Layout will save the layout but not the current view displayed.
- The Print toolbar action will only print the rowset in the current view only.
- The Export to Excel toolbar action will only export the rowset in the current view only.
- After the Download Application Framework command is invoked, the application will redraw with the first page in view.
- After the Import Application Framework command is invoked, the application will redraw with the first page in view.
- DynamicProcedure and custom procedures in toolbar actions are allowed but the end-user will need to navigate to see results on other pages.
- As a Longview App Designer, you may need to set the LVG_TableRedraw flag appropriately in order for the end-user to view accurate results.

Syntax:

```
PageSize numRows
```

where:

- numRows specifies the number of rows to display in each page view.

Example:

```
PageSize 1000
```

Parentheses

Use this function to specify whether or not negative numbers display with parentheses instead of a negative sign. If you do not use this function, the negative numbers in the Table are preceded by a negative sign.

Syntax:

```
Parentheses ON|OFF
```

where:

- ON displays negative numbers enclosed in parentheses.
- OFF displays negative numbers preceded by a negative sign (-).

Example

```
Parentheses OFF
```

Protect

Use this Table View function to protect the specified columns in the Table from input by users. If a user adds a new row, the Protect function still applies to the specified columns in the newly-created row.

If you want the entire table to be read-only, use the ReadOnly function.

Note: If you use the Protect function and the EnabledCriteria keyword in the SetColumn function for the same column, Protect takes priority.

Syntax:

```
Protect ColumnName1[,ColumnName2,…]
```

where:

- ColumnName is the name of the Table column to protect from user input.

Example:

```
Protect ID

Protect ID,Name,EmployeeType
```

ReadOnly

Use this function to make an entire Table read-only. This feature can be useful when you want to present data to users in a format in which they can sort and manipulate columns, but not add or submit data.

If you want only certain columns to be read-only, use the Protect function.

Syntax:

```
ReadOnly TRUE|FALSE
```

where:

- TRUE makes the Table read-only.
- FALSE makes the Table editable.

Example:

```
ReadOnly TRUE
```

SetColumn (for Table Views)

Use this Table View function to specify display options for a column.

Syntax:

```
SetColumn "ColumnName ", "Description:My Description",
["EnabledCriteria:ConditionalExpression"]
```

where:

- ColumnName is the name of the DataTable column to display in the Table. Separate multiple column names with a comma.
- My Description is the column header text to display. If you do not specify a description, the ColumnName displays.
- ConditionalExpression is optional and is the conditional expression under which a column is enabled or disabled.

Note: This option does not apply to virtual DataTables.

If the result of the expression is not zero, the column is enabled. If the result of the expression is zero, the column is disabled. Conditional expressions can include conditional operators, attributes, and column tokens. You must enclose column tokens in escaped quotes (\").

Depending on the data type for the column, there are special considerations for the ConditionalExpression for EnabledCriteria.

Note: If you use the Protect function and the EnabledCriteria keyword in the SetColumn function for the same column, Protect takes priority.

Data type	Considerations for conditional expressions
autouser, user, and userlist	User values for expressions are case insensitive. For conditional expressions with Longview users, you must use the following syntax: LONGVIEW\username If you're comparing a user, you need to enclose the username in quotation marks. These quotation marks then need to be escaped using the backslash.
boolean	Conditional expressions must use 1 and 0 for TRUE/FALSE values, respectively.
date	Conditional expressions must use the following format for dates. "EnabledCriteria:\"{{Column,ColumnName}}\" EQ \"yyyy-mm-dd\hh:mm:ss"" You must include the time, in the format hh:mm:ss, for SQL databases. The time is optional (but unused) for Oracle databases. If you're comparing a Date, you need to enclose the Date in quotation marks. You must then escape these quotes using the backslash, as indicated in the syntax above.
file	EnabledCriteria is not supported for columns with this data type.
number	Conditional expressions must match the number in the database exactly.
string	String values for expressions are case sensitive. If you're comparing a string, you need to enclose the string in quotation marks. These quotation marks then need to be escaped using the backslash.
symbol	Symbol values for expressions are case insensitive. If you're comparing a symbol, you need to enclose the symbol in quotation marks. These quotation marks then need to be escaped using the backslash.

Example:

```
SetColumn "ID", "Description:Employee Number"


SetColumn "Notes", "EnabledCriteria:{{Column,Amount}}>0"

SetColumn "Reviewer", "Description:Reviewer", "EnabledCriteria:\"
{{Column,Workflow}}\"

SetColumn "Approver", "Description:Approver", "EnabledCriteria:\"
{{Column,Workflow}}\" != 1"

SetColumn "Amount", "EnabledCriteria:\"{{Column,Tax_Due}}\" EQ \"Payment\"
OR \"{{Column,Tax_Due}}\" EQ \"Refund\""
```

ShowToolbarButtons

Use this Table View function to specify the buttons to show in the Table's toolbar. If you do not specify this function, no buttons display in the Table's toolbar.

Syntax:

```
ShowToolbarButtons ButtonName|All
```

where:

- ButtonName can be one or more of the following, separated by a pipe (|):

Button name	Description
Add	Displays a button that allows users to add a new row to the Table.
Delete	Displays a button that allows users to delete the current row from the Table.
Duplicate	Displays a button that allows users to duplicate the current row in the Table.
Reset	Displays a button that allows users to reset the current row in the Table that contain an error.
ResetAll	Displays a button that allows users to reset all rows in the Table that contain an error.

- All displays all of the above buttons in the Table.

Example:

```
ShowToolbarButtons All

ShowToolbarButtons Add|Duplicate|Reset|ResetAll
```

SortOrder

Use this Table View function to sort items in one or more Table columns in ascending or descending order.

Note: Specifying a SortOrder when applying user preferences may cause unpredictable sorting. If you specify a SortOrder, Longview recommends that you set ApplyUserPreferences to FALSE. If you do not specify a SortOrder and ApplyUserPreferences is FALSE, the columns are sorted based on the order in which they are returned from the database.

Syntax:

```
SortOrder "ColumnName [ASC|DESC]", ["ColumnName2 [ASC|DESC]"...]
```

where:

- ColumnName is the name of the column to sort.
- ASC sorts items in the column in ascending order. This is the default sort order.
- DESC sorts items in the column in descending order.

Example:

```
SortOrder "Salary DESC", "EmployeeID DESC" ,"Employee Name ASC"
```

SummaryRow

Use this function to add a summary row to a Table. Summary rows can display information such as subtotals and grand totals.

Note: You can allow users to toggle the display of subtotal and grand total rows by including the Options System Action. For more information, see SystemAction (for Table Views).

Syntax:

```
SummaryRow Type, "Column1|Column2...|ColumnN"[, "Tooltip"]
```

where:

- Type is one of the following:
  - SUBTOTAL — Adds a row that calculates and displays the subtotal sum of all records for each of the specified columns grouped by the Group By functionality. If the user does not group records, the summary row appears below all records. This type applies only to columns with the NUMBER or BOOLEAN data types. For BOOLEAN data types, the summary is the count of TRUE values.
  - GRANDTOTAL — Adds a row below all records that calculates and displays the sum of all records for each of the specified columns. This type applies only to columns with the NUMBER or BOOLEAN data types. For BOOLEAN data types, the summary is the count of TRUE values.

Note: To include a label for the each summary row, use the SummaryRowLabel function.

Example:

```
SummaryRow SUBTOTAL, "Units|Cost", "SubTotal"

SummaryRow GRANDTOTAL, "Units|Cost", "Grand Total"
```

SummaryRowLabel

Use this function to include a label in the left-most column for a summary row. If you include more than one SummaryRowLabel function per type, the last one displays in the Table as the summary row label. You must include the SummaryRow function for a summary row to display in the Table.

Syntax:

```
SummaryRowLabel Type, "Label"
```

where:

- Type is one of the following:
  - SUBTOTAL — includes a label for the subtotal row as specified by the SummaryRow function.
  - GRANDTOTAL — includes a label for the grandtotal row, as specified by the SummaryRow function.
- Label is the label for the row, enclosed in double quotation marks.

Note: If the left-most column displayed is being summed, the label does not display.

Example:

```
SummaryRowLabel SUBTOTAL, "Sub Total"
SummaryRowLabel GRANDTOTAL, "Grand Total"
```

SystemAction (for Table Views)

Use this Table View function to add a button with a predefined system action to the toolbar. For example, you can use system actions to include a toolbar button that allows users to export to Microsoft Excel or to add a button that allows users to specify whether they view symbol names, descriptions, or an attribute value in a Table.

Syntax:

```
SystemAction Action, "Text", "Image", "Helptext"
```

where:

- Action is the system action to add to the toolbar and can be one of the following values:
  - Close, which adds a button that users can click to close the current Table.
  - ExportToExcel, which adds a button that allows users to export the current view to Microsoft Excel.
  - Filter, which adds buttons that allows users to filter the current view based on criteria for each column and to remove all filters.
  - GroupBy, which adds a list that allows users to group the current view by the selected column.
  - Options — Adds a button that allows users to access the following options when the corresponding SummaryRow function is also included in the table view document:

Option	Description
Show Group Subtotals toggle	Allows users to show or hide group subtotal rows if defined by the SummaryRow function. For more information, see SummaryRow.
Show Grand Total toggle	Allows users to show or hide grand total rows if defined by the SummaryRow function. For more information, see SummaryRow.

> **Note:** If you use the Options SystemAction without including the SummaryRows function in your Table View document, the Options button does not display to the user.

- PrintPreview — Adds a button that allows users to open a Print Preview dialog.

> **Note:** Users' printing preferences are saved when the Table is closed.

- ViewLayout, which adds a button with drop-down options that allow the user to reload the most recently saved view layout (the layout is saved when the Table is closed) or reset the view to its original layout (as specified by the developer that created the Table).

- Text is the text to display on the button. This option does not apply to the Filter and GroupBy actions. If you want to omit this parameter and display the image only, use two double quotation marks (""). The following table lists the default text.

Keyword	Default text
Close	Close
ExportToExcel	Export to Excel
Options	Options
PrintPreview	Print Preview
ViewLayout	Layout

- Image is the image to display on the button. This option does not apply to the GroupBy, Filter, or Close actions. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. If you want to omit this parameter, the system uses a default icon. To omit this parameter, use two double quotation marks (""). The following table indicates the default images.

Keyword	Default image
ExportToExcel	
Options	
PrintPreview	
ViewLayout	

- Helptext is the tooltip text that appears when a user hovers over the button. This option does not apply for the GroupBy action. There is no default for this parameter.

Example:

```
SystemAction ExporttoExcel, "Export", "Resources/images/toolbar/Excel.png",
"Export the current view to Excel"

SystemAction ExporttoExcel, "", "", "Export the current view to Excel"
SystemAction GroupBy "", "", ""
SystemAction Filter "", "", ""
SystemAction PrintPreview "", "", "Print Preview"
SystemAction Options "", "", ""
```

ThousandsSeparator

Use this function to specify whether numeric values display with a locale-specific thousands separator. For example, if your location setting is “United States”, numeric values display with commas (,) as the thousands separator.

Syntax:

```
ThousandsSeparator ON|OFF
```

where:

- ON displays a locale-specific thousands separator based on the operating system setting.
- OFF does not display a thousands separator. If this function is not specified, OFF is used.

Example:

```
ThousandsSeparator ON
```

ValuesPane

Use this Table View function to present the columns of the active row in a pane where the columns are transposed (presented in rows). The values pane is ideal for Tables containing numerous columns which would require users to scroll horizontally in order to enter row details.

If you do not use this function, the Table displays with the values pane feature disabled. If the values pane feature is ON for the Table, but the pane is not present, double-click the row selector to open the values pane.

Syntax:

```
ValuesPane ON|OFF [, "Title:MyTitle"] [, "HideOnLoad:TRUE|FALSE" ]
```

where:

- ON will enable the values pane feature.
- OFF will disable the values pane feature.
- MyTitle is the text to display in the pane's title bar. The default title is "Values Pane".
- HideOnLoad define whether the values pane is visible when the Table is initially displayed. This option can have one of the following values:

Value	Description
TRUE	Opens the Table with the values pane in view.
FALSE	The Table is opened without the values pane in view. This is the default value. The user must double-click on the row selector to open the values pane if required.

Example:

```
ValuesPane ON, "Title:TaskDetails", "HideOnLoad:FALSE"

ValuesPane ON, "Title:Properties", "HideOnLoad:FALSE"
ValuesPane ON, "Title:TaskDetails", "HideOnLoad:TRUE"
ValuesPane ON
ValuesPane OFF
```

VisibleRows

Use this function to specify the maximum number of DataTable rows to display in the Table. For example, you can use this function, with the SortOrder function, to create a Top 10 report.

Syntax:

```
VisibleRows NumOfRows
```

where:

- NumOfRows is the maximum number of DataTable rows to display in the Table.

Example:

```
VisibleRows 10
```

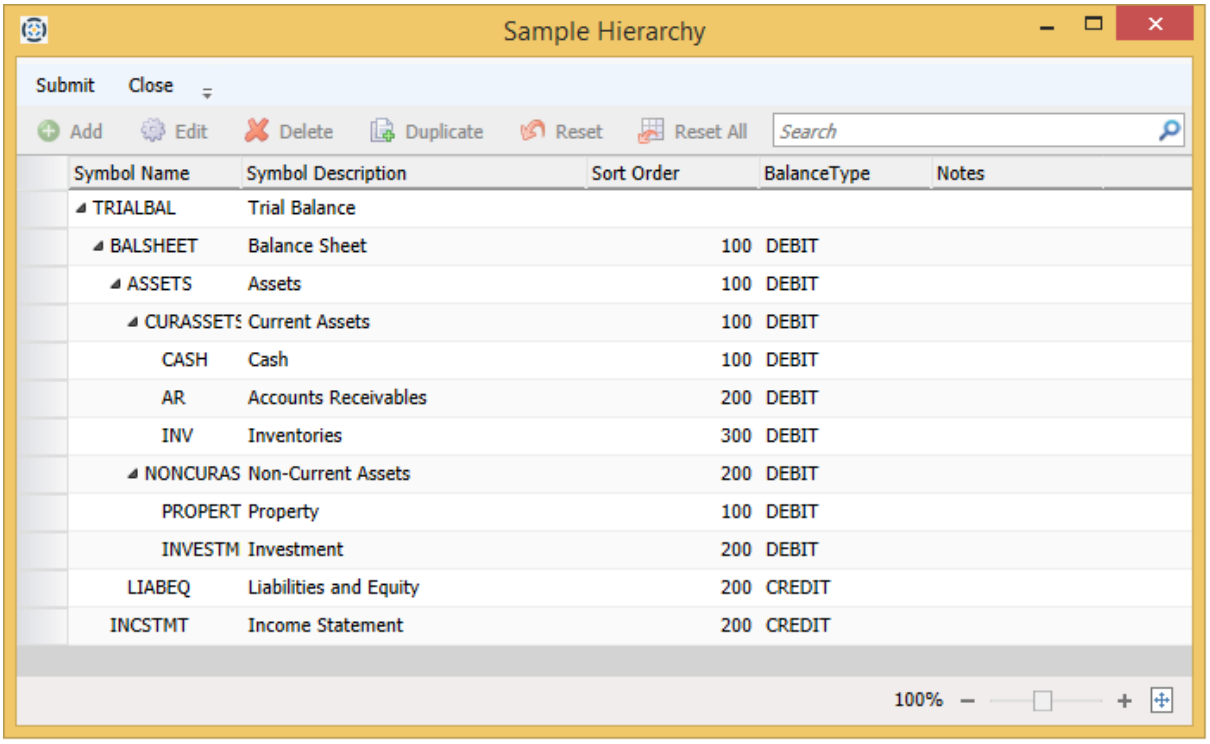
Creating Hierarchy View Definition Files

Hierarchy View definition files define the layout, format, and behavior of data in a Hierarchy. The Hierarchy View definition file is also referenced by the Show DataTable command used in a procedure file.

Hierarchy View definitions are stored as ASCII files and can be created and edited in any text editor. The recommended file extension for Hierarchy View definitions is .lvhvw.

Note: The data displayed in a Hierarchy is determined by the DataTable object specified in the Show DataTable command. For more information, see Defining DataTable objects and Show DataTable.

The following graphic shows a typical Hierarchy.



A Hierarchy, such as the one above, could be defined in a Hierarchy View definition similar to the following:

```
Columns SymName,SymDesc,Parent,SortOrder,BalanceType,Notes

NameColumn SymName

DescriptionColumn SymDesc

ParentColumn Parent

SortOrderColumn SortOrder
```

```
SetColumn "SymName", "Description:Symbol Name"
SetColumn "SymDesc", "Description:Symbol Description"
SetColumn "SortOrder", "Description:Sort Order"
ColumnWidth 100
ColumnWidth 200, SymDesc
DefaultDecimals 0
ShowToolbarButtons All
Action "Submit", "", "Upload dtSymbols", "Submit", TOOLBAR
SystemAction Close, "Close", "", ""
```

You can use the following functions in a Hierarchy View definition:

Action (for Hierarchy Views)	DynamicProcedure	SetColumn (for Hierarchy Views)
ApplyUserPreferences	EditProcedure	ShowToolbarButtons (for Hierarchy Views)
Columns (for Hierarchy Views)	Expand (for Hierarchy Views)	SortOrderColumn
ColumnDecimals	NameColumn	SystemAction (for Hierarchy Views)
ColumnWidth	ParentColumn	ThousandsSeparator
DefaultDecimals	Parentheses	ValuesPane
DescriptionColumn	Protect	
DuplicateExclusions	ReadOnly	

To view a sample Hierarchy View definition, see Sample documents for Tables.

Action (for Hierarchy Views)

Use this Hierarchy View function to add a toolbar button or a context menu with an associated action in a Hierarchy. For example, you can use actions to run a procedure, or open a new Table or Data Grid.

Syntax:

```
Action "Text", "Image", "Command", "Helptext"[, Location[,
"ConditionalExpression"[, AND|OR "ConditionalExpression"]]]
```

where:

- Text is the text to display in the toolbar. To display an image-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.
- Image is the path to the image to display as the toolbar or context menu icon. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. To display a text-only icon, omit this parameter and use two double quotation marks (""). You must specify

either Text or Image.

> **Note:** Longview recommends using 16 x 16 pixel images to optimize the appearance of your Table toolbar.

- Command is the Application Framework command to execute when a user clicks the toolbar icon or context menu item. To display a disabled icon, omit this parameter and use two double quotation marks (""). For example, you can add a button that executes the Launch App command.
- Helptext is the tooltip text that appears when a user hovers over the toolbar icon. To omit tooltip text, use two double quotation marks ("").
- Location is optional and defines the location of the action in the user interface. To add multiple locations, separate the parameters with a pipe (|). If you do not specify a location, the action item is added to the toolbar. Location can be one of the following:

Parameter	Description
ROWCONTEXT	Add the action as part of the context menu for a row. **Note:** Changes applied as the result of an action using the ROWCONTEXT location are row-specific only. Using this option updates the specified row and does not refresh or reorder the entire hierarchy.
TOOLBAR	Adds the action to the Hierarchy toolbar.

- CondtionalExpression can be specified only if Location is also specified as ROWCONTEXT and must be enclosed in double quotation marks (""). This parameter is optional and is the expressions under which the context menu is available on the row header. You must enclose column tokens in escaped quotes (\"). ConditionalExpression uses the following syntax:

  {{Column, ColumnName}} Operation Condition

  where:
  - ColumnName is the name of the column to evaluate for the expression.
  - Operation is one of the following operators:

Symbol	Meaning
NE or !=	is not equal to
EQ or ==	is exactly equal to
GT or >	is greater than
GE or >=	is greater than or equal to
LT or <	is less than
LE or <=	is less than or equal to

  - Condition is the condition to evaluate.

Note: If you require double quotation marks (“) in the parameters for the Action, precede them with a backslash.

Example:

```
Action "Clear", "Common/images/toolbar/Eraser.png", "Run Procedure
ClearHierarchy.lvpro", "Clear Hierarchy", TOOLBAR

Action "Product Search", "MagnifyingGlass.jpg", "Run Procedure
LaunchProductPage.lvpro" "some tooltip text", ROWCONTEXT

Action "Update Balance (Active And Owner=TAXADMIN)", "Resources/Longview
Icons/icons_ap/16x16/flash_edit.png", "Run Procedure Update.lvpro" "Increase
balance by 10%", ROWCONTEXT, "{{Column, Active}}" AND \"{{Column, Owner}}\"
EQ \"TAXADMIN\""
```

ApplyUserPreferences

Use this Hierarchy View function to specify whether to display the user’s last view when it is first opened (the view is automatically saved when the user closes the Hierarchy) or to always display the default view as defined in the Hierarchy View definition.

If you set this function to FALSE, Reset will be disabled in the Layout menu. If you do not use this function, the Hierarchy displays the user’s last view when the user opens the Hierarchy.

Syntax:

```
ApplyUserPreferences TRUE|FALSE
```

where:

- TRUE — Opens the Hierarchy with the user's last view. The default is TRUE if this function is not used.
- FALSE — Opens the Hierarchy with the default view as defined in the Hierarchy View definition.

Example:

```
ApplyUserPreferences FALSE
```

Columns (for Hierarchy Views)

Use this Hierarchy View function to specify the columns in the DataTable object that you want to display in the Hierarchy, in the order they are to be displayed. If you do not use this function, all columns in the DataTable object display in the Hierarchy.

Columns that are defined as NameColumn and ParentColumn must be included in this Columns function.

Note: For information on including columns in the DataTable object, see Columns (for DataTable definitions).

Caution: If you restrict the columns and do not specify the column that is designated as the primary key or any non-nullable columns, the Add, Edit, Delete, and Duplicate buttons will be disabled in the Hierarchy and users will not be able to add new rows.

Syntax:

```
Columns ColumnName1[, ColumnName2... ]
```

where:

- ColumnName is the name of the DataTable column to display in the Hiearchy. Separate multiple column names with a comma. Columns display in the Hierarchy in the order specified in this function.

Example:

```
Columns SymName,SymDesc,Parent,SortOrder,BalanceType,Notes
```

ColumnDecimals

Use this Hierarchy View function to specify the number of decimals for numeric values in specific column (s). This function overrides the DefaultDecimals function for the specified columns in the Hierarchy. The default number of decimal places for numeric values in a column is 2.

Syntax:

```
ColumnDecimals DecimalPlaces, "ColumnName1[|ColumnName2... |ColumnN]"
```

where:

- DecimalPlaces is the number of decimal places to use for numeric values in the specified columns.
- ColumnName1[|ColumnName2... |ColumnN] are the columns for which to specify the number of decimal places.

Example:

```
ColumnDecimals 0, "EmployeeID|DivisionNumber|SalaryLevel|Overtime Hours"

ColumnDecimals 2, "Salary|Bonus|Commissions"
ColumnDecimals 1, "Percentage"
```

ColumnWidth

Use this Hierarchy View function to specify the column width for all or for particular columns in the Hierarchy.

Syntax:

```
ColumnWidth Width, [ColumnName1|ColumnName2...|ColumnN]
```

where:

- Width is the width for the specified columns, in pixels. The minimum column width that will be displayed in the Hierarchy's user interface is 20 pixels. The default column width ranges from 100 pixels to 200 pixels depending on the data type of a column.
- ColumnName1, ColumnName2... are the names of the DataTable column for which to set the width. If you do not specify at least one column, the width applies to all columns in the Hierarchy. To specify different widths for different columns, use multiple ColumnWidth statements.

Example:

```
ColumnWidth 100

ColumnWidth 200, SymDesc
```

DefaultDecimals

Use this Hierarchy View function to specify the default number of decimal places for numeric values in a Hierarchy. If you do not specify this function, the default number of decimal places for numeric values is 2.

If you want to override this value for a specific column, you can use the ColumnDecimals function.

Syntax:

```
DefaultDecimals DecimalPlaces
```

where:

- DecimalPlaces is the default number of decimal places for numeric values in the Hierarchy.

Example:

```
DefaultDecimals 0
```

DescriptionColumn

Use this Hierarchy View function to specify the column that stores description of items in the Hierarchy.

Syntax:

```
DescriptionColumn ColumnName
```

where:

- ColumnName is the name of the DataTable column where the description of items can be found in a Hierarchy.

Example:

```
DescriptionColumn SymbolDescription
```

DuplicateExclusions

Use this Hierarchy View function to specify the columns to exclude from duplication when a user duplicates a row. Columns specified as excluded from duplication appear blank and require user input. This function does not apply to columns with file, autointeger, autouser, or boolean data types.

Syntax:

```
DuplicateExclusions ColumnName|...|ColumnName
```

where:

- ColumnName is a pipe-delimited list of the columns for which to exclude from duplication.

Example:

```
DuplicateExclusions EmployeeNumber|Name|StartDate
```

DynamicProcedure

Use this Hierarchy View function to specify a procedure to run when a user enters data manually or edits data using a control within a Hierarchy. You may also specify a procedure to run when a user invokes a standard row operation such as Add, Edit, Duplicate, Delete, Reset, and ResetAll.

Before the procedure runs, the following variables are populated and are available for use in the procedure:

- C_CELLVALUE
- C_CELLVALUEPRIOR
- C_COLUMNINDEX
- C_ROWINDEX
- C_DATATABLE
- C_CONTEXT

For more information on these variables, see "Working with context variables".

You can use the global variable LVG_TableRedraw to set the table redraw indicator after the procedure is run. For more information, see "Working with global variables".

Syntax:

```
DYNAMICPROCEDURE Type, "Procedure" [, ColName1|ColName2|…|ColNameN][,
RunOnMultiPaste:TRUE|FALSE]
```

where:

- Type can be one of the following:

Field	Description
CELL	Used to define the procedure to run when a user manually inputs data or edits data using a control
ROW	Used to define the procedure to run when a user invokes one of the following standard row operations: Add, Edit, Duplicate, Delete, Reset.
TABLE	Used to define the procedure to run when a user invokes the standard row operation ResetAll.

- Procedure is the procedure to run, enclosed in double quotation marks. The procedure must exist. You can specify multiple DynamicProcedure statements. Procedures will run in the order in which they appear in the Hierarchy View definition (.lvhvw) file.
- ColName1, ColName2... are the names of the DataTable column for which the dynamic procedure applies. This parameter allows you to optionally restrict which columns trigger the specified dynamic procedure. If you do not specify at least one column, the dynamic procedure applies to all columns in the Hierarchy.
- RunOnMultiPaste specifies whether the dynamic procedure will run when multiple values are pasted into the Data Table. By default, this option is FALSE.
  - TRUE The dynamic procedure will run when multiple values are pasted into the Data Table.
  - FALSE The dynamic procedure will not run when multiple values are pasted into the Data Table.

Example:

```
DYNAMICPROCEDURE CELL, "RecalcForecast.lvpro"

DYNAMICPROCEDURE ROW, "CalculateTotalPrice.lvpro", UNITPRICE|VOLUME

DYNAMICPROCEDURE ROW, "CalculateTotal.lvpro"

DYNAMICPROCEDURE TABLE, "Validate.lvpro"

DYNAMICPROCEDURE CELL, "CalculateTotal.lvpro", PRICE|VOLUME,
RunOnMultiPaste:TRUE
```

EditProcedure

Use this Hierarchy View function to add a custom procedure that the user can invoke to edit a cell. For example, you can invoke a form to prompt the user for input and calculate a value for the cell. To set the value back to the cell, you can use the SetDataTableCell function.

Note: EditProcedure can only be specified for columns of type STRING and NUMBER.

Syntax:

```
EditProcedure ColumnName, "Procedure" [,AllowManualInput:TRUE|FALSE]
```

where:

- ColumnName is the name of the DataTable column for which the procedure will be run when the user clicks on the edit control.
- Procedure is the procedure to run, enclosed in double quotation marks. The procedure must exist.
- AllowManualInput specifies whether the user can type in the textbox. By default, this option is FALSE.
  - TRUE User can type in the textbox or click a button to invoke the procedure
  - FALSE User cannot type in the textbox and must click the button to invoke the procedure

Example:

```
EditProcedure "Salary", "EditSalary.lvpro", AllowManualInput:TRUE
```

Expand (for Hierarchy Views)

Use this Hierarchy View function to specify the initial expansion state of the items in the Hierarchy.

Syntax:

```
Expand numLevels
```

where:

- numLevels is the number of levels the root of a Hierarchy is to be expanded when it is opened. Default is 99 which means the root is fully expanded.

Example:

```
Expand 2
```

NameColumn

Use this Hierarchy View function to specify the column that stores name of items in the Hierarchy.

Syntax:

```
NameColumn ColumnName
```

where:

- ColumnName is the name of the DataTable column where name of items can be found in a Hierarchy.

Example:

```
NameColumn SymbolName
```

ParentColumn

Use this Hierarchy View function to specify the column that stores parent of items in the Hierarchy.

Syntax:

```
ParentColumn ColumnName
```

where:

- ColumnName is the name of the DataTable column where parent of items can be found in a Hierarchy.

Example:

```
ParentColumn Parent
```

Parentheses

Use this Hierarchy View function to specify whether or not negative numbers display with parentheses instead of a negative sign. If you do not use this function, the negative numbers in the Hierarchy are preceded by a negative sign.

Syntax:

```
Parentheses ON|OFF
```

where:

- ON displays negative numbers enclosed in parentheses.
- OFF displays negative numbers preceded by a negative sign (-).

Example:

```
Parentheses ON
```

Protect

Use this Hierarchy View function to protect the specified columns in the Hierarchy from input by users. If a user adds a new row, the Protect function still applies to the specified columns in the newly-created row.

If you want the entire hierarchy to be read-only, use the ReadOnly function.

Note: If you use the Protect function and the EnabledCriteria keyword in the SetColumn function for the same column, Protect takes priority.

Syntax:

```
Protect ColumnName1[,ColumnName2,…]
```

where:

- ColumnName is the name of the column to protect from user input.

Example:

```
Protect ID,Name,EmployeeType
```

ReadOnly

Use this Hierarchy View function to make an entire Hierarchy read-only. This feature can be useful when you want to present data to users in a format in which they can sort and manipulate columns, but not add or submit data.

If you want only certain columns to be read-only, use the Protect function.

Syntax:

```
ReadOnly TRUE|FALSE
```

where:

- TRUE makes the Hierarchy read-only.
- FALSE makes the Hierarchy editable.

Example:

```
ReadOnly TRUE
```

SetColumn (for Hierarchy Views)

Use this Hierarchy View function to specify display options for a column.

Syntax:

```
SetColumn "ColumnName ", "Description:My Description",
["EnabledCriteria:ConditionalExpression"]
```

where:

- ColumnName is the name of the DataTable column to display in the Hierarchy. Separate multiple column names with a comma.
- My Description is the column header text to display. If you do not specify a description, the ColumnName displays.
- ConditionalExpression is optional and is the conditional expression under which a column is enabled or disabled.

Note: This option does not apply to virtual DataTables.

If the result of the expression is not zero, the column is enabled. If the result of the expression is zero, the column is disabled. Conditional expressions can include conditional operators, attributes, and column tokens. You must enclose column tokens in escaped quotes (\").

Depending on the data type for the column, there are special considerations for the ConditionalExpression for EnabledCriteria.

Note: If you use the Protect function and the EnabledCriteria keyword in thefunction for the same column, Protect takes priority.

Data type	Considerations for conditional expressions
autouser, user, and userlist	User values for expressions are case insensitive. For conditional expressions with Longview users, you must use the following syntax: `LONGVIEW\username` If you're comparing a user, you need to enclose the username in quotation marks. These quotation marks then need to be escaped using the backslash.
boolean	Conditional expressions must use 1 and 0 for TRUE/FALSE values, respectively.

Data type	Considerations for conditional expressions
date	Conditional expressions must use the following format for dates. "EnabledCriteria:\"{{Column,ColumnName}}\" EQ \"yyyy-mm-dd\hh:mm:ss"" You must include the time, in the format hh:mm:ss, for SQL databases. The time is optional (but unused) for Oracle databases. If you're comparing a Date, you need to enclose the Date in quotation marks. You must then escape these quotes using the backslash, as indicated in the syntax above.
file	EnabledCriteria is not supported for columns with this data type.
number	Conditional expressions must match the number in the database exactly.
string	String values for expressions are case sensitive. If you're comparing a string, you need to enclose the string in quotation marks. These quotation marks then need to be escaped using the backslash.
symbol	Symbol values for expressions are case insensitive. If you're comparing a symbol, you need to enclose the symbol in quotation marks. These quotation marks then need to be escaped using the backslash.

Example:

```
SetColumn "ID", "Description:Employee Number"


SetColumn "Notes", "EnabledCriteria:{{Column,Amount}}>0"

SetColumn "Reviewer", "Description:Reviewer", "EnabledCriteria:\"
{{Column,Workflow}}\"

SetColumn "Approver", "Description:Approver", "EnabledCriteria:\"
{{Column,Workflow}}\" != 1"

SetColumn "Amount", "EnabledCriteria:\"{{Column,Tax_Due}}\" EQ \"Payment\"
OR \"{{Column,Tax_Due}}\" EQ \"Refund\""
```

ShowToolbarButtons (for Hierarchy Views)

Use this Hierarchy View function to specify the buttons to show in the Hierarchy's toolbar. If you do not specify this function, no buttons display in the Hierarchy's toolbar.

Syntax:

```
ShowToolbarButtons ButtonName|All
```

where:

- ButtonName can be one or more of the following, separated by a pipe (|):

Button name	Description
Add	Displays a button that allows users to add a new row as a child to the current row to the Hierarchy.
Edit	Display a button that allows users to edit certain fields of the current row in the Hierarchy.
Delete	Displays a button that allows users to delete the current row from the Hierarchy.
Duplicate	Displays a button that allows users to duplicate the current row in the Hierarchy.
Reset	Displays a button that allows users to reset the current row in the Hierarchy that contain an error.
ResetAll	Displays a button that allows users to reset all rows in the Hierarchy that contain an error.

All displays all of the above buttons in the Hierarchy.

Example:

```
ShowToolbarButtons All

ShowToolbarButtons Add|Duplicate|Reset|ResetAll
```

SortOrderColumn

Use this Hierarchy View function to specify the column that stores sort order of items in the Hierarchy.

Syntax

```
SortOrderColumn ColumnName
```

where:

- ColumnName is the name of the DataTable column where sort order of items can be found in a Hierarchy.

Example:

```
SortOrderColumn SortOrder
```

SystemAction (for Hierarchy Views)

Use this Hierarchy View function to add a button with a predefined system action to the toolbar. For example, you can use system actions to include a toolbar button that allows users to export to Microsoft Excel or to add a button that allows users to specify whether they view symbol names, descriptions, or an attribute value in a Hierarchy.

Syntax:

```
SystemAction Action, "Text", "Image", "Helptext"
```

where:

- Action is the system action to add to the toolbar and can be one of the following values:
  - Close, which adds a button that users can click to close the current Hierarchy.
  - ExportToExcel, which adds a button that allows users to export the current view to Microsoft Excel.
  - PrintPreview — Adds a button that allows users to open a Print Preview dialog.
  - Users' printing preferences are saved when the Hierarchy is closed.
  - ViewLayout, which adds a button with drop-down options that allow the user to reload the most recently saved view layout (the layout is saved when the Hierarchy is closed) or reset the view to its original layout (as specified by the developer that created the Hierarchy).

- Text is the text to display on the button. If you want to omit this parameter and display the image only, use two double quotation marks (""). The following table lists the default text.

Keyword	Default text
Close	Close
ExportToExcel	Export to Excel
PrintPreview	Print Preview
ViewLayout	Layout

- Image is the image to display on the button. This option does not apply to the Close action. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. If you want to omit this parameter, the system uses a default icon. To omit this parameter, use two double quotation marks (""). The following table indicates the default images.

Keyword	Default image
ExportToExcel	
PrintPreview	
ViewLayout	

- Helptext is the tooltip text that appears when a user hovers over the button. There is no default for this parameter.

Example:

```
SystemAction ExporttoExcel, "Export", "Resources/images/toolbar/Excel.png",
"Export the current view to Excel"

SystemAction PrintPreview "", "", "Print Preview"
```

ThousandsSeparator

Use this Hierarchy View function to specify whether numeric values display with a locale-specific thousands separator. For example, if your location setting is “United States”, numeric values display with commas (,) as the thousands separator.

Syntax:

```
ThousandsSeparator ON|OFF
```

where:

- ON displays a locale-specific thousands separator based on the operating system setting.
- OFF does not display a thousands separator. If this function is not specified, OFF is used.

Example:

```
ThousandsSeparator ON
```

ValuesPane

Use this Hierarchy View function to present the columns of the active row in a pane where the columns are transposed (presented in rows). The values pane is ideal for Hierarchies containing numerous columns which would require users to scroll horizontally in order to enter row details.

If you do not use this function, the Hierarchy displays with the values pane feature disabled. If the values pane feature is ON for the Hierarchy, but the pane is not present, double-click the row selector to open the values pane.

Syntax:

```
ValuesPane ON|OFF [, "Title:MyTitle"] [, "HideOnLoad:TRUE|FALSE" ]
```

where:

- ON will enable the values pane feature.
- OFF will disable the values pane feature.
- MyTitle is the text to display in the pane's title bar. The default title is "Values Pane".
- HideOnLoad define whether the values pane is visible when the Hierarchy is initially displayed. This option can have one of the following values:

Value	Description
TRUE	Opens the Hierarchy with the values pane in view.
FALSE	The Hierarchy is opened without the values pane in view. This is the default value. The user must double-click on the row selector to open the values pane if required.

Example:

```
ValuesPane ON, "Title:TaskDetails", "HideOnLoad:FALSE"


ValuesPane ON, "Title:Properties", "HideOnLoad:FALSE"
ValuesPane ON, "Title:TaskDetails", "HideOnLoad:TRUE"
ValuesPane ON
ValuesPane OFF
```

Creating Calendar View Definition Files

Calendar View definition files define the layout, format, and behavior of data in a Calendar. The Calendar View definition file is also referenced by the Show DataTable command used in a procedure file.

Calendar View definitions are stored as ASCII files and can be created and edited in any text editor. The recommended file extension for Calendar View definitions is .lvcvw.

> **Note:** The data displayed in a Calendar is determined by the DataTable object specified in the Show DataTable command. For more information, see Defining DataTable objects and Show DataTable.

Action (for Calendar Views)

Use this function to add a toolbar icon or a context menu with an associated action in a Calendar. For example, you can use actions to run a procedure, open a Table or Data Grid, or run a model.

You can reference any of the above context variables in Longview Application Framework documents.

Syntax:

```
Action "Text", "Image", "Command", "Helptext"[, Location]
```

where:

- Text is the text to display in the toolbar. To display an image-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.
- Image is the path to the image to display as the toolbar or context menu icon. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. To display a text-only icon, omit this parameter and use two double quotation marks (""). You must specify either Text or Image.

  > **Note:** Longview recommends using 16 x 16 pixel images to optimize the appearance of your Calendar toolbar.

- Command is the Application Framework command to execute when a user clicks the toolbar icon or context menu item. To display a disabled icon, omit this parameter and use two double quotation marks (""). For example, you can create a button that executes the Launch App command.
- Helptext is the tooltip text that appears when a user hovers over the toolbar icon. To omit tooltip text, use two double quotation marks ("").
- Location is optional and defines the location of the action in the user interface. To add multiple locations, separate the parameters with a pipe (|). If you do not specify a location, the action item is added to the toolbar. Location can be one of the following:

Parameter	Description
TOOLBAR	Adds the action to the Calendar toolbar

Note: If you require double quotation marks (") in the parameters for the Action, precede them with a backslash.

Example:

```
Action "Clear", "Common/images/toolbar/Eraser.png", "Run Procedure
ClearTable.lvpro", "Clear Table", TOOLBAR

Action "Expenses", "Resources/app images/new.png", "Launch App Expenses"
"Expenses", TOOLBAR
```

DateColumn

Use this function to specify the column that contains the values to use as dates in the Calendar.

Syntax:

```
DateColumn ColumnName
```

where:

- ColumnName is the name of the App table column that contains the values to use as dates.

Example:

```
DateColumn DueDates
```

DescriptionColumn

Use this function to specify the column that contains the values to use as descriptions in the Calendar.

Syntax:

```
DescriptionColumn ColumnName
```

where:

- ColumnName is the name of the App table column that contains the values to use for descriptions.

Example:

```
DescriptionColumn Events
```

SystemAction (for Calendar Views)

Use this Calendar View function to add a button with a predefined system action to the toolbar. For example, you can use system actions to include toolbar buttons that allow users to refresh and close the Calendar.

Syntax:

```
SystemAction Action, "Text", "Image", "Helptext"
```

where:

- Action is the system action to add to the toolbar and can be one of the following values:
  - Close, which adds a button that users can click to close the current Calendar.
- Text is the text to display on the button. If you want to omit this parameter and display the image only, use two double quotation marks (""). The following table lists the default text.

Keyword	Default text
Close	Close

- Image is the image to display on the button. This option does not apply to the Close action. To omit this parameter, use two double quotation marks ("").
- Helptext is the tooltip text that appears when a user hovers over the button. There is no default for this parameter.

Example:

```
SystemAction Close, "", "", ""
```

Testing your Longview Table

Longview strongly recommends testing your Longview Table before you publish your Longview App to users.

Testing is considered complete after you have tested the following, if applicable to your Table:

- The Table loads without any initial warnings displayed to the user.
- The Table is read-only or editable, as desired. For more information, see Understanding why a Table is read-only.
- The Table loads with all controls and images displayed as desired.
- The Table loads with the correct valid values for drop-down lists and multiple selection lists.
- The Table has the correct defaults for check boxes, if applicable.
- The Table honors column order and correctly enables columns based on their EnabledCriteria, if applicable.
- If you are providing a Table with preexisting data to your users (for example, through import), make sure that any yellow error indicators are resolved.

Understanding why a Table is read-Only

There are three cases in which a Table will display as read-only:

- if you did not include the column from the database table that is the primary key
- if you created a Virtual DataTable without the WRITABLE option specified in the Table definition
- if you set the ReadOnly function to TRUE in your Table View definition

Longview recommends that you use the ReadOnly function or create a virtual DataTable to provide your users with a Table that is read-only.

Understanding why buttons are disabled

If a user does not have symbol access for at least one of the symbols specified for a symbol column, and the symbol column is not nullable, the Add, Duplicate, and Delete buttons will be disabled in the Table, and the user will not be able to add new rows.

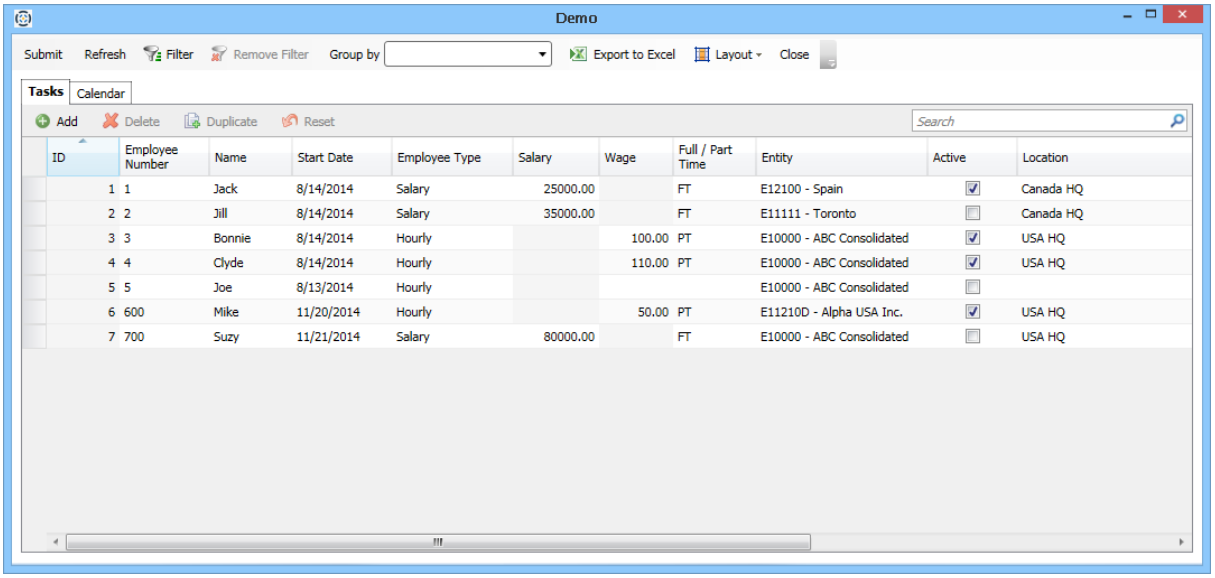
Sample Documents For Tables

This section includes code samples for both persistent and virtual DataTable objects:

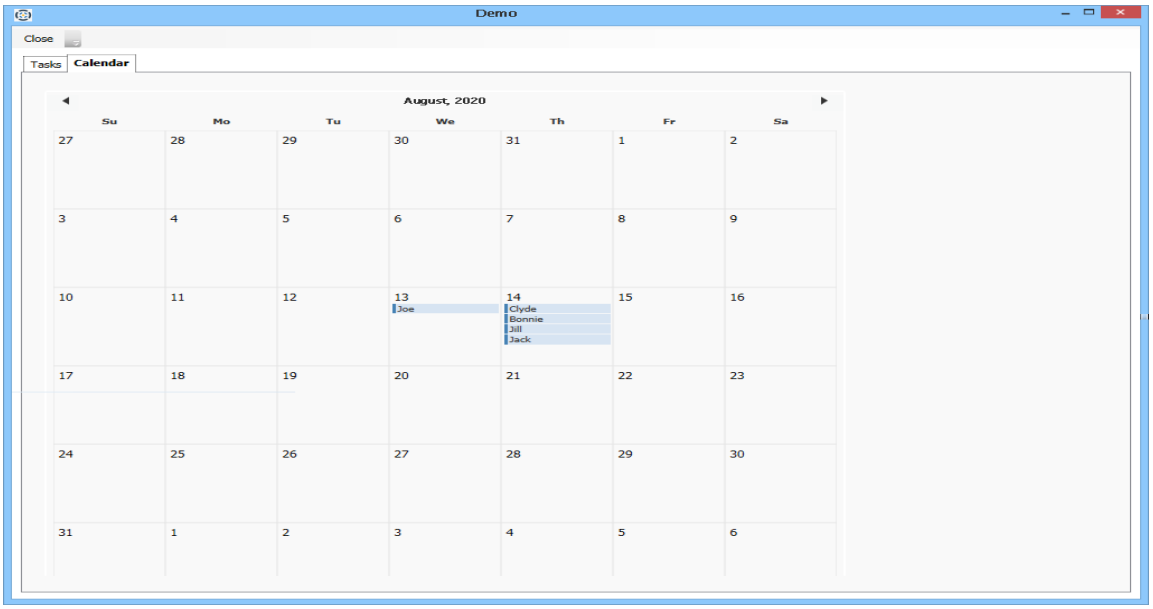
- Samples for a persistent DataTable object
- Samples for a virtual DataTable object

Samples for a persistent DataTable object

This section includes sample files that use a persistent DataTable as an example. You could create a Table that displays salary information by using similar code.



The same DataTable is displayed as a Calendar on the second tab:



To view samples for a specific file type, see:

- Sample DataTable definition
- Sample Table View definition
- Sample Hierarchy View definition
- Sample Calendar View definition
- Sample Procedure
- Sample App UI file

Sample DataTable definition

```
DATATABLE dtSalary

Columns ID, Name, Category, Type, Salary, Wage, Entity, Active, Start_Date,
Location, Manager, Vacation_Entitlement
```

Sample Table View definition

```
Columns ID, Name, Start_Date, Type, Salary, Wage, Category, Entity, Active,
Location


ColumnWidth 100

Protect Category

ApplyUserPreferences FALSE

Action "Submit", "Resources/images/submit.png", "Upload dtSalary", "",
TOOLBAR

SystemAction Filter, "", "", ""

SystemAction GroupBy, "GroupBy", "", ""

SystemAction ExporttoExcel, "Export to Excel", "", ""

Action "Clear Table", "Resources/images/reset-import.png", "DELETE
DATATABLEROWS FROM dtSalary", "", TOOLBAR

Action "Close", "Resources/images/exit.png", "SHOW RETURN", "", TOOLBAR

SystemAction ViewLayout, "Layout", "", ""

SystemAction PrintPreview, "", "", "Print Preview"
```

Sample Hierarchy View definition

```
Columns SymName,SymDesc,Parent,SortOrder,BalanceType,Notes

NameColumn SymName
DescriptionColumn SymDesc
ParentColumn Parent
SortOrderColumn SortOrder
SetColumn "SymName", "Description:Symbol Name"
SetColumn "SymDesc", "Description:Symbol Description"
SetColumn "SortOrder", "Description:Sort Order"
ColumnWidth 100
ColumnWidth 200, SymDesc
DefaultDecimals 0
ShowToolbarButtons All
Action "Submit", "", "Upload dtSymbols", "Submit", TOOLBAR
SystemAction Close, "", "", ""
```

Sample Calendar View definition

```
DateColumn Start_Date

DescriptionColumn Name
SystemAction Close, "Close", "", ""
```

Sample Procedure

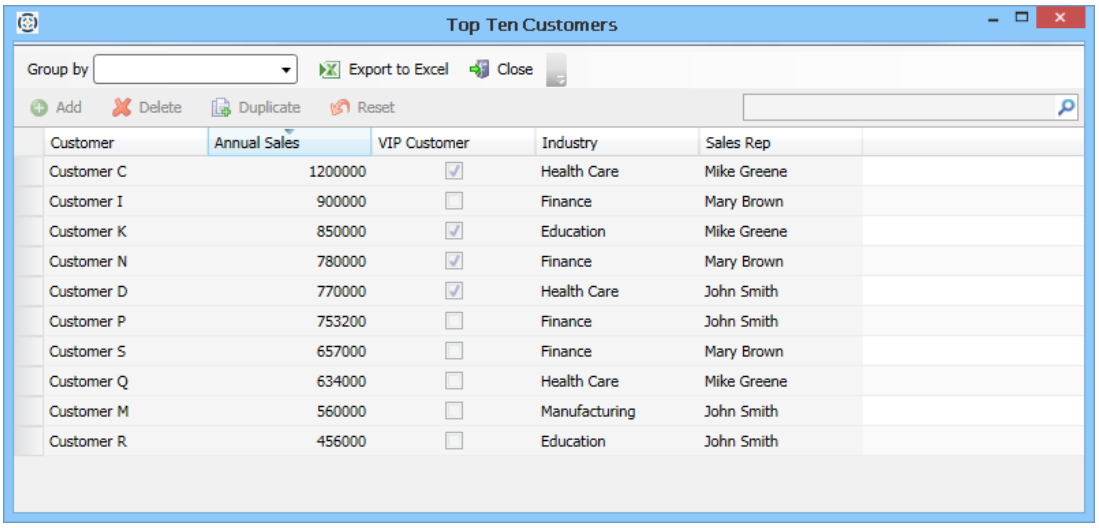
```
Create DATATABLE dtSalary using "Salary.lvdtd"

Create DATATABLE dtSymbols using "Symbols.lvdtd"
Download dtSalary
Download dtSymbols
Show DATATABLE dtSymbols USING HIERARCHY "Symbols.lvhvw"

Show UI USING ”Table.lvui"
```

Sample App UI file

```
LAYOUTTYPE Tabbed

DATATABLE, dtSalary, TABLE, "Salary.lvtvw", "Tasks", "", ""
DATATABLE, dtSalary, CALENDAR, "EmpStartDates.lvcvw", "Calendar", "", ""
```

Samples for a virtual DataTable object

This section includes sample files that use a virtual DataTable as an example. You could create a Table that displays a top ten list by using similar code.



To view samples for a specific file type, see:

- Sample DataTable definition
- Sample ImportSpec
- Sample source file for ImportSpec
- Sample Table View definition
- Sample Procedure

Sample DataTable definition

```
// Define a Virtual Table to display Customer Sales

DATATABLE VIRTUAL
AddColumn STRING, "Customer"
```

```
AddColumn STRING, "Industry"
AddColumn STRING, "SalesRep"
AddColumn BOOLEAN, "Preferred", "Default:FALSE"
AddColumn NUMBER, "Value"
```

Sample ImportSpec

```
//Description: Imports Sales Data into a Virtual Table


//Specify the source file
DataSource TEXT, "Sales.txt", EXTERNAL, "@"

//Set fields in file (1 per field to be processed)
Set Customer, 1
Set Industry, 2
Set Value, 5
Set Preferred, 4
Set SalesRep, 3

Define Customer, MATCH, Customer
Define Industry, MATCH, Industry
Define Value, MATCH, Value
Define Preferred, MATCH, Preferred
Define SalesRep, MATCH, SalesRep

//Set the maximum number of errors before the import is stopped
MaxError 0

//Specify a file to record errors
ErrorFile "import_errors.txt", EXTERNAL

//Specify a file to log import details
LogFile "import_log.txt", EXTERNAL

//Specify Decimal Character
DecimalCharacter "."
```

Sample source file for ImportSpec

```
Customer A@Finance@Mike Greene@0@200000.00

Customer B@Manufacturing@Mary Brown@0@400000.00
Customer C@Health Care@Mike Greene@1@1200000.00
Customer D@Health Care@John Smith@1@770000.00
Customer E@Manufacturing@Mike Greene@0@100000.00
Customer F@Manufacturing@Mary Brown@0@350000.00
Customer G@Health Care@John Smith@0@280000.00
Customer H@Finance@John Smith@0@230000.00
Customer I@Finance@Mary Brown@0@900000.00
Customer J@Education@John Smith@0@234000.00
Customer K@Education@Mike Greene@1@850000.00
Customer L@Health Care@John Smith@0@330000.00
Customer M@Manufacturing@John Smith@0@560000.00
Customer N@Finance@Mary Brown@1@780000.00
Customer O@Manufacturing@John Smith@0@250000.00
Customer P@Finance@John Smith@0@753200.00
Customer Q@Health Care@Mike Greene@0@634000.00
Customer R@Education@John Smith@0@456000.00
Customer S@Finance@Mary Brown@0@657000.00
Customer T@Manufacturing@Mary Brown@0@230000.00
```

Sample Table View definition

```
// Specify columns in the DataTable object to show and their order

Columns Customer, Value, Preferred, Industry, SalesRep

//Specify column descriptions to display
SetColumn "Customer", "Description:Customer"
SetColumn "Industry", "Description:Industry"
SetColumn "SalesRep", "Description:Sales Rep"
SetColumn "Value", "Description:Annual Sales"
SetColumn "Preferred", "Description:VIP Customer"
```

```
// Sort by highest Sales
SortOrder "Value DESC"

// Restrict to show top ten customers
VISIBLEROWS 10

Columnwidth 120

SystemAction GroupBy, "GroupBy", "", ""

SystemAction ExporttoExcel, "Export to Excel", "", ""

SystemAction PrintPreview, "", "", "Print Preview"

Action "Close", "Resources/images/exit.png", "SHOW RETURN", "", TOOLBAR

ApplyUserPreferences FALSE
```

Sample Procedure

```
//Export Sales Data in Data Area to file

RUN EXPORT Sales.lvexp ON SampleDA

// Create a VIRTUAL Data Table
Create DATATABLE dtTableMode using "VirtualSalesTable.lvdtd"

// Import Salary Data into Data Table
RUN IMPORT "Sales.lvimp" TO dtTableMode

// Present user Salary Data in table mode
Show DATATABLE dtTableMode using TABLE "VirtualSalesTableTopTen.lvtvw"
```

Developing Longview Forms

Longview Forms technology was created to enable Developers to design consistent forms across the Longview product without a reliance on HTML. You can create dynamic forms that allow users to provide information for your Longview App. You can use the information users enter into your form in variables used in a procedure. You can call a form from a procedure using the SHOW FORM command. For more information on calling your form, see “Show Form”. For information on creating a dynamic form, see Designing dynamic forms.

A Longview Form document (.lvfrm) defines the layout, format, and controls to display for a form and contains form functions. As a general rule, the order of functions specified in your form document is the order in which fields appear on the form.

For information on guidelines and best practices for designing Longview Forms, see Understanding guidelines and considerations and Reviewing best practices.

Note: You can specify optional parameters in any order for forms functions.

Creating a Longview Form consists of the following steps:

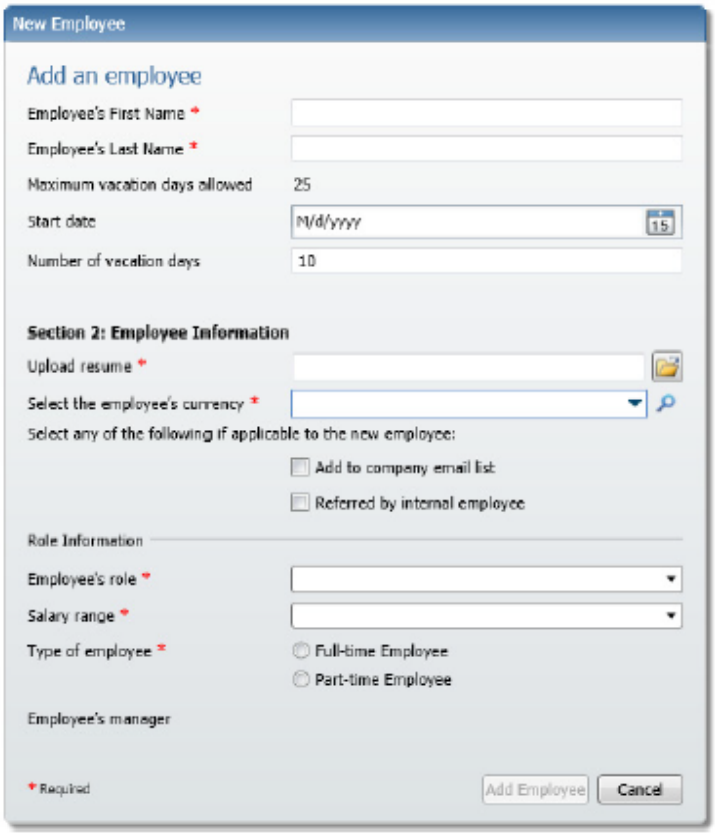
No	Step...	Document to use
1	Create a Longview App. For more information, see “Configuring Longview Apps”.	Longview App configuration file (.lvapp)
2	Create the supporting procedure for the Longview Form, including the Show Form command. For more information, see Show Form.	Longview procedure (.lvpro)
3	Create the Longview Form document as outlined in this chapter and create any required variables in the supporting procedure.	Longview Form document (.lvfrm), Longview procedure (.lvpro)
4	Optionally, create any procedures required to validate or dynamically update the form. For more information, see Designing dynamic forms.	Longview procedure (.lvpro)
5	Test your Longview Form and make changes as needed. For more information, see Reviewing best practices.	Possibly all supporting files (.lvapp, .lvpro, .lvfrm)
6	Deploy your Longview App. For more information, see “Deploying Longview Apps”.	All supporting files (.lvapp, .lvpro, .lvfrm6

You can use the following functions in a Longview Form document:

Button	Heading	SymbolSelector
CheckBox	Number	SymbolSpecSelector
ComboBox	NumberBox	Text
DateFormat	RadioGroup	TextBox
DatePicker	Separator	Window

Button	Heading	SymbolSelector
FileChooser	ScheduleSymbolSelector	

The following graphic shows a form created using a Longview Form document.



To view the code used to create this form, see Sample Longview Form and supporting procedures.

Understanding guidelines and considerations

Review the following guidelines and considerations before creating Longview Forms:

- Parameters
- Global Variables
- Layout
- UseListDelimiter command

Parameters

Longview Forms functions contain keyword-value pairs for optional parameters, such as “Title:title”. This syntax enables the system to identify the keyword and pass on the value without requiring keyword-value pairs to be in a particular order. For all optional parameters for Longview Forms functions, you can specify the optional parameters in any order.

Though not recommended, in some cases you may want to include a blank label for a control. For other Longview Application Framework commands and functions, you must use two double quotation marks ("") for blanks; however, to include a blank label for Longview Form functions, use a space enclosed in double quotation marks (" ").

Global Variables

For Longview Forms, you can use the following global variables for which you can set values:

- LVG_FormValid — this variable returns custom validation logic.
- LVG_FormMessage — this variable returns a customizable message to a user after a form validates.

For more information, see Working with global variables.

Layout

Longview Forms contain two columns: the left-side label column and the right-side control column. The width of the left-side label column is dictated by the longest label included in any of the functions specified in your Longview Form document. The right-side control column uses the remaining available width of the form, as specified by the Window function, if included. If you do not specify the Width parameter of the Window function, or you do not include the Window function in your Longview Form document, the width of the form is 550 pixels.

Though not recommended, in some cases you may use a variable for the label parameter, which may cause the right-side control column of your form to be unreadable by users. Always test the way your form displays, including testing possible variable values that could be passed in by your users. For more information, see Reviewing best practices.

UseListDelimiter command

The ComboBox and RadioGroup forms functions allow STRINGLIST variables that use a delimiter to separate values. Depending on your system and the delimiter used, you may need to use the UseListDelimiter command to convert at sign (@) delimiters to pipes (|) or vice versa.

Reviewing best practices

Longview strongly recommends testing your Longview Form before you publish your Longview App to users. Testing is considered complete after you have tested the following, if applicable to your form:

- The form loads with all controls displayed as desired and no error messages.
- The form loads without any initial warnings displayed to the user.
- The form loads with target values resolved and displayed as expected.
- The form loads with the expected available values for prompt controls such as the Symbol Selector and Symbol Spec Selector.
- Any specified variables populate as expected after controls are completed by the user.

- Any specified variables that are referenced in subsequent procedures are set before being referenced.
- Any specified buttons become available as expected based on the Validate keyword, if included.
- Any specified OnChange procedures that modify controls do so as expected.

Designing dynamic forms

As a Developer, you can create simple forms that use information entered by users to update your Longview Apps; however, you can also create complex and dynamic forms. The level of complexity of your form is up to you. A dynamic form is one that updates as the user completes the fields. For example, you could write your own validation procedure to display a message to the user if they do not include an at sign (@) when prompted to type their email address, or you could write an OnChange procedure to modify the selections in a combo box based on the selection from another control.

To view sample code, see Sample Longview Form and supporting procedures.

Using the OnChange keyword

Many Longview Forms functions allow you to use the optional OnChange keyword. You can use the OnChange keyword to specify a procedure to run after the system detects a change - this lets you add your own validation and/or create a form that updates dynamically as the user fills out the form. OnChange procedures trigger immediately after the user modifies the control such as typing in a text box or selecting a radio button.

> **Note:** OnChange procedures cannot update the available symbols in the Symbol Selector control for the SymbolSelector and SymbolSpecSelector functions.

Using the OnClick keyword

The Button function provides you with the optional OnClick keyword. You can use the OnClick keyword to specify a procedure to run after the user clicks the button. You could, for example, write an OnClick procedure to kick off a calculation or you could write an OnClick procedure to add a new symbol to the system using the values for the variables that the user entered in your form. For more information, see Button.

> **Note:** Target variables are populated with the values entered or selected by the user when they complete the form. If you include a "Cancel" button and want target variables reset to default values, it is up to you to write an appropriate OnClick procedure that reverts the target variables.

Defining properties

You can use the functions in this section to define the properties of your form:

- Button
- Window

Button

Use this function to define the buttons included on your form. Buttons appear on the button bar at the bottom of the form. To add more than one button, you must include more than one Button function. The order in which you specify Button functions in your form document is the order in which buttons appear from left to right on your form. Optionally, you can specify a procedure to run after the user clicks the button.

Note: If you do not include the Button function in your form document, the form includes a Close button, by default, that is always available.

Syntax:

```
Button "Value"[, "Text:TextValue"][, Validate:TRUE|FALSE][,
"OnClick:Procedure"]
```

where:

- Value is the value returned to the LVS_BUTTONCLICKED system variable when the user clicks the button. For more information, see “Using the LVS_BUTTONCLICKED system variable”.
- TextValue is optional and is the text to display on the button. If you do not specify TextValue, the content specified for Value displays as the button text.
- Validate is optional and can be one of the following: TRUE or FALSE.
- Procedure is optional and is the name of the OnClick procedure to run after the user clicks the button. To close the form, include the SHOW RETURN command in your OnClick procedure. If you do not specify an OnClick procedure, the system automatically executes the SHOW RETURN command when the user clicks the button, and closes the form. Include the extension and the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory.

For more information, see “Using the OnClick keyword”.

Syntax Example 1

```
Button "Add", "Text:Add Employee", Validate:TRUE, "OnClick:button.lvpro"
```

Syntax Example 2

```
Button "Cancel", "Text:Cancel"
```

Window

Use this function to specify the title, width, and height of the form window. All parameters of the Window function are optional; however, if you include the Window function in your form document, you must specify at least one parameter.

Syntax:

```
Window [“Title:Title”][, Width:Width][, Height:Height]
```

Note: You must specify at least one of Title, Width, or Height if you include the Window function in your form document.

where:

- Title is optional and is the string used as the title for the form window. Enclose this parameter in double quotation marks (""), including the Title keyword. If you do not specify Title, the form opens with the Longview App description as the title of the form window.
- Width is optional and specifies the width, in pixels or as a percentage, of the form window. Indicate a percentage with a percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. If you do not specify Width, the default width of the form is 450 pixels.
- Height is optional and specifies the height, in pixels or as a percentage, of the form window. Indicate a percentage with a percent (%) sign. If you do not use a percent sign, the parameter is assumed to be in pixels. Review the following table for the expected behavior of form height:

If...	The system...
you specify Height as a value too small for the user to interact with the form...	overrides your specification for Height and imposes a minimum height of 100 pixels so that the button bar displays.
you specify Height as a value too large for the maximum height available on the user’s screen...	overrides your specification for Height and displays the form with the maximum height available on the user’s screen and includes a vertical scroll bar on the right-hand side of the form.

Syntax Example 1

```
Window "Title:New Employee"
```

Syntax Example 2

```
Window Width:300
```

Syntax Example 3

```
Window "Title:New Employee", Width:15%, Height:30%
```

Defining formats

You can use the functions in this section to define formats for your form:

- DateFormat

DateFormat

Use this function to specify the format in which to return the value of a date that a user enters into a control specified by the DatePicker function. You may want to, for example, kick off a rollover process based on the month the user selects. To do so, you must know which number in the selected date refers to the day, month, and year. If you do not include the DateFormat function in your form document, date values are returned in the ISO format YYYY-MM-DD. For more information, see DatePicker.

Note: Dates display to the user in the format specified by their operating system's regional settings.

Syntax:

```
DateFormat "Format"
```

where:

- Format specifies the format in which the value of a date input is returned, enclosed in double quotation marks (“”), and can be one of the following:

Note: The default format if you do not include the DateFormat function in your form document is the ISO format YYYY-MM-DD.

Syntax Example 1

```
DateFormat "DD-MM-YYYY"
```

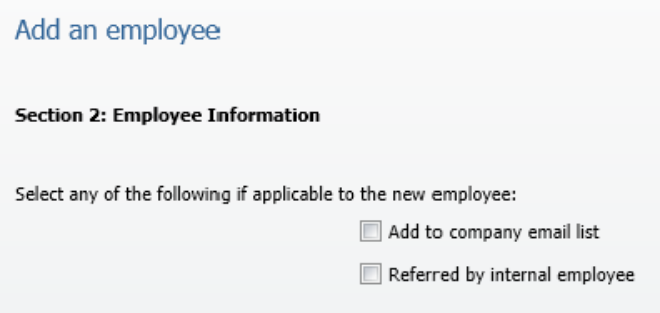
Displaying information

You can use the functions in this section to display information on your form:

- Heading

Heading

Use this function to include text that spans the label column and the control column of the form. You can use the Heading function to include a heading with larger text or, for example, to provide instructions to users in the default text size and style. You can also use the Heading function to break your form into sections. For more information on form layout, see Layout.



Note: The CheckBox field also appears in the above graphic as an example of a heading spanning the full width of a form.

Syntax:

```
Heading "Text"[, Style:Title|Subtitle|Normal]
```

where:

- Text is the text to display, enclosed in double quotation marks (""). To include a blank row in your form, use the Separator function. For more information, see Separator.
- Style is optional and is the appearance of the text. This parameter can be one of the following values:

Value	Description
Title	Displays the specified text in Tahoma 16px Bold #FF336699.
Subtitle	Displays the specified text in Tahoma 11px Bold Black.
Normal	Displays the specified text in Tahoma 11px Black. This is the default if you do not specify Style.

Syntax Example 1

```
Heading "Add an employee", Style:Title
```

Syntax Example 2

```
Heading "Section 2: Employee Information", Style:Subtitle
```

Syntax Example 3

```
Heading "Select any of the following if applicable to the new employee:"
```

Adding static controls

You can use the functions in this section to add static controls to your form:

- Number
- Separator
- Text

Number

Use this function to display a labelled number to users. You can use the Number function to display a static number or to display a number based on another input on your form. For example, you could display a new employee's number of vacation days remaining in a calendar year based on the employee's start date. For more information, see Designing dynamic forms.

You can specify whether a number displays left- or right-aligned on the form, and you can also specify whether to display numbers using the user's regional settings for thousand separators.

```
Maximum vacation days allowed          25
```

Syntax:

```
Number "Label", Value [, Align:LEFT|RIGHT][,ThousandsSeparator:ON|OFF]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Value is a number or the name of the NUM variable created in the supporting procedure for this Number function.
- Align is optional and specifies where to display the number on the form. Use LEFT to display the number on the left-hand side of the control column. Use RIGHT to display the number on the right-hand side of the control column. If you do not specify Align, numbers display on the right-hand side of the control column by default.
- ThousandsSeparator is optional and specifies whether to display numbers using the user's region-specific character to separate thousands. Use ON to display numbers with a thousand separator. Use OFF to display numbers without a thousand separator. If you do not specify ThousandsSeparator, numbers display without a thousand separator character.

Syntax Example 1

```
Number "Maximum vacation days allowed", 25, Align:LEFT
```

Syntax Example 2

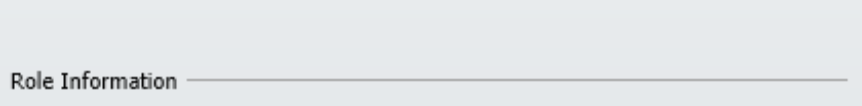
```
Number "Vacation days in current year", nRetrodays
```

Syntax Example 3

```
Number "Suggested Salary", nSuggestedSalary, Align:RIGHT,
ThousandsSeparator:ON
```

Separator

Use this function to include a separator in your form. You can use the Separator function to break up areas on your form with blank space or a line, or you can optionally include a text label to divide your form into separate sections.



Syntax:

```
Separator Style[, "Text:TextValue"]
```

where:

- Style is one of the following:

Value	Description
BLANK	Use this value to insert a blank horizontal separator that spans the width of the form.
LINE	Use this value to insert a solid horizontal line that spans the width of the form.

- TextValue is optional and is the left-aligned text to display as the label for the separator. Enclose this value in double quotation marks (""), including the Text keyword.

Syntax Example 1

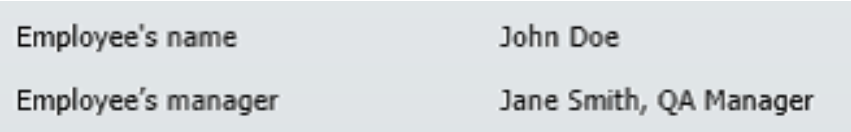
```
Separator BLANK
```

Syntax Example 2

```
Separator LINE, "Text:Role Information"
```

Text

Use this function to display labelled text to users. You can use the Text function to display static text or to display text based on another input on your form. For example, you could display a new employee's manager based on their role. For more information, see "Designing dynamic forms".



Syntax:

```
Text "Label", Value
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Value is a string or the name of the STRING variable created in the supporting procedure for this Text function.

Syntax Example 1

```
Text "Employee's name", "John Doe"
```

Syntax Example 2

```
Text "Employee's manager", sEmpManager
```

Adding interactive controls

You can use the functions in this section to add input controls to your form:

- DatePicker
- FormulaBuilder
- NumberBox
- TextBox

DatePicker

Use this function to add a date picker to your form. Users can type or select a date that is input into a specified target variable. Dates display to the user in the format specified by their operating system's

regional settings and users are expected to type dates in their regional preference format. Optionally, you can specify a procedure to run after the date is input. You can also optionally restrict the range of dates the user is allowed to input.

Note: Date values are returned in the format specified by the DateFormat function. If you have not included the DateFormat function in your form document, date values are returned in the default ISO format YYYY-MM-DD. For more information, see DateFormat.

Syntax:

```
DatePicker "Label", Target[, "OnChange:Procedure"][, Min:Date][, Max:Date][,
Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this DatePicker function.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory.

  For more information, see Using the OnChange keyword.
- Date is optional and is the date range for the date picker, defined using the Min and Max keywords. Use the format specified by the DateFormat function, otherwise use the default format which is the ISO format YYYY-MM-DD. You do not have to specify both Min and Max; you may include only the Min keyword value pair to restrict the minimum allowed date or you may include only the Max keyword value pair to restrict the maximum allowed date.
- Required is optional and can be one of the following:

Value	Description
TRUE	Use this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Syntax Example 1

```
DatePicker "Start date", sStart, "OnChange:SetStartDate.lvpro", Min:2000-01-
01, Max:2016-01-01
```

FormulaBuilder

Use this function to add a formula builder to your form. Users can type a formula or select symbols and operators for a formula using interactive controls. The formula specified can be input into a target variable. Numbers are displayed and should be input based on the user's operating system regional format. Optionally, you can specify a procedure to run after the formula is input. You can also optionally restrict the symbols that the user is allowed to input.

Syntax:

```
FormulaBuilder "Label", Target, Dimension[,Symbols:SymbolSpecifications]
[,OnChange:Procedure"][, Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this FormulaBuilder function.
- DimensionName is the dimension containing the symbols to include in the Formula Builder.
- SymbolSpecifications is optional and is the level of symbols to display in the Formula Builder hierarchy, in relation to the specified symbol, for example, SymbolName### or SymbolName#99.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory.
- Required is optional and can be one of the following:

Value	Description
TRUE	Use this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Note: If the dimension selected is the time period dimension, floating time periods will be included in the symbol specifications.

Syntax example:

```
FORMULABUILDER "VarCostRatio", FormulaTarget, Accounts, Symbols:IncStmt###
```

NumberBox

Use this function to add an input box for numbers to your form. Users can type a number that is input into a specified target variable. Users can also enter the plus sign (+), minus sign (-), dot (.), and comma (,), and can use parentheses to indicate negative numbers, such as (2). Numbers user enter use the format specified by their operating system's regional settings, including their region-specific character for thousand separator, if you include the optional thousands separator keyword. Optionally, you can specify a procedure to run after the number is input. You can also optionally restrict the range of numbers the user can input or restrict the number of allowed decimals and specify whether numbers display left- or rightaligned.

Syntax:

```
NumberBox "Label", Target[, "OnChange:Procedure"][, Min:Number][,
Max:Number][, Decimals:Decimals][,Align:LEFT|RIGHT]
[ThousandsSeparator:ON|OFF][, Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the NUM variable created in the supporting procedure for this NumberBox function.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.
- Number is optional and is the number range for the number box, defined using the Min and Max keywords. You do not have to specify both Min and Max; you may include only the Min keyword value pair to restrict the minimum allowed number or you can include only the Max keyword value pair to restrict the maximum allowed number.
- Decimals is optional and specifies the number, from 0-9, of digits allowed after the decimal character. Use 0 to restrict the user from typing any numbers after the decimal character. The default value is 9 if you do not specify Decimals.

- Align is optional and specifies where to display the number within the number box. Use LEFT to display the number on the left-hand side of the number box. Use RIGHT to display the number on the right-hand side of the number box. If you do not specify Align, numbers display on the right-hand side of the number box by default.
- ThousandsSeparator is optional and specifies whether to display numbers using the user's region-specific character to separate thousands. Use ON to display numbers with a thousand separator. Use OFF to display numbers without a thousand separator. If you do not specify ThousandsSeparator, numbers display without a thousand separator character.
- Required is optional and can be one of the following:

Value	Description
TRUE	Use this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Syntax Example 1

```
NumberBox "Number of vacation days", nVacation,
"OnChange:ValidateVacation.lvpro", Min:10, Max:25, Decimals:0, Align:LEFT,
Required:TRUE
```

Syntax Example 2

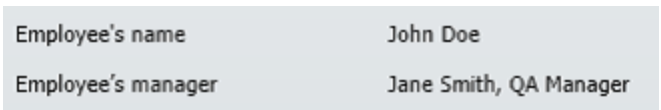
```
NumberBox "Suggested Salary", nSuggestedSalary,
"OnChange:ValidateSalary.lvpro", Min:5, Max:7, Decimals:0, Align:RIGHT,
ThousandsSeparator:ON, Required:TRUE
```

Syntax Example 3

```
NumberBox "Number of vacation days", oEmployee.nVacation
```

TextBox

Use this function to display labelled text to users. You can use the **Text** function to display static text or to display text based on another input on your form. For example, you could display a new employee's manager based on their role. For more information, see Designing dynamic forms.



Syntax:

```
Text "Label", Value
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Value is a string or the name of the STRING variable created in the supporting procedure for this Text function.

Syntax Example 1

```
Text "Employee's name", "John Doe"
```

Syntax Example 2

```
Text "Employee's manager", sEmpManager
```

Adding selection controls

You can use the functions in this section to add selection controls to your form:

- CheckBox
- ComboBox
- RadioGroup

CheckBox

Use this function to add a check box to your form. To add more than one check box, you must include more than one CheckBox function. Optionally, you can specify a procedure to run after the check box is selected.

Syntax:

```
CheckBox "Label", Target[, "OnChange:Procedure"]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING or NUM variable created in the supporting procedure for this CheckBox function. STRING variables return values of either TRUE or FALSE; NUM variables return values of either 1 or 0.

- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.

Syntax Example 1

```
Checkbox "Add to company email list", sEmailList,
"OnChange:SetEmaiList.lvpro"
```

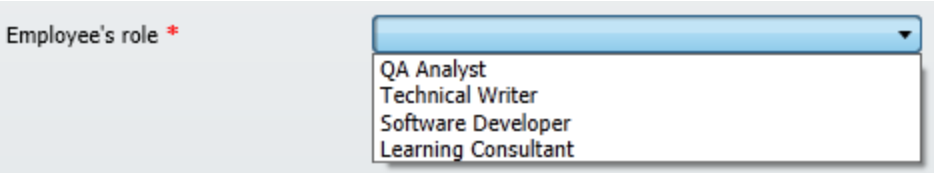
Syntax Example 2

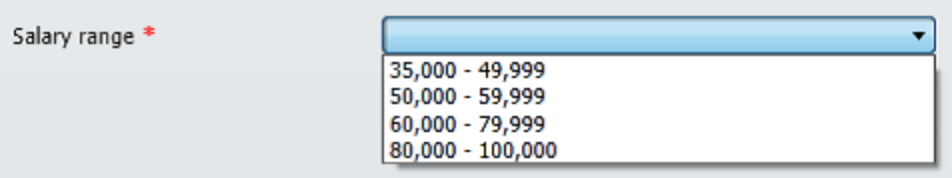
```
Checkbox "Referred by internal employee", sReferred
```

ComboBox

Use this function to add a non-editable combo box (also referred to as drop-down list) to your form. The order in which you specify the combo box values in this function is the order in which the options appear in the list. Optionally, you can specify a procedure to run after the user selects an option. Combo boxes are always treated as required fields, and a red asterisk displays beside combo box labels indicating to the user that the field is required.

Note: You may need to use the UseListDelimiter command to convert the delimiter used between values. For more information, see UseListDelimiter.





Syntax:

```
ComboBox "Label", Target, Values[, Text:TextValues][, "OnChange:Procedure"]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this ComboBox function. If Target contains a valid value, it is selected when the forms displays. If Target contains an invalid value or no value, the form displays with nothing selected.
- Values is a pipe (|) delimited string, or the name of the STRINGLIST, NUMLIST, or RANGE variable created in the supporting procedure for this ComboBox function to display as the drop-down options. Enclose values in double quotation marks ("").

  Note: Blank values are not valid.

- TextValues is optional, is the text to display for each drop-down option, and is a pipe (|) delimited string, or the name of a STRINGLIST, NUMLIST, or RANGE variable in the supporting procedure. If you do not specify TextValue, the content specified for Values displays as the text for the options in the list.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.

Syntax Example 1

```
ComboBox "Employee's role", sRole, "QA|DOC|SD|LC", "Text:QA
Analyst|Technical Writer|Software Developer|Learning Consultant",
"OnChange:SetRole.lvpro"
```

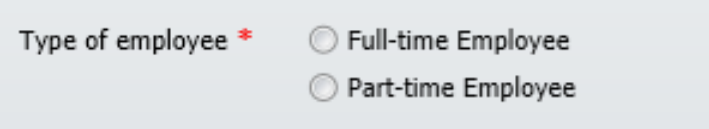
Syntax Example 2

```
ComboBox "Salary range", sSalary, "1|2|3|4", "Text:35,000 - 49,999|50,000 -
59,999|60,000 - 79,999|80,000 - 100,000", "OnChange:SetSalary.lvpro"
```

RadioGroup

Use this function to add a group of radio buttons to your form. The order in which you specify the radio values in this function is the order in which the buttons appear from top to bottom on your form. When the form displays for the first time, no radio button is selected by default, unless you set the target variable. Radio groups are always treated as required fields, and a red asterisk displays beside radio group labels indicating to the user that the field is required.

Note: You may need to use the UseListDelimiter command to convert the delimiter used between values. For more information, see UseListDelimiter.



Syntax:

```
RadioGroup "Label", Target, Values[, Text:TextValues][,
"OnChange:Procedure"]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this RadioGroup function.
- Values is a pipe (|) delimited string, or the name of the STRINGLIST, NUMLIST, or RANGE variable created in the supporting procedure for this RadioGroup function to display as the button options. Enclose values in double quotation marks ("").
- TextValues is optional, is the text to display for each button, and is a pipe (|) delimited string, or the name of the STRINGLIST, NUMLIST, or RANGE variable created in the supporting procedure for this RadioGroup function. If you do not specify TextValue, the content specified for values displays as the radio button text.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.

Syntax Example 1

```
RadioGroup "Type of employee", sEmpType, "FT|PT", "Text:Full-time
Employee|Part-time Employee", "OnChange:SetEmpType.lvpro"
```

Syntax Example 2

```
RadioGroup "Type of employee", sEmpType, "$emptype$", "Text:Full-time
Employee|Part-time Employee", "OnChange:SetEmpType.lvpro"
```

Adding prompt controls

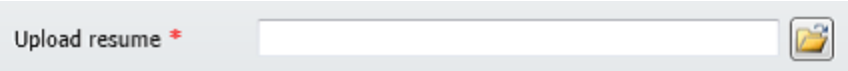
You can use the functions in this section to add prompt controls to your form:

- FileChooser
- ScheduleSymbolSelector

- SymbolSelector
- SymbolSpecSelector

FileChooser

Use this function to prompt users to select a file for your form. Optionally, you can specify a procedure to run after the file is selected. You can also optionally specify the types of files allowed and the button options for the prompt that opens.



Syntax:

```
FileChooser "Label", Target[, "FileRestrictions:FileRestrictions"[,
BrowseType:OPEN|SAVE][, "DefaultPath:Path"][, "OnChange:Procedure"][,
Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this FileChooser function.
- FileRestrictions is optional and is a list of the suggested file type and extension pairs for the user to select, enclosed in double quotation marks(""). Separate each file type and extension with a pipe (|).

  Note: If you want to allow all file types, omit the FileRestrictions parameter, or use "FileRestrictions:All Files|*.*".

- BrowseType is optional and can be Open or Save. Open or Save As appears as the title for the browse dialog that opens when users click the folder icon in the file chooser, and Open or Save appears as the button text option in the browse dialog.
- Path is optional and is the default path selected for the browse dialog, enclosed in double quotation marks.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the

working directory. For more information, see Using the OnChange keyword.

- Required is optional and can be one of the following:

Option	Description
TRUE	Use this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Syntax Example 1

```
FileChooser "Upload resume", sResume, "FileRestrictions:pdf file|*.pdf|text
file|*.txt", "BrowseType:SAVE", "DefaultPath:C:\Users\ABCUser\Documents",
"OnChange:SetResume.lvpro", Required:TRUE
```

ScheduleSymbolSelector

Use this function to add a Schedule Symbol Selector to your form. Users can select one or more symbols from the Schedule Symbol Selector for a specified schedule dimension. Optionally, you can specify a procedure to run after the symbol is selected.

Note: OnChange procedures triggered from other functions cannot update the Schedule Symbol Selector control.

Syntax:

```
ScheduleSymbolSelector "Label", Target, ScheduleName, DimensionName
[,IncludeTotal:TRUE|FALSE][,AllowMultiple:TRUE|FALSE][,
"OnChange:Procedure"][, Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING, STRING[ ], or RANGE variable created in the supporting procedure for this ScheduleSymbolSelector function.
- ScheduleName is the name of the schedule the ScheduleSymbolSelector will use.

- DimensionName is the schedule dimension containing the symbols to include in the Schedule Symbol Selector.
- AllowMultiple is optional and specifies whether users can select multiple symbols and can have a value of TRUE or FALSE. The default if you do not specify AllowMultiple is FALSE.
- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see “Using the OnChange keyword”.
- Required is optional and can be one of the following:

Value	Description
TRUE	Use this value to include a red asterisk beside the field indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required

Syntax example:

```
ScheduleSymbolSelector "Select the employee’s currency", sCurrency,
“UserInfoSchedule” EmployeeCurrency, IncludeTotal:FALSE,
AllowMultiple:FALSE, "OnChange:SetSymbols.lvpro", Required:TRUE
```

SymbolSelector

Use this function to add a Symbol Selector to your form. Users can select one or more symbols from the Symbol Selector for a specified dimension. Optionally, you can specify a procedure to run after the symbol is selected. You can also optionally specify a symbol specification, up to two attribute filters, and the allowable symbol types.

Note: OnChange procedures triggered from other functions cannot update the Symbol Selector control.

Syntax:

```
SymbolSelector "Label", Target, DimensionName[,
Symbols:SymbolSpecifications][, AllowLeaf:TRUE|FALSE][,
AllowParent:TRUE|FALSE][, AllowReadOnly:TRUE|FALSE][,
AllowMultiple:TRUE|FALSE][, "AttributeFilters:AttributeFilters"][,
"OnChange:Procedure"][, Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING, STRING[ ], or RANGE variable created in the supporting procedure for this SymbolSelector function.
- DimensionName is the dimension containing the symbols to include in the Symbol Selector.
- SymbolSpecifications is optional and is the level of symbols to display in the Symbol Selector hierarchy, in relation to the specified symbol, for example, SymbolName### or SymbolName#99.
- AllowLeaf is optional and specifies whether users can select leaf symbols and can have a value of TRUE or FALSE. The default if you do not specify AllowLeaf is FALSE.
- AllowParent is optional and specifies whether users can select parent symbols and can have a value of TRUE or FALSE. The default if you do not specify AllowParent is FALSE.
- AllowReadOnly is optional and specifies whether users can select read-only symbols and can have a value of TRUE or FALSE. The default if you do not specify AllowReadOnly is FALSE.
- AllowMultiple is optional and specifies whether users can select multiple symbols and can have a value of TRUE or FALSE. The default if you do not specify AllowMultiple is FALSE.
- AttributeFilters is optional can be up to two attribute filters linked by AND or OR, enclosed in double quotations marks, using the syntax:

  FilterType{AttrName{Operation{Expression

  where:

Value	Description
FilterType	specifies the method to use to search the hierarchy for symbols matching the filter criteria and can be one of ALL, PARENT, LEAF, or ROOT. If you specify two attribute filters, FilterType must be the same for both filters.
AttrName	is the name of an attribute.
Operation	can be EQ for exactly equal to or NE for not equal to.

Value	Description
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For non-list attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For list attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

If you specify two attribute filters, enclose each filter in parentheses, for example "(attributefilter1) AND (attributefilter2)".

- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file. If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.

- Required is optional and can be one of the following:

Value	Description
TRUE	Use this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Syntax Example 1

```
SymbolSelector "Select the employee's currency", sCurrency, CURRENCIES,
Symbols:Source_Currencies#99, AllowLeaf:TRUE, AllowParent:TRUE,
```

```
AllowReadOnly:TRUE, AllowMultiple:TRUE, "OnChange:SetSymbols.lvpro",
Required:TRUE
```

Syntax Example 2

```
SymbolSelector "Select an entity", sEntity, ENTITIES, Symbols:TENTITIES#99,
AllowLeaf:TRUE, AllowParent:TRUE, AllowReadOnly:TRUE, AllowMultiple:TRUE,
"AttributeFilters:LEAF{ZGPNativeCurrency{EQ{CAD",
"OnChange:SetSymbols.lvpro", Required:TRUE
```

SymbolSpecSelector

Use this function to add a Symbol Spec Selector to your form. The Symbol Spec Selector prompts users to select a symbol from the Symbol Selector, to select one of All, Leaf, Parent, or Root and Parent for the selected symbol, and to specify the number of levels down (froms 0 to 99) for the specified symbol. Optionally, you can specify a procedure to run after the symbol is selected. You can also optionally specify a symbol specification, up to two attribute filters, and the allowable symbol types.

Note: OnChange procedures triggered from other functions cannot update the Symbol Selector control.

Syntax:

```
SymbolSpecSelector "Label", Target, DimensionName[,
Symbols:SymbolSpecifications][, AllowLeaf:TRUE|FALSE][,
AllowParent:TRUE|FALSE][, AllowReadOnly:TRUE|FALSE][,
"AttributeFilters:AttributeFilters"][, "OnChange:Procedure"][,
Required:TRUE|FALSE]
```

where:

- Label is the label for the field, enclosed in double quotation marks ("").
- Target is the name of the STRING variable created in the supporting procedure for this SymbolSpecSelector function. The Symbol Spec Selector populates with the selections from the specified target variable. If you leave target blank by using double quotation marks (""), the Symbol Spec Selector populates with no symbol selected, ALL as the symbol definition, and 99 as the default number of levels. If you specify an invalid symbol, the Symbol Spec Selector loads with no symbol and a warning message displays.
- DimensionName is the dimension containing the symbols to include in the Symbol Selector.
- SymbolSpecifications is optional and is the level of symbols to display in the Symbol Selector hierarchy, in relation to the specified symbol, for example, SymbolName### or SymbolName#99.
- AllowLeaf is optional and specifies whether users can select leaf symbols from the Symbol Selector control and can have a value of TRUE or FALSE. The default if you do not specify

AllowLeaf is FALSE.

> **Note:** AllowLeaf is tied to the symbols available to the user in the Symbol Selector control and not the symbol definition control. Users can always select one of All, Leaf, Parent, or Root and Parent from the symbol definition control.

- AllowReadOnly is optional and specifies whether users can select read-only symbols from the Symbol Selector control and can have a value of TRUE or FALSE. The default if you do not specify AllowReadOnly is FALSE.

- AttributeFilters is optional can be up to two attribute filters linked by AND or OR, enclosed in double quotations marks, using the syntax:

  FilterType{AttrName{Operation{Expression

  where:

Value	Description
FilterType	specifies the method to use to search the hierarchy for symbols matching the filter criteria and can be one of ALL, PARENT, LEAF, or ROOT. If you specify two attribute filters, FilterType must be the same for both filters.
AttrName	is the name of an attribute.
Operation	can be EQ for exactly equal to or NE for not equal to.
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For non-list attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For list attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

  If you specify two attribute filters, enclose each filter in parentheses, for example " (attributefilter1) AND (attributefilter2)".

- Procedure is optional and is the name of the OnChange procedure to run after the user changes the field. Always include the extension, and include the file path if the file is not located in the default path. The default path is the path specified in the <rootpath> tag of the .lvapp file.

If no root path is specified, the system assumes the root path is the applications folder in the working directory. For more information, see Using the OnChange keyword.

- Required is optional and can be one of the following:

Value	Description
TRUE	sUse this value to include a red asterisk beside the field label indicating to the user this field is required. This value ensures any buttons specified in the form document that include the optional Validate:TRUE parameter are unavailable until fields marked as required are completed by the user. For more information, see Button.
FALSE	Use this value to leave the field label without an asterisk. This is the default if you do not specify Required.

Syntax Example 1

```
SymbolSpecSelector "Select a time period", sTimePeriod, TIMEPER,
Symbols:Budget_Periods#99, AllowLeaf:TRUE, AllowParent:TRUE,
AllowReadOnly:TRUE, "OnChange:SetSymbolSpec.lvpro", Required:TRUE
```

Syntax Example 2

```
SymbolSpecSelector "Select an entity", sEntity, ENTITIES,
Symbols:TENTITIES###, AllowLeaf:TRUE, AllowParent:TRUE, AllowReadOnly:TRUE,
"OnChange:SetSymbolSpec.lvpro", Required:TRUE
```

Sample Longview Form And Supporting Procedures

The following code blocks show the Longview Form (.lvfrm) document and supporting procedure used to create the sample form displayed at the beginning of this chapter, plus sample OnChange and OnClick procedures.

Note: These code blocks are intended as examples only and are not exhaustive. The examples listed here do not include every line of code required to create a Longview App and a complex Longview Form.

Sample Longview Form document

Syntax:

```
//Define the properties of the form - specify window title, form width, and
buttons

Window "Title:New Employee", Width:500
Button "Add", "Text:Add Employee", Validate:TRUE, "OnClick:button.lvpro"
Button "Cancel", "Text:Cancel"


//Define formats - specify the way date values are returned to AF (display =
user's regional settings)

DateFormat "DD-MM-YYYY"

//Display information - add main heading

Heading "Add an employee", Style:Title


//Add interactive controls - request employee's first and last name

TextBox "Employee's First Name", sFirstName, "OnChange:SetName.lvpro",
MaxLength:200, Required:TRUE

TextBox "Employee's Last Name", sLastName, "OnChange:SetName.lvpro",
MaxLength:200, Required:TRUE


//Add static control - display max allowed vacation days

Number "Maximum vacation days allowed", 25


//Add interactive controls - request start date and dynamically display
vacation days retro to start date

DatePicker "Start date", sStart, "OnChange:SetStartDate.lvpro", Min:2000-01-
01, Max:2016-01-01

NumberBox "Number of vacation days", nVacation


//Add static control - pad with space
```

```
Separator BLANK


//Display information - add section 2 heading

Heading "Section 2: Employee Information", Style:Subtitle


//Add prompt controls - request employee's resume and currency

FileChooser "Upload resume", sResume, "DefaultPath:C:\Users",
"FileRestrictions:pdf file|*.pdf|text file|*.txt", "BrowseType:SAVE",
"OnChange:SetResume.lvpro", Required:TRUE

SymbolSelector "Select the employee's currency", sCurrency, CURRENCIES,
Symbols:Source_Currencies#99, AllowLeaf:TRUE, AllowParent:TRUE,
AllowReadOnly:TRUE, AllowMultiple:TRUE, "OnChange:SetSymbols.lvpro",
Required:TRUE


//Display information and add interactive controls - use heading to provide
text instructions for check box controls

Heading "Select any of the following if applicable to the new employee:",
Style:Normal
Checkbox "Add to company email list", sEmailList,
"OnChange:SetEmaiList.lvpro"
Checkbox "Referred by internal employee", sReferred


//Add static control and add interactive controls - divide sections and
request employee's role, salary range, and employee type

Separator LINE, "Text:Role Information"

ComboBox "Role", sRole, "QA|DOC|SD|LC", "Text:QA Analyst|Technical
Writer|Software Developer|Learning Consultant", "OnChange:SetManager.lvpro"

ComboBox "Salary range", sSalary, "1|2|3|4", "Text:35,000 - 49,999|50,000 -
59,999|60,000 - 79,999|80,000 - 100,000", "OnChange:SetSalary.lvpro"

RadioGroup "Type of employee", sEmpType, "FT|PT", "Text:Full-time
Employee|Part-time Employee", "OnChange:SetEmpType.lvpro"


//Add static control - show employee's manager after the employee's role is
selected
Text "Employee's manager", sEmpManager
```

Sample supporting procedure

Syntax:

```
//Create required variables

create Variable nVacation as NUM

set Variable nVacation = 10
```

```
create Variable nRetrodays as NUM
create Variable sFirstName as STRING
create Variable sLastName as STRING
create Variable sDescription as STRING
create Variable sStart as STRING
create Variable sEmpManager as STRING
create Variable sEmailList as STRING
create Variable sReferred as STRING
create Variable sRole as STRING
create Variable sSalary as STRING
create Variable sEmpType as STRING
create Variable sResume as STRING
create Variable sCurrency as STRING

//Open the form
Show FORM using "SampleFormDocument.lvfrm"
```

Sample OnChange validation procedure

You could create an OnChange procedure such as the following to validate, for example, that a user has entered an email address in the correct format.

Syntax:

```
If StrContains ("$sEmail$", "@")
Set Variable LVG_FormValid = 1
Else
Set Variable LVG_FormValid = 0
Set Variable LVG_FormMessage = "Your email address must include the @ sign."
End if
```

Sample OnChange dynamic procedure

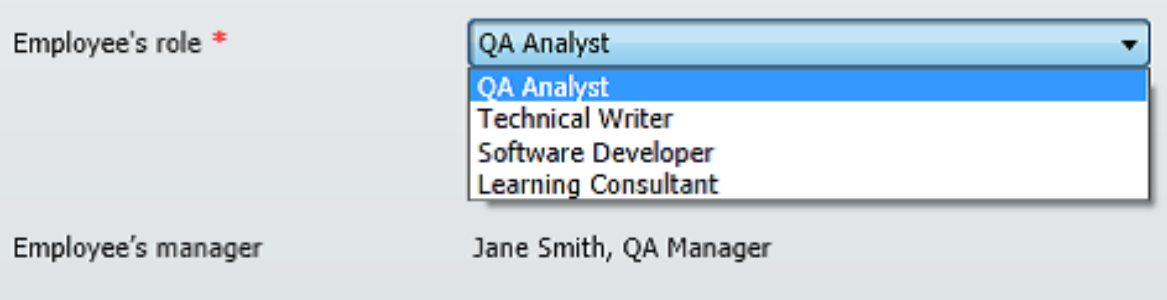
Consider the following syntax examples for the ComboBox and Text functions:

Syntax:

```
ComboBox "Role", sRole, "QA|DOC|SD|LC", "Text:QA Analyst|Technical
Writer|Software
```

```
Developer|Learning Consultant", "OnChange:SetManager.lvpro"
Text "Employee's manager", sEmpManager
```

You could write the SetManager procedure such as the following so that the sEmpManager variable used in the Text function updates dynamically. After the forms user selects the new employee's role in the combo box control, the employee's manager displays in the text area.



Sample OnChange dynamic procedure Syntax:

```
if $sRole$ EQ "QA"

set variable sEmpManager = "Jane Smith, QA Manager"
End If
if $sRole$ EQ "DOC"
set variable sEmpManager = "Liz Clark, Documentation Manager"
End If
if $sRole$ EQ "SD"
set variable sEmpManager = "Mary Scratch, Software Development Manager"
End If
if $sRole$ EQ "LC"
set variable sEmpManager = "Jim Coles, Director of Learning"
End If
```

Sample OnClick procedure

Syntax:

```
MAINTENANCE ON

SET VARIABLE sDescription = $sFirstName$ + $sLastName$
```

```
CREATE SYMBOL EMPLOYEES $sFirstName$ $sDescription$ Standard Manual Neither
PARENT

AllEmployees "0"

MAINTENANCE OFF
```

Developing HTML Pages

You can create custom .html pages to control the look and feel of your Longview App, to display information, and to gather information for setting up variables. You can call these .html pages from your main procedure using the Show HTML function.

For more information on this function, see Show HTML.

Displaying the Symbol Selector within HTML pages

You can add JavaScript code to an .html page to display a Symbol Selector in any Longview Apps that you create. You can optionally include up to two attribute filters so that users are presented with a filtered list of symbols for selection.

You can use the following JavaScript code to add the Symbol Selector to a page:

```
function showSymbolSelector(dimension, spec, initial, allowLeaf,
allowParent, allowReadOnly, allowMultiple, attributefilters)

{

return window.external.ShowSymbolSelectorAdvancedView(dimension, spec,
initial, allowLeaf, allowParent, allowReadOnly, allowMultiple,
attributefilters);
}
```

where:

- dimension is the dimension that should appear in the Symbol Selector.
- spec is the level of symbols to display in the Symbol Selector hierarchy, in relation to a specified symbol name, for example SymbolName### or SymbolName#99.
  For more information on symbol hierarchy specifications, see Symbol hierarchies.
- initial is the initial symbol selected in the Symbol Selector.
- allowLeaf specifies whether users can or cannot select leaf symbols and can have a value of true or false.
- allowParent specifies the type of parent symbols that users can select and can have one of the following values:

Value	Description
all	Users can select all parent symbols in the Symbol Selector dialog.
none	Users cannot select any parent symbols in the Symbol Selector dialog.
Static	Users can select static parent symbols only in the Symbol Selector dialog.

- allowReadOnly specifies whether users can select read-only symbols and can have a value of true or false.

- allowMultiple specifies whether users can select multiple symbols and can have a value of true or false.
- attributefilters can be up to two attribute filters linked by AND or OR, enclosed in double quotations marks, using the syntax `FilterType{AttrName{Operation{Expression`

  where:

Field	Description
FilterType	specifies the method to use to search the hierarchy for symbols matching the filter criteria and can be one of ALL, PARENT, LEAF, or ROOT. If you specify two attribute filters, FilterType must be the same for both filters.
AttrName	is the name of an attribute.
Operation	can be EQ for exactly equal to or NE for not equal to.
Expression	is a character string. If the expression contains spaces, enclose the expression in double quotation marks preceded by a backslash (\"expression with spaces\"). If the expression is a list, separate multiple items with a pipe (|). For Non-List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches only if the attribute is an exact match of the expression. ■ Attribute NE Expression — Matches if the attribute is not an exact match of the expression. For List Attributes, the filter behaves as follows: ■ Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression. ■ Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

  If you specify two attribute filters, enclose each filter in parentheses, for example " (attributefilter1) AND (attributefilter2)".

  If you don't include an attribute filter, use two double quotation marks ("").

The JavaScript below shows a sample script that calls the showSymbolSelector Java Script function and updates a text field. To access the selected symbols from a Symbol Selector, use the SelectedSymbols property.

```
function invokeSymbolSelector(dimension, tag)

{
```

```
var symbolSelector = showSymbolSelector(dimension, "",
document.getElementById(tag).value, true, all, false, true, "LEAF
{ZGPNativeCurrency{EQ{CAD");
if (symbolSelector.DialogResult == true)
{
document.getElementById(tag).value = symbolSelector.SelectedSymbols;
}
}
```

The following code is an example of an entire .html page that includes a Symbol Selector:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/
DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Symbol Selection</title>
<script language="javascript" >
function showSymbolSelector(dimension, spec, initial, allowLeaf,
allowParent, allowReadOnly, allowMultiple, attributefilter)
{
return window.external.ShowSymbolSelectorAdvancedView(dimension, spec,
initial, allowLeaf, allowParent, allowReadOnly, allowMultiple,
attributefilters);
}
function invokeSymbolSelector(dimension, tag)
{
var symbolSelector = showSymbolSelector(dimension, "",
document.getElementById(tag).value, true, all, false, true, "LEAF
{ZGPNativeCurrency{EQ{CAD");
if (symbolSelector.DialogResult == true)
{
document.getElementById(tag).value = symbolSelector.SelectedSymbols;
}
}
</script>
</head>
<body>
```

```
<h3>Select Some Symbols!</h3>

<form name="TestForm">

<div class="form-content">

<table>

<tr>

<td>Accounts:</td>

<td><input type="text" name="accountsTextBox" value="" id="LV_Accounts"
/><Button onclick="javascript:invokeSymbolSelector('Accounts', 'LV_
Accounts');">...</Button></td>

</tr>

<tr>

<td>Time Periods</td>

<td><input type="text" name="timePeriodsTextBox" value="" id="LV_TIMEPER"
/><Button onclick="javascript:invokeSymbolSelector('TIMEPER', 'LV_
TIMEPER');">...</Button>

</td>

</tr>

<tr>

<td>Entities</td>

<td><input type="text" name="entitiesTextBox" value="" id="LV_Entities"
/><Button onclick="javascript:invokeSymbolSelector('Entities', 'LV_
Entities');">...</Button>

</td>

</tr>

<tr>

<td>Currency</td>

<td><input type="text" name="currenciesTextBox" value="" id="LV_Currency"
/><Button onclick="javascript:invokeSymbolSelector('Currency', 'LV_
Currency');">...</Button></td>

</tr>

</table>

</div>

</form>

</body>

</html>
```

Displaying the file browser within HTML pages

You can add JavaScript code to an .html page to display a file browser in any Longview Apps that you create.

Use the following JavaScript function to add the file browser to a page:

```
function ShowFileDialog()

{
return window.external.ShowFileChooser();

}
```

You can add other parameters that allow you to customize your file browser. For example:

To...	Syntax
Restrict the file types available in the drop-down list	ShowFileChooser("Description|*.Type| Description2|*.Type2...") where: ■ Description is the description of the file type to display in the drop-down list of the file browser dialog. ■ Type is the file type to include. You can include any file type. Separate multiple entries with a pipe (|). If you want to specify subsequent parameters, and you want to leave this parameter blank, you must use two double quotation marks ("").
Create the file browser as an Open or a Save dialog	ShowFileChooser("", "Open|Save") If you specify Open, Open appears on the window title and the button. If you specify Save, Save As appears on the window title and Save appears on the button. If you do not specify this parameter, the file browser is created as an Open dialog. If you want to specify subsequent parameters, and you want to leave this parameter blank, you must use two double quotation marks ("").
Specify the default path for the file browser	ShowFileChooser("", "", "C:\\temp\\filePath"); You must escape any file separators with a backslash (\ ). If you do not specify this parameter, the file browser opens at the current directory for Longview Apps (typically MyDocuments\Longview).

The JavaScript below shows a sample script that calls the file browser Java Script and updates a text field. To access the return value of the file browsers, use the .FullPath property.

```
function ShowFileDialog()
```

```
{
return window.external.ShowFileChooser("csv file|*.csv | text file (*.txt) |
*.txt", "Open", "C:\\temp\\filePath");
}
function InvokeFileDialog(tag)
{
var selected = ShowFileDialog();
if (selected.DialogResult == true)
{
document.getElementById(tag).value = selected.FullPath;
}
}
```

Displaying the Print dialog within HTML pages

You can add JavaScript code to an .html page and hook it to a link, button, or other interface element to display the standard Internet Explorer Print dialog in any Longview Apps that you create.

Use the following JavaScript function to invoke the Print dialog:

```
function showPrintDialog()

{
window.external.ShowPrintDialog();

}
```

The following code shows an example of a Print dialog that is invoked when a user clicks a button in an .html form.

```
<input type="submit" id="Print" value="PRINT"
onclick="javascript:showPrintDialog()"/>
```

Passing variables between Longview Apps and Longview Application Framework

You can pass variable values between the .html pages of your Longview App and the underlying Application Framework code to create reusable Longview Apps. Use the variables that you create in Application

Framework along with the id attribute to pass variable values between Longview Application Framework and .html forms for the following elements in your .html pages:

- input
- select
- textarea

Note: For all other .html elements, you must use JavaScript to get and set the variable values correctly.

The element id that you use should be the corresponding Application Framework variable name, prefixed with LV_.

For example, if you have a variable called Location, the element id should be LV_Location.

The following Application Framework code snippet shows an example of how to create a variable to use within a page called Sample.html and how the variable value can be used after the user makes a selection on the .html page.

```
// Initialize
Create Variable Location AS STRING
SET VARIABLE Location = "E11111"
Create Variable Msg AS STRING
// Prompt user for input
Show HTML "Sample.html" 600 200
// echo user's selection
Set Variable Msg = "You selected " + $Location$
SHOW MESSAGE $Msg$
```

The following code is an example of an .html form that includes an Application Framework variable that will be resolved when the .html page is rendered and then passed back to Application Framework once the user clicks on a button on the .html page.

```
<form name="TestForm">
<div class="form-content">

<fieldset>
<legend>Select a Location</legend>
<label for="LV_Location">Location</label>
<select id="LV_Location" name="LV_Location">
<option value="E11111">E11111 - Toronto Manufacturing</option>

<option value="E11112">E11112 - Toronto Finance</option>
```

```
<option value="E11113">E11113 - Toronto Marketing</option>
<option value="E11114">E11114 - Toronto Admin</option>
</select>
</fieldset>

<button type="submit" value="Submit">Submit</button>

</div>
</form>
```

Using the LV_onclose() JavaScript function

You can use the LV_onclose() JavaScript function if you need to manipulate and/or send values back to Application Framework when a user exits an .html form by clicking the browser window close button (the x in the top right of the window).

LV_onclose() is an optional function that you can use when the OnClose command is used in the procedure that issues the ShowHTML command. The close button is enabled on the browser window displaying the .html file when the OnClose command is used. This gives the user an alternate way to exit the .html and, if the user exits the .html by clicking the close button, you can use the LV_onclose() function to help manipulate and/or submit values to Application Framework.

When a user clicks the browser window close button, Longview Smart Client:

- executes the LV_onclose() function, if it exists
- transfers the values set for the variables in the .html form back to Application Framework
- runs the specified OnClose procedure and terminates the application

Example

The following code is an example of how to use the LV_onclose() function in an external JavaScript file.

```
function LV_onclose() {
document.getElementById("LVS_buttonClicked").value = "CLOSE";
}
```

Using the LV_onload() JavaScript function

You can use the LV_onload() JavaScript function if you need Application Framework variable values for items other than input, select, and textarea elements within your .html page. You can write some JavaScript to help retrieve and set those values.

LV_onload() is an optional function that you can use to help retrieve and set values whenever you need to reference any Application Framework variables in your .html in areas other than input, select, and textarea elements. When the Show HTML command is called from Longview Application Framework, Longview Smart Client:

- gets the related .html
- populates the value attribute for any input, select, and textarea elements where the id is LV_ variable and variable is the corresponding Application Framework variable's value
- executes the LV_onload() function, if it exists, before showing the .html
- populates the value attribute for any input, select, and textarea elements where the id is LV_ variable and variable is the corresponding Application Framework variable's value that were created during the LV_onload()

The regular onload() JavaScript function runs before the Application Framework variables populate the input controls on the form, so if you need to use the values of Application Framework variables in your onload logic, you must use LV_onload() instead.

Example

The following code is an example of how to use the LV_onload() function in an external JavaScript file.

```
function LV_onload() {

var location = document.getElementById('LV_Location').value;

if (location == 'E11111') {
document.getElementById('Regular_Div').style.cssText = "display: none;";
document.getElementById('Special_Div').style.cssText = "display: block;";
document.getElementById('SpecialInstructions').innerHTML = 'Special
instructions only shown for E11111';
} else {
document.getElementById('Regular_Div').style.cssText = "display: block;";
document.getElementById('Special_Div').style.cssText = "display: none;";
document.getElementById('SpecialInstructions').innerHTML = 'Standard
instructions for all other entities';
}
}
```

The following code is an example of how to change the way the .html page is rendered using the Application Framework variable Location and calling the JavaScript above.

```
<html>

<head>
<script type="text/javascript" src="http://<%[[WEBSERVER]]%><%
[[WEBBRIDGE]]%>?LongviewIdentifier=<%[[WEBVARIABLE,
```

```
LongviewIdentifier]]%>&LongviewAction=GetAppFile&LongviewContentType='applic
ation/x-
javascript'&LongviewFileName='Sample1/js/ShowInstructions.js'"></script>
</head>
<body>
<h2 id="SpecialInstructions"></h2>
<form name="SampleForm" action="">
<div id="content">
<!-- Hidden field to get the location value -->
<textarea style="display: none;" name="LV_Location" id="LV_Location"
/></textarea>
</div>
<div id="Special_Div">
<!-- some special HTML to show only for E11111 -->
</div>
<div id="Regular_Div">
<!-- some regular HTML to show for all other entities -->
</div>
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

Maintaining Longview Client

Longview Client is the portal through which users can access Longview reports, Longview Apps, and other Longview components in one interface. After Longview Client is implemented for your system by Longview Professional Services, you can continue to maintain the files to keep your system up to date.

Understanding Longview Client

Before you perform any maintenance tasks for Longview Client, you should understand the following concepts:

- Understanding client files
- Locating client files
- Understanding Longview GUIDs
- Using special characters
- Displaying items conditionally in the navigation pane
- Showing or hiding file extensions
- Showing category short names

Understanding client files

The backbone of Longview Client is a series of XML files. Longview Client files can have one of the following extensions:

- .lvclient – referred to as the client file, this is the file that is initially loaded by the client to determine the navigation layout
- .lvmodule or .xml file – referred to as a module file, this file adds additional content in the form of a module or category in the client. These files are often added via the <include> markup to allow them to be re-used across different client files.

The main file is the client file, which includes or “calls” at least one module file. Each module file included in the main client file represents a tab in Longview Client. The client file includes elements to reference up to 20 module files via system attributes (10 general client modules and 10 system client modules). The attribute values depend on your system type.

Framework Client Configuration

Attribute	Value
ASLVClientOptions	Solutions\modules\DefaultOptions.xml
ASLVClientModule01	Solutions\modules\LongviewCPM.xml
ASLVClientModule02 to ASLVClientModule10	

Attribute	Value
ASLVClientSystem01	Solutions\modules\LongviewDesigner.xml
ASLVClientSystem02	Solutions\modules\LongviewDashboardDesigner.xml
ASLVClientSystem03 to ASLVClientSystem08	
ASLVClientSystem09	Solutions\modules\MyComponents.xml
ASLVClientSystem10	

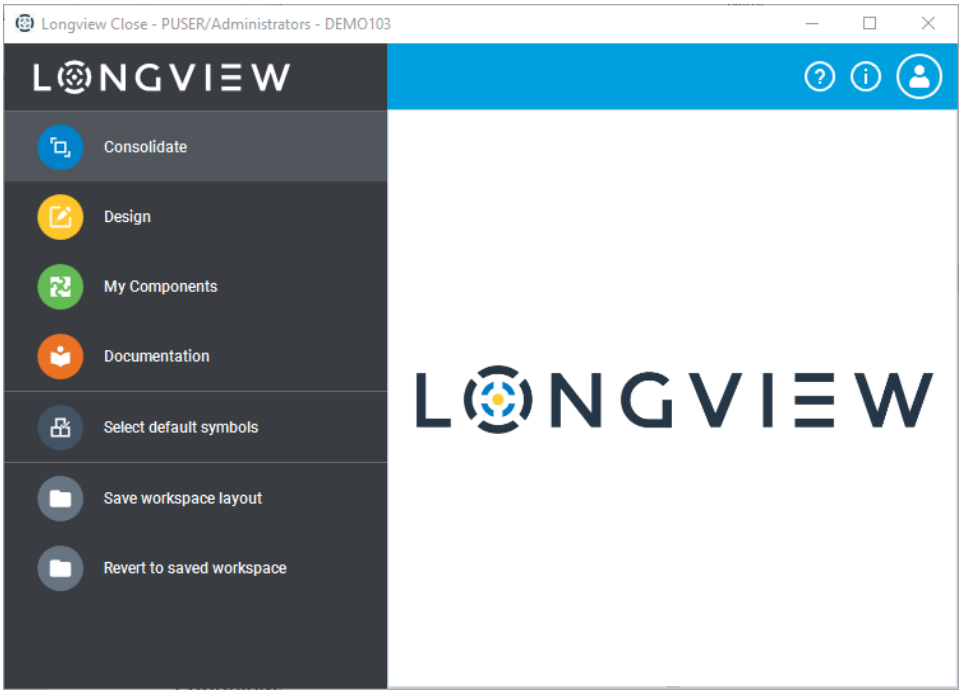
Longview Close Client Configuration

Attribute	Value
ASLVClientOptions	Solutions\modules\Options.xml
ASLVClientModule01	Solutions\modules\Consolidation.xml
ASLVClientModule02 to ASLVClientModule10	
ASLVClientSystem01	Solutions\modules\LongviewDesigner.xml
ASLVClientSystem02	Solutions\modules\LongviewDashboardDesigner.xml
ASLVClientSystem03 to ASLVClientSystem08	
ASLVClientSystem09	Solutions\modules\MyComponents.xml
ASLVClientSystem10	

Longview Tax Client Configuration

Attribute	Value
ASLVClientOptions	Solutions\modules\TaxOptions.xml
ASLVClientModule01	LongviewTaxProvision.lvmodule (if you have tax provision)
ASLVClientModule02	LongviewTaxPlanning.lvmodule (if you have tax planning)
ASLVClientModule03	LongviewTaskManagement.lvmodule (if you have task management)
ASLVClientModule04	LongviewGlobalTransparency.lvmodule (if you have global transparency)
ASLVClientModule05	Solutions\modules\LongviewTransferPricing.xml (if you have transfer pricing)
ASLVClientModule06 to ASLVClientModule10	
ASLVClientSystem01	Solutions\modules\LongviewDesigner.xml
ASLVClientSystem02	Solutions\modules\LongviewDashboardDesigner.xml
ASLVClientSystem03 to ASLVClientSystem08	
ASLVClientSystem09	Solutions\modules\MyComponents.xml
ASLVClientSystem10	

In the following graphic, four modules are shown: Consolidate, Design, My Components, and Documentation.



Caution: Do not modify the out-of-box client or module files.

In XML, elements contain attributes with name/value pairs, such as the following:

```
<Include FileName="Consolidation.lvmodule"/>
```

where:

- Include is an XML element
- FileName is the name portion of an XML attribute.
- Consolidation.lvmodule is the value portion of the FileName XML attribute. The value may also be determined by an attribute as follows:

```
<Include FileName="[[SYSTEM,ASLVClientModule01,DBDEFAULT]]"/>
```

As an administrative user of Longview Client, all XML attributes you can modify are optional. Where possible, this chapter lists available XML attributes, and you can choose to include or exclude them as appropriate to your system.

Locating client files

Typically, Longview Client files are stored within applications folder on the Data Server working directory, such as `C:\Longview\DataServers\<LID>\applications`; however, depending on

your system implementation, your files may be in an alternate location or folder.

Depending on your system you will have different out-of-box files:

Product	File	Location
All	Longview.lvlicent	applications
Longview Close	Administration.xml	applications\Solutions\modules
Longview Close	Consolidation.xml	applications\Solutions\modules
Longview Close	Forecast.xml	applications\Solutions\modules
Longview Close	Options.xml	applications\Solutions\modules
Longview Close	Planning.xml	applications\Solutions\modules
Longview Tax	LongviewTaxProvision.lvmodule	applications
Longview Tax	LongviewTaxProvision.lvmodule	applications
Longview Tax	LongviewTaxPlanning.lvmodule	applications
Longview Tax	LongviewTaskManagement.lvmodule	applications
Longview Tax	LongviewGlobalTransparency.lvmodule	applications
Longview Tax	LongviewTransferPricing.xml	applications\Solutions\modules
Solutions Framework	LongviewCPM.xml	applications\Solutions\modules
All	LongviewDesigner.xml	applications\Solutions\modules
All	LongviewDashboardDesigner.xml	applications\Solutions\modules
All	MyComponents.xml	applications\Solutions\modules

Caution: Do not modify the out-of-box client or module.

Understanding Longview GUIDs

Typically, a GUID (short for “globally unique identifier”) is a unique reference number used as an identifier. For Longview Client files, Longview GUIDs are alphanumeric and follow the naming conventions created by Longview Professional Services. As you add or modify items in client and module files, it is best practice to adhere to the Longview GUID naming conventions already in place. Regardless of the naming convention, every Longview GUID you include in your series of client and module files must be unique.

Using special characters

In XML, you must use XML character entities for the system to read special characters within your strings. Review the following table for a list of special characters and the way to include them within strings in your client and module files.

Special character	XML character entity to use
double quotation mark "	&quot;
single quotation/apostrophe ’	&apos;
left angle bracket <	&lt;

Special character	XML character entity to use
right angle bracket >	&gt;
ampersand &	&amp;

Displaying items conditionally in the navigation pane

As an administrative user, you can choose to display items in the navigation pane conditionally. You may want to, for example, display an item in the navigation pane based on a Longview attribute value using the Show XML attribute. The default if the Show XML attribute is not included in your series of client and module files is True, and all items display in the navigation pane. If the expression for the Show XML attribute is invalid, the system treats the expression as True and all items display in the navigation pane.

You can use the Show XML attribute for the following items:

- Folder elements.
- Link elements. For more information, see Working with links.
- Symbol selector elements. For more information, see Working with symbol selectors.

Syntax

To hide an item in the navigation pane, include the following at the end of the relevant line in your client or module file:

```
<... Show="Expression"/>
```

where:

- Expression is one of: True, False, or an attribute token.

Examples

```
<Folder Name="My Reports" Show="False">

<LongviewDocs Guide="LVWorkflow.pdf" Show="
[[System,ASWorkflowInUse,DBDefault]]"/>

<LVApp Name="This is a Test" AppName="HelloWorld" GUID="LongviewCPM_App001"
Show="[[USER, AUPowerUser, THIS]]"/>
```

Showing or hiding file extensions

Depending on your system implementation, file extensions may be shown or hidden by default in the navigation pane for Longview Apps and Longview reports. As an administrative user, you can override the default setting at the folder level or link level by adding the ShowExtensions XML attribute. The

default if the ShowExtensions XML attribute is not included in your series of client and module files is True, and file extensions display in the navigation pane.

Syntax For Folders

```
<Folder Name="Name" ShowExtensions="Value">
```

where:

- Name is the name of the folder to display, enclosed in double quotation marks.
- Value is whether to show or hide file extensions for all relevant items in the folder and can be TRUE to show file extensions or FALSE to hide file extensions.

Examples

```
<Folder Name="My Reports" ShowExtensions="True">


<Folder Name="My Reports" ShowExtensions="False">
```

Syntax For Links

```
<ItemType ItemName="MyItem.ext" GUID="Id" ShowExtensions="Value"/>
```

where:

- ItemType is type of item to link to and can be LVApp or ReportTemplates.
- ItemName is the name of the item to include and can be AppName for a Longview App or FileName for a Longview report.
- MyItem.ext is the name and relevant extension for the item to link to, enclosed in double quotation marks.
- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.
- Value is whether to show or hide file extensions for all relevant items in the folder and can be TRUE to show file extensions or FALSE to hide file extensions.

Examples

```
<LVApp AppName="NewEmployee.lvapp" GUID="LongviewCPM_NewEmp"
ShowExtensions="True"/>
```

```
ShowExtensions="True"/>

<ReportTemplates FileName="MyReport.rtp" GUID="LongviewCPM_rtp001"
ShowExtensions="True"/>
```

Syntax for Longview categories

```
<ListCategory Name="App Category Name" Types="Types" GUID="Id"
ShowExtensions="Value"/>
```

where:

- App Category Name is the name of the Longview category as created Longview Analysis and Reporting, enclosed in double quotation marks.
- Types is a pipe (|) delimited list of items assigned to the category and can be Report and/or App, enclosed in double quotation marks.
- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.
- Value is whether to show or hide file extensions for all relevant items in the folder and can be TRUE to show file extensions or FALSE to hide file extensions.

Examples

```
<ListCategory Name="Admin" Types="Report|App" GUID="LongviewCPM_AdminCatExt"
ShowExtensions="True"/>
```

Note: Longview does not support the modification of client and module files outside of the tasks outlined in this chapter.

Showing category short names

Depending on your system implementation, category short names may be shown or hidden in the navigation pane. The category short name is short text that can be displayed under the category image for ease of identifying categories quickly. By default, the ShowCategoryShortNames XML attribute is not included in your client file, which will use its default value of False. In this case, only the image for each category will be displayed in the navigation pane. To enable the display of category short names, the ShowCategoryShortNames XML attribute must be added to the <UI> element in your client or module file.

Syntax:

```
<UI ShowCategoryShortNames="Value">
```

where:

- Value is whether to show category short names if they are defined. Valid values are TRUE or FALSE. The short name for a category will only render if the ShortName XML attribute for the category element is also defined. For more information, see Understanding category elements.

Examples

```
<UI ShowCategoryShortNames="true">
```

Debugging Longview Client Launch Issues

If the Longview Client fails to launch and you are having difficulty discovering the reason you can enable detailed logging to help solve the issue.

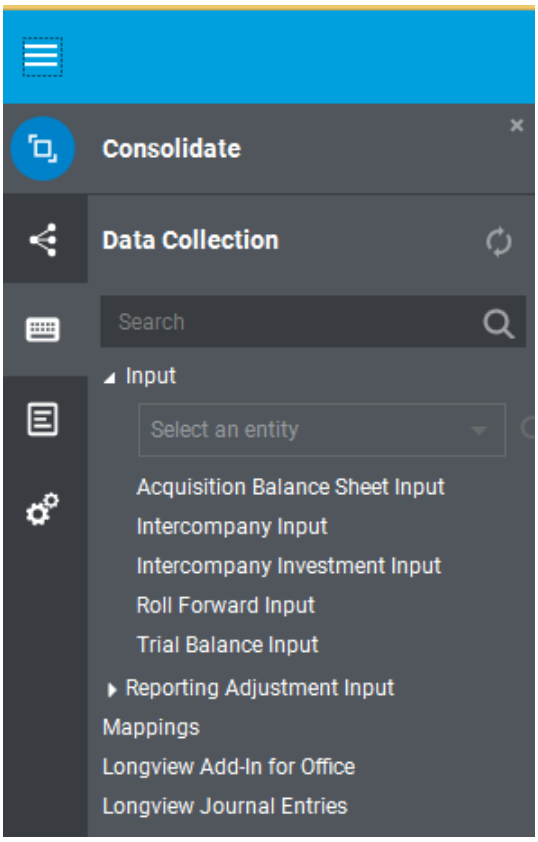
To enable detailed logging:

1. Navigate to the local Longview folder (Documents\Longview\<Identifier>.
2. Open the logs folder. If there is no logs folder create one.
3. Create a new text file named “LongviewClient.log”.
4. Launch the Longview Client.
5. Review the contents of the log file.

Configuring the Longview client

You can customize the Longview Client to specify what will be displayed. This includes the ability to customize:

- The modules or “circles” at the top left of the navigation pane, and within the menu icon.
- The categories within the navigation pane.
- Folders and Links within the categories.



Understanding the client file

The client file uses the 'Include Filename' tag to include modules in the Longview Client. These modules are defined using system attributes.

Examples

```
<client>
    <Config>

<description>[[SYSTEM,ASLVClientDescription,DBDEFAULT]]</description>

 <procedure>[[SYSTEM,ASLVClientProcedure,DBDEFAULT]]</procedure>

<rootpath>[[SYSTEM,ASLVClientRootPath,DBDEFAULT]]</rootpath>

 <userid/>

<groupid/>

 <serverid/>
```

```
<defaulthelp>[[SYSTEM,ASLVClientDefaultHelp,DBDEFAULT]]</defaulthelp>
 <helpmappings>[[SYSTEM,ASLVClientHelpMappings,DBDEFAULT]]</helpmappings>
 </Config>
<Include FileName="[[SYSTEM,ASLVClientOptions,DBDEFAULT]]"/>
<UI ShowCategoryShortNames="true">
 <Include FileName="[[SYSTEM,ASLVClientModule01,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule02,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule03,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientModule04,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientModule05,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule06,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule07,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule08,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientModule09,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientModule10,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientSystem01,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientSystem02,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientSystem03,DBDEFAULT]]"/>
 <Include FileName="[[SYSTEM,ASLVClientSystem04,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientSystem05,DBDEFAULT]]"/>
<Include FileName="[[SYSTEM,ASLVClientSystem06,DBDEFAULT]]"/>
```

```
<Include FileName="[[SYSTEM,ASLVClientSystem07,DBDEFAULT]]"/>

 <Include FileName="[[SYSTEM,ASLVClientSystem08,DBDEFAULT]]"/>

<Include FileName="[[SYSTEM,ASLVClientSystem09,DBDEFAULT]]"/>

 <Include FileName="[[SYSTEM,ASLVClientSystem10,DBDEFAULT]]"/>

</UI>
</client>
```

These system attributes can be adjusted to point to any custom module file if required.

Longview Close and Plan modules

Longview Close and Plan includes 6 out-of-box module files:

Module	Name
	Consolidate
	Plan
	Design
	Dashboard Designer
	My Components
	Documentation

Longview Tax modules

Longview Tax includes out-of-box module files. Only module files to which you have purchased a license will be visible in your Longview Tax client

Module	Name
	Tax Provision
	Tax Planning
	Task Management
	Global Transparency
	Transfer Pricing
	Design
	Dashboard Designer
	My Components
	Documentation

Solutions Framework modules

Solutions Framework includes out-of-box module files:

Module	Name
	Consolidate
	Design
	Dashboard Designer
	My Components

Module	Name
	Documentation

Customizing the client file

If the content within the Longview.lvclient file needs to be modified, a custom client file will need to be created. Since the content is full attribute driven the client file should only be updated if you need more than 20 module files. Otherwise, you should modify the system attributes to point to custom module and options files as required.

Note: The standard client file provided should not be modified or renamed. Instead, a new client file should be created, and the content updated accordingly. This will avoid the client file being overwritten when an upgrade is performed.

To customize the client file:

1. Navigate to the applications folder in your Data Server working directory. For example,

   `C:\Longview\DataServers\Longview\applications`.

2. Create a copy of the client file.
3. Rename the copy of the client file to a new custom name.
4. Update the system attribute SLVClientConfigFileName to point to the name of the new client file.
5. Edit the new client file as required. The Include FileName tags allow you to include modules within the Longview Client.

Understanding the Longview Close module files

Each of the module files (Consolidation.xml and Planning.xml) support configuration of folders and entities symbol selection without the need to customize the module files.

Each module contains the following categories:

1. Processes: holds process maps used to direct the user through various steps to complete system processes.
2. Data Collection: holds apps used to input or import data into the system.
3. Reporting: holds apps and reports used to review or analyze results.
4. Workflow: holds the application used for setting and reporting on the status of a data area in the Longview application and provides a mechanism to notify users of status.
5. Administration: holds apps and reports used to manage the system.

Within the processes, data collection and reporting categories is support for up to ten folders. These folders allow you to group related items into a logical collection. Each folder also contains an entity selector that allows the user to select an entity symbol to be used with each app or report within the

folder if activated. The contents of each folder are determined by the category a process map, app or report is published to.

The name of the folder and the entity symbol selection option is configured via the Configure Categories app.

The consolidate module

The consolidate module contains up to ten folders each for process maps, data collection and reporting. The Administration category is defined by a shared Administration.xml file used for both consolidation and planning.

Sample Data Collection Folder

```
<Folder Name="[[SYSTEM,ASlvclientCategoryCONSDC01,DBDEFAULT]]"
Expand="False">


<SymbolSelector Name="SymbolSelector"

Dimension="[[SYSTEM,SGPEntitiesDimension,DBDEFAULT]]"

    Symbolspec="[[SYSTEM,ASlvclientEntitySymbolSpecDC,DBDEFAULT]]"

    SelectionType="[[SYSTEM,ASlvclientCategoryCONSDC01ST,DBDEFAULT]]"

    MultipleSelection="False" ReadAccessSelection="False"

DefaultText="Select an entity" DefaultAttribute="UGPD2InputDefault"

    DefaultAttributeType="User" ContentType="NameAndDescription"/>

<LVApp Category="CONSDC01" GUID="CONSDC01" />

</Folder>
```

The planning module

The planning module contains a budget and forecast folder within the process maps, data collection and reporting categories. Within each folder up to ten sub-folders can be configured.

Sample Budget Data Collection Folder

```
<Folder Name="Budget" Expand="True">

  <Folder Name="[[SYSTEM,ASlvclientCategoryBDGTDC01,DBDEFAULT]]"
Expand="False">


 <SymbolSelector Name="SymbolSelector"


Dimension="[[SYSTEM,SGPEntitiesDimension,DBDEFAULT]]"
```

```
      Symbolspec="[[SYSTEM,ASlvclientEntitySymbolSpecDC,DBDEFAULT]]"

SelectionType="[[SYSTEM,ASlvclientCategoryBDGTDC01ST,DBDEFAULT]]"
      MultipleSelection="False" ReadAccessSelection="False"

DefaultText="Select an entity" DefaultAttribute="UGPD2InputDefault"

DefaultAttributeType="User" ContentType="NameAndDescription"/>

<LVApp Category="BDGTDC01" GUID="BDGTDC01" />

</Folder>
</Folder>
```

The administration category

The administration category is shared by the consolidation and planning modules and is defined in the Administration.xml file. The administration category defines several pre-defined folders to organize administration apps. Configured administration apps can be added to the administration category by publishing the app to specific categories.

Folder contents are specified using categories:

Published to category	Folder
MTCEAC	Accounts
MTCETP	Time Periods
MTCEEN	Entities
MTCECU	Currencies
MTCEOD	Other Dimensions
MTCEALLOC	Allocations
MTCECALC	Calculations
MTCEFX	Foreign Exchange
MTCERO	Rollover
MTCEIMPEXP	Import and Export
MTCESYS	System

Note: The contents of the Security and Monitoring folder are pre-defined and not configured by publishing to specific categories.

Understanding the Longview Tax module files

Each of the module files (LongviewTaxProvision.lvmodule, LongviewTaxPlanning.lvmodule, LongviewTaskManagement.lvmodule, LongviewGlobalTransparency.lvmodule and LongviewTransferPricing.xml) contain the following categories:

1. Processes: holds process maps used to direct the user through various steps to complete system processes.
2. Data Collection: holds apps used to input or import data into the system.
3. Reporting: holds apps and reports used to review or analyze results.
4. Workflow: holds the application used for setting and reporting on the status of a data area in the Longview application. This module is used for Tax Provisioning and Tax Planning modules only.
5. Administration: holds apps and reports used to manage the system.

Within the processes category there is support for up to ten folders. These folders allow you to group related items into a logical collection. Each folder also contains an entity selector that allows the user to select an entity symbol to be used with each app or report within the folder if activated. The contents of each folder is determined by the category a process map is published to.

The name of the folder and the entity symbol selection option is configured via the Configure Categories app.

Understanding the Solutions Framework module files

The consolidate module contains up to ten folders each for process maps, data collection and reporting. The Administration category is defined by a shared Administration.xml file used for both consolidation and planning.

The out-of-box module file (LongviewCPM.xml) contains up to ten folders each for process maps, data collection and reporting for each of Consolidation, and Planning. The Administration category is defined by the Administration.xml file. The file otherwise contains the same content as Longview Close and Plan modules.

Customizing the module files

If the content within any of the modules in the Longview Client need to be changed, custom module files will need to be created.

> **Note:** The standard module files provided should not be modified or renamed. Instead, a new module file should be created, and the system attributes updated accordingly. This will avoid any module files being overwritten when an upgrade is performed.

To create a custom module file:

1. Create a new module file under the applications folder in your Data Server working directory.

   > **Note:** A copy of a standard module can be used as a starting point for a custom module. See Locating client files to find the module file you want to customize.

2. Edit the new module file to include the required categories, folders, and links to be displayed in the Longview Client.
3. Update the system attributes to point to the customized module file.

Configuring the module files

Out-of-box module files are configured using the Configure Categories app.

To access configure categories:

1. Within the Design module, select the Administration category.
2. Click Configure Categories.
3. The Configure Categories table appears.

Configuring categories

Depending on your implementation the configure categories table will appear with:

- A combination of the Consolidate tab and the Budget and Forecast tabs.
- A combination of Tax Provision, Task Management, Global Transparency, Tax Planning and Longview Transfer Pricing

Note: Any changes made to existing categories will not be reflected until the navigation pane is refreshed.

Any changes made to existing categories will not be reflected until the navigation pane is refreshed. Each of the tabs works in the same way but configures different folders within the Longview client.

The configure category table contains the following columns:

Active	Indicates whether the publish category is active in the system.
ID	The ID of the publish category.
Category	The category in the client configuration file.
Folder Description	The description of the folder within the navigation. **Note:** ■ The folder displayed for active categories is defined by the folder description. Submit will update the category description and attribute value. ■ The name is limited to 100 characters and cannot contain the following characters: double quotes (“), less than (<), greater than (>) and ampersand (&).

Active	Indicates whether the publish category is active in the system.
Symbol Selection Type	The symbol selection allowed for the entity selector within the folder. Options include: ■ All: user is allowed to select a leaf or parent symbol ■ Leaf: user is only allowed to select a leaf symbol ■ Parent: user is only allowed to select a parent symbol ■ None: no symbol selector will be displayed for the folder

To activate a category:

1. Check the Active column.
2. Enter a description for the folder (mandatory).
3. Select the symbol selection type (mandatory).
4. Click Apply.

The category will be added to the list of publish categories available, and the related system attributes will be updated. The folder will appear in the client navigation once a process map, app or report is published to that category.

To deactivate a category:

1. Open Longview Analysis and Reporting.
2. Click on the Categories tab.
3. Select the category to deactivate.
4. Press the Delete key.

Note: You will only be able to delete a category that has no reports, process maps or apps published to it.

Module icons

If you are customizing your Longview modules, you may want to include a different module icon. The following icon set is provided:

Icon	Location
	Solutions\modules\icons\apps@2.png
	Solutions\modules\icons\budget@2.png
	Solutions\modules\icons\calendar@2.png

Icon	Location
	Solutions\modules\icons\cog@2.png
	Solutions\modules\icons\components_2@2.png
	Solutions\modules\icons\components_3@2.png
	Solutions\modules\icons\components_4@2.png
	Solutions\modules\icons\config@2.png
	Solutions\modules\icons\configure@2.png
	Solutions\modules\icons\consolidate@2.png
	Solutions\modules\icons\data_collection@2.png
	Solutions\modules\icons\design@2.png
	Solutions\modules\icons\edit@2.png
	Solutions\modules\icons\events@2.png
	Solutions\modules\icons\export@2.png
	Solutions\modules\icons\eye@2.png
	Solutions\modules\icons\forecast@2.png
	Solutions\modules\icons\foreign_exchange@2.png
	Solutions\modules\icons\global@2.png
	Solutions\modules\icons\home@2.png
	Solutions\modules\icons\import@2.png
	Solutions\modules\icons\itemize@2.png
	Solutions\modules\icons\library@2.png
	Solutions\modules\icons\longview_logo@2.png
	Solutions\modules\icons\manage_symbols@2

Icon	Location
	Solutions\modules\icons\monitoring@2.png
	Solutions\modules\icons\note@2.png
	Solutions\modules\icons\options@2.png
	Solutions\modules\icons\period_end@2.png
	Solutions\modules\icons\planning@2.png
	Solutions\modules\icons\process_maps_1@2.png
	Solutions\modules\icons\rollover@2.png
	Solutions\modules\icons\rollover_option_2@2.png
	Solutions\modules\icons\tax_provision@2.png
	Solutions\modules\icons\transfer_pricing@2.png
	Solutions\modules\icons\workflow@2.png

Working with categories

In Longview Client, folders contain groupings of links. Depending on the category and your system, you may see several folders within a category. You can click a folder to expand or collapse it in the navigation pane. You may also see a Symbol Selector above a grouping of folders or directly underneath a folder. You can select a symbol that applies to all links underneath the Symbol Selector.

In the following graphic, Review is a folder containing links to several Longview Apps and reports.

Understanding category elements

The <Category> element is a single XML element with the optional Name, Image, ShowExtensions and ShortName XML attributes. If you include a category without the Name XML attribute, “Untitled” displays as the category name. You must include at least one link for the category to display. Category elements must have a <Module> element as their parent.

Syntax:

```
<Category Name="My Category" Image="My Image.png" ShortName="My ShortName">

  <MyContent/>
</Category>
```

where:

- My Category is the Name XML attribute of the category enclosed in double quotation marks.
- My Image is the Image XML attribute of the category enclosed in double quotation marks.
- My Shortname is the ShortName XML attribute of the category enclosed in double quotation marks. This attribute is used in conjunction with the ShowCategoryShortNames XML attribute for the <UI> XML element. If the ShortName is defined and the ShowCategoryShortNames attribute is true, the value for My Shortname will appear under the image for the category. If My Shortname is too long, it will be displayed with ellipses (...). This feature can help users quickly identify the categories in the navigation pane. For more information, see Showing category short names.
- MyContent is the name of at least one folder or link to include within the category. For more information, see Adding folders and Adding links.

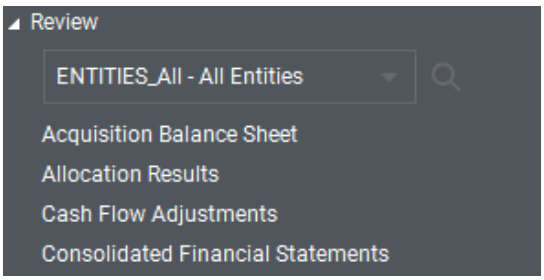
Examples

```
<Category Name="Period End Review" Image="Icons\Calendar.png" ShortName="End
Rev">

  <Folder Name="Review" Expand="False">
    <ListCategory Name="CONSR" Types="APP|REPORT" GUID="CONSR" />
  </Folder>
</Category>
```

Working with folders

In Longview Client, folders contain groupings of links. Depending on the category and your system, you may see several folders within a category. You can click a folder to expand or collapse it in the navigation pane. You may also see a Symbol Selector above a grouping of folders or directly underneath a folder. You can select a symbol that applies to all links underneath the Symbol Selector.

In the following graphic, Review is a folder containing links to several Longview Apps and reports.



Understanding folder elements

The <Folder> element is a single XML element with the optional Name, Expand and Style XML attributes. If you include a folder without the Name XML attribute, “Untitled” displays as the folder name. You must include at least one link for the folder to display. Folder elements must have a <Category> element or another folder as their parent.

Syntax:

```
<Folder Name="My Folder" Expand="Expand" Style="Heading">

  <MyLink/>
</Folder>
```

where:

- My Folder is the Name XML attribute of the folder to add or modify, enclosed in double quotation marks.
- Expand is whether the folder is initially expanded and can be True to expand or False to collapse the folder, enclosed in double quotation marks. The default if you do not include the Expand attribute is False.
- Heading specifies that the folder should be styled as a top-level folder. If the XML attribute Style is not present, the folder appears with normal styling.
- MyLink is the name of at least one link to include within the folder. For more information, see Adding links.

Example:

```
<Folder Name="Admin" Expand="True">

  <ReportTemplates Category="Admin" GUID="LongviewCPM_Admin_Reports"/>

</Folder>
```

Adding folders

You can add a folder to an existing category. A folder must contain at least one link for the folder to display. Folders appear in the navigation pane in the order they are listed in the module file.

To add a folder to an existing category:

1. Open the relevant module file in your preferred text editor.
2. Locate the category in which you want to add the new folder.
3. In the appropriate place, add the <Folder> element and include at least one link. For more information, see Understanding folder elements and Adding links.
4. Save your changes.
5. Launch Longview Client and verify that the new folder appears as expected.

Modifying folders

You can modify an existing folder.

To modify an existing folder:

1. Open the relevant module file in your preferred text editor.
2. Locate the <Folder> element you want to modify.
3. Make the necessary changes. For more information, see Understanding folder elements and Adding links.

   Note: If you remove the Name XML attribute of a folder, “Untitled” displays as the folder name.

4. Save your changes.
5. Launch Longview Client and verify that the new folder appears as expected.

Deleting folders

You can delete an existing folder.

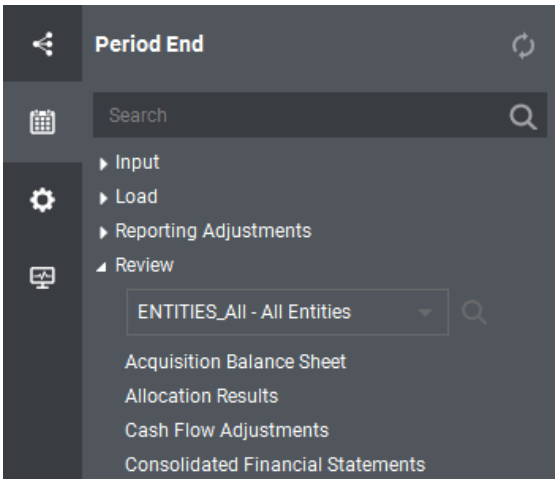
Caution: Deleting a folder removes all links contained within the folder. To restore a deleted folder, you must recreate the folder, including all links.

To delete an existing folder:

1. Open the relevant module file in your preferred text editor.
2. Locate the <Folder> element you want to delete.
3. Delete all content from the beginning of the relevant <Folder> element to the end of the relevant </Folder> element.
4. Save your changes.
5. Launch Longview Client and verify that the new folder does not appear.

Working with links

In Longview Client, links are typically grouped together in a folder but can also appear under a category. The following graphic shows several folders directly under a category Period End:



Understanding link elements

You can include links to the following items in Longview Client:

- Longview Apps
- Longview categories
- Longview components
- Longview Dashboard panels
- Longview Documentation
- Longview process maps
- Longview reports
- Longview tools and editors
- URLs

Link elements must have a <Category> or <Folder> element as their parent.

For Longview Apps and Longview reports, you can optionally show or hide file extensions. You can also specify whether to show or hide file extensions for all Longview Apps and Longview reports assigned to a Longview category. For more information, see Showing or hiding file extensions.

Longview Apps

You can create a link to a Longview App using the following code:

```
<LVApp AppName="AppName.lvapp" Name="Name" GUID="Id"/>
```

where:

- AppName is the name of the .lvapp to include, enclosed in double quotation marks.

  > **Note:** To include all Longview Apps assigned to a category, use the ListCategory element. For more information, see Longview categories.

- Name is the name of the link to display, enclosed in double quotation marks. This parameter overrides the optional ShowExtensions XML attribute, if included. For more information, see Showing or hiding file extensions.

- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

Example:

```
<LVApp AppName="NewEmployee.lvapp" Name="New Employee" GUID="LongviewCPM_
NewEmp"/>
```

Longview categories

You can create a series of links to all Longview Apps, and reports assigned to an existing category using the following code:

```
<ListCategory Name="App Category Name" Types="Types" GUID="Id"/>
```

where:

- App Category Name is the name of the Longview category as created in Longview Analysis and Reporting, enclosed in double quotation marks.

- Types is a pipe (|) delimited list of items assigned to the category and can be Report and/or App, enclosed in double quotation marks.

- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

Example:

```
<ListCategory Name="Admin" Types="Report|App" GUID="LongviewCPM_AdminCat"/>
```

Longview components

You can create links to all Longview components to which a user has access using the following code:

```
<LongviewComponents GUID="Id"/>
```

where:

- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

Review the following table for the code to use to create a link to a single Longview component.

Note: To include all Longview Apps assigned to a category, use the ListCategory Users can see links only to the components to which they have access.

Component	Syntax Options
Longview Add-In for Office	<AddinOffice /> <AddinOffice FileName="FileName.xlsx"/> <AddinOffice Name="Override Name" FileName="[server:]File.xlsx"/> where: ■ **FileName** is the name of the Longview Add-In for Office file to open, including the extension, enclosed in double quotation marks. ■ **server:** is an optional parameter and is used when launching Longview Add-In for Office files located in the Longview custom folder on the Longview data server. ■ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.
Longview Application Administrator	<ApplicationAdministrator/> <ApplicationAdministrator Name="Override Name"/> <ApplicationAdministrator Name="Override Name" Section="Section"/> where: ■ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks. For example, "Hierarchy maintenance". ■ **Section** is the page to display when Longview Application Administrator opens and can be one of the following: Attributes, Batches, Events, IntercompanyEliminations, JournalEntries, Locks, Rules, Schedules, SymbolAccess, UsersGroups, Symbols, Symbols|DimensionName (where DimensionName is the dimension page to display), enclosed in double quotation marks.

Component	Syntax Options
Longview Analysis and Reporting	<AnalysisReporting/> <AnalysisReporting Name="Override Name "/> where: ▪ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.
Longview Dashboard Designer	<DashboardDesigner/> <DashboardDesigner Name="Override Name "/> where: ▪ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.
Longview Journal Entries	<JournalEntries/> <JournalEntries Name="Override Name"/> where: ▪ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.
Longview Server Manager	<ServerManager/> <ServerManager Name="Override Name "/> where: ▪ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.
Longview Workflow Designer	<WorkflowDesigner/> <WorkflowDesigner Name="Override Name"/> where: ▪ **Override Name** is the name of the link to display if you do not want the component name to display, enclosed in double quotation marks.

For more information, see Adding links.

Examples

```
<LongviewComponents GUID="LongviewCPM_My_Components"/>
<AddinOffice Name="My Workbook" FileName="myworkbook.xlsx"/>
<AddinOffice Name="My Workbook" FileName="server:web/excel/workbook.xlsx"/>
<ApplicationAdministrator Name="Hierarchy Maintenance" Section="Symbols"/>
<AnalysisReporting Name="Launch Longview A&R"/>
```

```
<DashboardDesigner Name="Launch Longview Dashboard Designer"/>
<JournalEntries Name="Launch Longview JEs"/>
<ServerManager Name="Launch Longview Server Manager"/>
<WorkflowDesigner/>
```

Longview Dashboard panels

You can create a link to a Longview Dashboard panel using the following code:

```
<LongviewPanel Name="Name" Panel="Panel" GUID="Id"/>
```

where:

- Name is the name of the link to display, enclosed in double quotation marks.
- Panel is the name of the existing Longview Dashboard panel, enclosed in double quotation marks.
- Id is the Longview GUID for the link, following your system's naming conventions. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

Examples

```
<LongviewComponents GUID="LongviewCPM_My_Components"/>
<LongviewPanel Name="Workflow" Panel="Approval_Workflow" GUID="LongviewCPM_
WorkflowPanel"/>
```

Longview Documentation

You can create a link to a Longview Documentation PDF using the following code:

Note: Users can always see all documentation links, regardless of the components to which they have access.

```
<LongviewDocs Guide="Guide Name"/>
```

where:

- Guide Name is the name of the PDF to include, enclosed in double quotation marks, and can be one of the following:

LV4XLGuide	LVCompGuide	LVInstallationGuide
LVAddinForOffice	LVDashboardGuide	LVIntegrationGuide
LVAandRGuide	LVDesktopGuide	LVJEGuide
LVAppAdminGuide	LVDeveloperGuid	LVManagerGuide
LVCommandsGuide		LVWorkflowGuide

Note: To display all PDFs, exclude Guide="guide name", for example: <LongviewDocs/>.

For more information, see Adding links.

Example:

```
<LongviewDocs Guide="LVDeveloperGuide"/>

<LongviewDocs/>
```

Longview process maps

You can create a link to a Longview process map using the following code:

```
<ProcessMap Name="Name" ProcessID="ProcessID" GUID="Id"/>
```

where:

- Name is the name of the link to display, enclosed in double quotation marks.
- ProcessID is the ID of the process map to display, enclosed in double quotation marks.
- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

Examples

```
<ProcessMap Name="High-Level Budget Process" ProcessId="P1" GUID="PMap1"/>
```

Longview reports

You can create a link to a Longview report using the following code:

```
<ReportTemplates FileName="File.rtp" Name="Name" GUID="Id"/>
```

where:

- File is the name of the .rtp to include, enclosed in double quotation marks.

  > **Note:** To include all reports assigned to a category, use the ListCategory element. For more information, see Longview categories.

- Name is the name of the link to display, enclosed in double quotation marks. This parameter overrides the optional ShowExtensions XML attribute, if included. For more information, see Showing or hiding file extensions.

- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

Examples

```
<ReportTemplates FileName="MyReport.rtp" Name="My Report" GUID="Longview_
rtp001"/>
```

Longview tools and editors

You can create a link to a Longview tool or editor using the following code:

```
<Item Name="Name" GUID="Id"/>
```

where:

- Item is the tool or editor and can be one of the following:

DataLocks	Events	EventsStatus
Mappings	Publisher	UserSubmissions

- Name is the name of the link to display, enclosed in double quotation marks.

- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

```
<Mappings Name="Mappings" GUID="LongviewCPM_Mappings"/>
```

URLs

You can create a URL using the following code:

```
<Url Name="Name" Address="Website Address" GUID="Id"/>
```

where:

- Name is the name of the link to display, enclosed in double quotation marks.
- Website Address is the address for the URL, enclosed in double quotation marks.
- Id is the Longview GUID for the link, following your system's naming conventions and enclosed in double quotation marks. For more information, see Understanding Longview GUIDs.

For more information, see Adding links.

Examples

```
<Url Name="Longview eLearning Library"
Address="https://longview.administratelms.com/" GUID="LongviewCPM_
eLearningLibrary"/>
```

Adding links

You can add a link to an existing folder or category. Links appear in the navigation pane in the order they are listed in the module file.

To add a link to an existing folder:

1. Open the relevant module file in your preferred text editor.
2. Locate the folder or category in which you want to add the new link.
3. In the appropriate place, add the relevant link code. For more information, see Understanding link elements.
4. Save your changes.
5. Launch Longview Client and verify that the new link appears as expected.

Modifying links

You can modify an existing link.

To modify an existing link:

1. Open the relevant module file in your preferred text editor.
2. Locate the link you want to modify.
3. Make the necessary changes. For more information, see Understanding link elements and Adding links.
4. Save your changes.
5. Launch Longview Client and verify that the link appears as expected.

Deleting links

You can delete an existing link.

Caution: Deleting a link cannot be undone. To restore a deleted link, you must recreate the link.

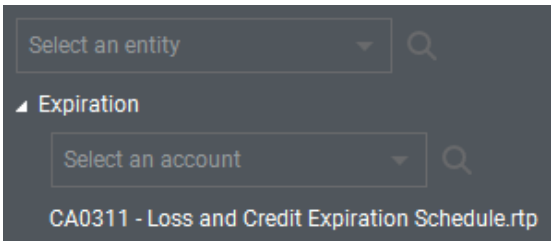
To delete an existing link:

1. Open the relevant module file in your preferred text editor.
2. Locate the link you want to delete.
3. Delete all content from the beginning of the relevant link element to the end of the relevant link element.
4. Save your changes.
5. Launch Longview Client and verify that the link does not appear.

Working with symbol selectors

In Longview Client, it is sometimes helpful to include a symbol selector so users can select a symbol to populate down through a series of folders or links. You can include a symbol selector above a series of folders or under a specific folder.

The following graphic shows a symbol selector for the ENTITIES dimension above a folder, and a symbol selector for the ACCOUNTS dimension under a folder.



Note: You cannot include a nested symbol selector for the same dimension as a top-level symbol selector.

This section contains information on the following main topics:

- Understanding symbol selector elements
- Adding symbol selectors
- Modifying symbol selectors
- Deleting symbol selectors

Understanding symbol selector elements

The <Symbol Selector> element is a single XML element with many optional XML attributes. Symbol selector elements must have a <Category> or <Folder> element as their parent.

Syntax:

```
<SymbolSelector Name="Name" Dimension="DimensionName" Symbolspec="Spec"
SelectionType="SelectionType" MultipleSelection="Multiple"
ReadAccessSelection="ReadAccess" DefaultText="WatermarkText"
DefaultSelection="SymbolName" DefaultAttribute="AttributeName"
DefaultAttributeType="AttributeType" ContentType="ContentType" />

  <AttributeFilters>
</SymbolSelector>
```

where:

- Name is currently not used.
- DimensionName is the dimension containing the symbols to include in the symbol selector, enclosed in double quotation marks.
- Spec is the level of symbols to display in the symbol selector hierarchy in relation to the specified symbol, for example, SymbolName### or SymbolName#99, enclosed in double quotation marks.
- SelectionType is the type of symbols users can select and can be either Parent or Leaf, enclosed in double quotation marks.
- Multiple is whether users can select multiple symbols or not and can be TRUE or FALSE, enclosed in double quotation marks.
- ReadAccess is whether users can select read-only symbols and can be TRUE or FALSE, enclosed in double quotation marks.
- WatermarkText is the text to display when no symbols are selected, enclosed in double quotation marks.
- SymbolName is the initial symbol to display in the symbol selector, enclosed in double quotation marks.
- AttributeName is the name of an attribute for which to display the default value, enclosed in double quotation marks.
- AttributeType is the type of attribute, if AttributeName is specified, and can be one of SYMBOL, SYSTEM, or USER, enclosed in double quotation marks.
- ContentType is whether to display symbol names or symbol names and descriptions, and can be either NAME or NAMEANDDESCRIPTION, enclosed in double quotation marks.
- AttributeFilters is an attribute filter in the format

```
<AttributeFilters Operator="Operator">
                        <AttributeFilter Spec="FilterType{AttributeName{Operation{Value"/>
```

```
</AttributeFilters>
```

where:

- Operator is the type of filter and can be AND or, enclosed in double quotation marks.
- FilterType is the method to use to search the hierarchy for symbols matching the filter criteria and can be one of ALL, PARENT, LEAF, or ROOT.
- AttributeName is the name of an attribute.
- Operation can be EQ for exactly equal to or NE for not equal to.
- Expression is a character string. If the expression contains spaces, enclose the expression in the XML character entity for double quotation marks (") such as "expression with spaces". If the expression is a list, separate multiple items with a pipe (|).

  For non-list attributes, the filter behaves as follows:
  - Attribute EQ Expression — Matches only if the attribute is an exact match of the expression.
  - Attribute NE Expression — Matches if the attribute is not an exact match of the expression.

  For list attributes, the filter behaves as follows:
  - Attribute EQ Expression — Matches if the attribute is an exact match of the expression, or is a list of values, any one of which exactly matches the expression.
  - Attribute NE Expression — Matches if the attribute is empty or a list of values, none of which exactly matches the expression.

Example:

```
<SymbolSelector Name="SymbolSelector" Dimension="Entities" Symbolspec=""
SelectionType="All" MultipleSelection="False" ReadAccessSelection="False"
DefaultText="Select an entity" DefaultAttribute="UD2InputDefault"
DefaultAttributeType="User" ContentType="NameAndDescription">

<AttributeFilters Operator="AND">

<AttributeFilter Spec="Leaf{ZGPNativeCurrency{EQ{CAD"/>

</AttributeFilters>

</SymbolSelector>
```

Adding symbol selectors

You can add a symbol selector to an existing category or folder.

To add a symbol selector:

1. Open the relevant module file in your preferred text editor.
2. Locate the element below which you want to add the new symbol selector.
3. In the appropriate place, add the <Symbol Selector> element. For more information, see Understanding symbol selector elements.

4. Save your changes.
5. Launch Longview Client and verify that the new symbol selector appears as expected.

Modifying symbol selectors

You can modify an existing symbol selector.

To add a symbol selector:

1. Open the relevant module file in your preferred text editor.
2. Locate the symbol selector you want to modify.
3. Make the necessary changes. For more information, see Understanding symbol selector elements.
4. Save your changes.
5. Launch Longview Client and verify that the symbol selector appears as expected.

Deleting symbol selectors

You can delete an existing symbol selector.

Caution: Deleting a symbol selector cannot be undone. To restore a deleted symbol selector, you must recreate the symbol selector.

To delete a symbol selector:

1. Open the relevant module file in your preferred text editor.
2. Locate the symbol selector you want to delete.
3. Delete all content from the beginning of the symbol selector element to the end of the symbol selector element.
4. Save your changes.
5. Launch Longview Client and verify that the symbol selector does not appear.

Working with Longview attributes

In some cases, you may want to include a Longview attribute token to display relevant values to users in Longview Client. You can include Longview attribute tokens within any XML element in your module files. You may want to, for example, display a user's current group in a folder name, such as the following:

```
<Folder Name="[[USER,UGPCurrentGroup,THIS]] Home">
```

For more information on Longview attribute token syntax, see Attribute tokens.