

Deploying LANSA Applications on Linux

- [About this Guide](#)
- [Deploy LANSA Applications to a Linux Server](#)
- [Execute Applications with a Linux Server](#)
- [Troubleshooting](#)

Edition Date March 19, 2014

© LANSA

About this Guide

- This guide provides instructions for planning and deploying LANSAs applications on a Linux Server. It does not include instructions or guidance in designing or creating applications with LANSAs.
- The contents are written for technical support staff and LANSAs developers.
- We recommend the use of the Korn shell (ksh) or Bourne shell (sh). All examples of Linux commands in this guide use the Korn shell.

It is assumed:

- Readers have a solid understanding of both the Linux operating system and LANSAs.
- The application to be ported to Linux already works with a Windows Server.
- An experienced System Administrator (root user) of the Linux system is available to carry out system administration tasks and advise the reader on Linux issues.
- An ORACLE Database Administrator (DBA) is available to create and configure databases, create user ids and advise the reader on ORACLE issues.

Also see

[Additional Information](#)

Additional Information

For more details about LANSAs on Linux, refer to these guides:

- *Installing LANSAs on Linux*
- *LANSAs Communications Setup*
- *LANSAs Technical Reference*

For the latest product information, refer to the LANSAs product web site at www.lansa.com/support

1. Deploy LANSAs Applications to a Linux Server

- Review [What is LANSAs on Linux?](#) in the *Installing LANSAs on Linux Guide*.
- Review the [1.1 Directory Structure for LANSAs under Linux](#).
- As a starting point go through the steps in the [1.2 Before You Begin Checklist](#).
- It is strongly recommended that you get the DEM partition working in your Linux environment before you go ahead with deploying your own application to Linux. [1.3 Test with the Verification and Sample Applications](#) describes what this involves. RDML code has been provided that can be used with your own application or the DEM partition to ensure that you have everything set up properly. Refer to [1.5 Verification Application Code \(L4WEX functions\)](#).
- Finally, [1.4 Deliver the Server Portion of an Application to Linux](#) describes how to export your application using the Deliver To feature in the LANSAs Editor.

Further Information

[1.1 Directory Structure for LANSAs under Linux](#)

[1.2 Before You Begin Checklist](#)

[1.3 Test with the Verification and Sample Applications](#)

[1.4 Deliver the Server Portion of an Application to Linux](#)

[1.5 Verification Application Code \(L4WEX functions\)](#)

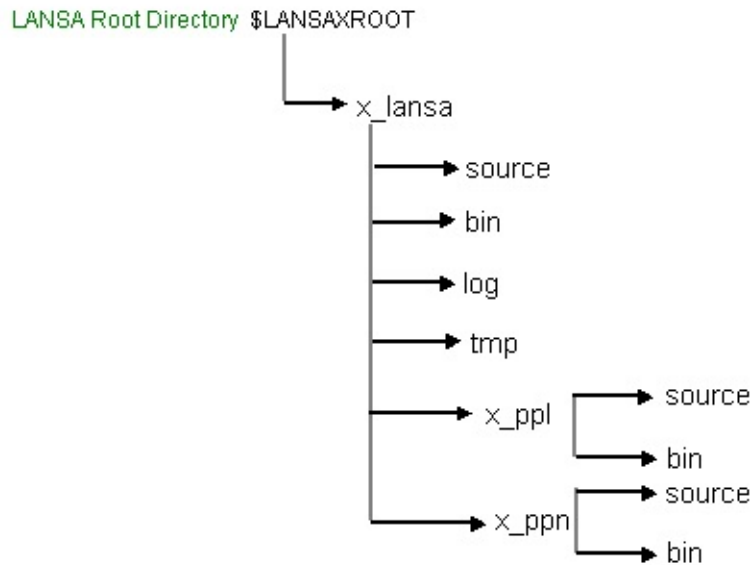
[↑ 1. Deploy LANSAs Applications to a Linux Server](#)

1.1 Directory Structure for LANSAX under Linux

The installation of LANSAX will create a set of directories under \$LANSAXROOT.

For each partition imported into LANSAX, a new directory called x_ppp (where ppp is the 3 character partition identifier) will also be created.

The main directory structure is shown below:



The following directories are used to store information common to all partitions:

\$LANSAXROOT/x_lansa

\$LANSAXROOT/x_lansa/source

\$LANSAXROOT/x_lansa/bin

For example, libx_bif.so (the BIF shared library) resides in x_lansa/bin because it is composed of common routines shared by all LANSAX generated applications.

Some of the types of objects stored in the source and bin sub-directories are:

Directory	File Type	Description
source	.c	C Code
	.h	C Code header

	.stg	C Code header - storage definitions
	.unx	Compiler/linker make file
	.xqi	DBID=*NONE useable read only index file
	.xqd	DBID=*NONE useable read only data file
	.xqf	DBID=*NONE useable read only flat file
	.ctd	Common table definitions
	.dat	Saved data for reload after table creation
	.txt	UTF-8 user-defined text strings
	.utx	C header for user-defined text strings
	.log	Log from Deliver To processing
bin	.so	Executable shared library (equivalent to a Windows .DLL)
	None	Executable object (equivalent to a Windows .EXE)
	.sh	Shell script (equivalent to a Windows .CMD or .BAT file)
log	x_err.log	Fatal error log
	lroute.*	Comms log and trace files.
tmp	x_trace*	Trace files for LANSAs runtime, when ITRO=Y is specified.
	.tmp	Temporary files

↑ 1. Deploy LANSAs Applications to a Linux Server

1.2 Before You Begin Checklist

ü Step

1. Create a **Minimum Supported Configuration document (MSC)** defining the minimum configuration your solution will viably support. This includes what servers, client platforms and web browsers your application will need. Consider:

- Hardware requirements
- Software requirements
- Supported screen resolutions
- Networking capabilities
- Maximum Data volumes.

2. Install and configure the latest version of LANSA on the Linux Server.

3. Request and install the appropriate licenses on the Linux Server.

4. Follow the instructions in the *LANSA Communications Setup Guide* to configure communications for the client and server machines

5. Verify the clients can communicate with the Server via TCP/IP.

Comments / Further Actions

A formal **Minimum Supported Configuration (MSC) Document** will:

- Inform decisions about the overall solution cost
- Establish the environment required to test the deployment of the solution or any patch/hotfix made to it.
- Raise management's awareness of the risk of implementing a "sub-MSC" solution.

Note - Any other application running on this "end-user" environment must also be considered when sizing your machine.

SuperServer clients must not have a newer version of LANSA installed than the version of LANSA on the Linux Server.

Refer to [the LANSA website](#) for information.

The Client and Server can only communicate via TCP/IP.

On the client, use the PING command specifying the name or IP address of the server system.

Wait until the PING command gives a good return code indicating it could successfully communicate with the server system.

6. Before you deploy your own application, test your configuration of the clients and LANSAs on Linux with the DEM partition and sample code.

Refer to [1.3 Test with the Verification and Sample Applications](#) for details.

↑ [1. Deploy LANSAs Applications to a Linux Server](#)

1.3 Test with the Verification and Sample Applications

We strongly recommend that you import the sample DEM partition and get it working with the L4WEX functions (SuperServer) before attempting to use your own application. This will achieve 2 objectives:

1. It will allow any application connection or translation problems that you encounter to be assessed within an environment that both you and LANSAs support staff have available.
2. It will provide you with a basic example of deploying the server portion of an application to Linux, again in an environment that both you and LANSAs support staff have available.

Create the sample L4WEX functions, provided in [1.5 Verification Application Code \(L4WEX functions\)](#) on the process menu L4WEXAM1.

Follow the instructions in [1.4 Deliver the Server Portion of an Application to Linux](#) to deploy DEM to Linux.

Once you have everything ready, you should attempt to execute the verification process L4WEXAM1.

Follow the instructions in [Executing Applications with a Linux Server](#) to test with the verification process L4WEXAM1. The function L4WEX01 should be used as the starting point.

Once you have successfully tested the DEM partition, go ahead and deploy and test your own application.

If you plan to support non-English languages, you should also do some basic testing in an additional language, to ensure there are no codepage / locale issues. As the demo only ships with English, French, and Japanese text, you may need to enter some language-specific text, or use your own simple application code to test.

[↑ 1. Deploy LANSAs Applications to a Linux Server](#)

1.4 Deliver the Server Portion of an Application to Linux

These instructions assume that the tasks listed in [1.2 Before You Begin Checklist](#) have been carried out.

Determine which system and application objects need to exist on the server. This means all files (and optionally, their data), and any reusable parts, Web objects, functions and their processes that will execute on the Server. These include trigger functions, system variable evaluation functions, RPCs (functions that will be called by CALL_SERVER_FUNCTION or LceLANSACall or LceSubmit), batch jobs that will be called directly from the command-line, and so on. Client-only objects, such as components and functions containing REQUEST, DISPLAY, or POP-UP commands do not need to be deployed to the Server.

Follow the instructions in [Other Remote System Monitors](#) in the *Visual LANSA Administrator's Guide* to define the Linux deployment system, initialize the partition, and deliver your objects to the server.

1.5 Verification Application Code (L4WEX functions)

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following code examples.

Client functions contain POP_UP and REQUEST commands so must be created as RDML, not RDMLX.

- [1.5.1 L4WEX01 Example of On Top Connect/Disconnect](#)
- [1.5.2 L4WEX02 Exchange Example: Client Portion](#)
- [1.5.3 L4WEX52 Exchange Example: Server Portion](#)
- [1.5.4 L4WEX03 List Example: Client Portion](#)
- [1.5.5 L4WEX53 List Example: Server Portion](#)

↑ [1. Deploy LANSAs Applications to a Linux Server](#)

1.5.1 L4WEX01 Example of On Top Connect/Disconnect

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following example.

```
FUNCTION OPTIONS(*DIRECT);
*****.
DEFINE FIELD(#L4W_AS400) TYPE(*CHAR) LENGTH(1) LABEL('AS/400');
DEFINE FIELD(#L4W_OTHER) TYPE(*CHAR) LENGTH(1) LABEL('Other');
DEFINE FIELD(#L4W_ANAM) TYPE(*CHAR) LENGTH(20) LABEL('Server Name');
DEFINE FIELD(#L4W_ONAM) TYPE(*CHAR) LENGTH(20) LABEL('Server Name');
DEFINE FIELD(#L4W_LOCK) TYPE(*CHAR) LENGTH(1) LABEL('Diver');
DEFINE FIELD(#L4W_SHOWM) TYPE(*CHAR) LENGTH(1) LABEL('Status');
DEFINE FIELD(#L4W_COMC) TYPE(*CHAR) LENGTH(1) LABEL('Command');
DEFINE FIELD(#L4W_DBCS) TYPE(*CHAR) LENGTH(1) LABEL('DBCS');
DEFINE FIELD(#L4W_CTST) TYPE(*CHAR) LENGTH(10) DESC('C-
>S Table') DEFAULT(ANSEBC1140);
DEFINE FIELD(#L4W_STCT) TYPE(*CHAR) LENGTH(10) DESC('S-
>C Table') DEFAULT(EBC1140ANS);
DEFINE FIELD(#L4W_EXEP) TYPE(*CHAR) LENGTH(2) LABEL('Exec F');
DEFINE FIELD(#L4W_RETC) TYPE(*CHAR) LENGTH(2) LABEL('Return');
DEFINE FIELD(#L4W_EARG) TYPE(*CHAR) LENGTH(255) LABEL('X_');
DEFINE FIELD(#L4W_EAR1) TYPE(*CHAR) LENGTH(60) LABEL('Over');
DEFINE FIELD(#L4W_EAR2) TYPE(*CHAR) LENGTH(60) LABEL('Over');
DEFINE FIELD(#L4W_EAR3) TYPE(*CHAR) LENGTH(60) LABEL('Over');
DEFINE FIELD(#L4W_EAR4) TYPE(*CHAR) LENGTH(60) LABEL('Over');
DEFINE FIELD(#L4W_USER) TYPE(*CHAR) LENGTH(10) LABEL('Server');
DEFINE FIELD(#L4W_PSWD) TYPE(*CHAR) LENGTH(10) LABEL('Server');
DEFINE FIELD(#L4W_PROC) TYPE(*CHAR) LENGTH(10) LABEL('Call');
DEFINE FIELD(#L4W_FUNC) TYPE(*CHAR) LENGTH(7) LABEL('Call F');
DEFINE FIELD(#L4W_BLKs) TYPE(*DEC) LENGTH(7) DECIMALS(0) L;
DEFINE FIELD(#L4W_TRC2) TYPE(*CHAR) LENGTH(1) LABEL('Trace 1');
DEFINE FIELD(#L4W_TRC4) TYPE(*CHAR) LENGTH(1) LABEL('Trace 1');
DEFINE FIELD(#L4W_TST1) TYPE(*CHAR) LENGTH(1) LABEL('Perform');
DEFINE FIELD(#L4W_TST2) TYPE(*CHAR) LENGTH(1) LABEL('Perform');
DEFINE FIELD(#L4W_APND) TYPE(*CHAR) LENGTH(1) DEFAULT(A);
*****.
DEF_LIST NAME(#SAVE1) FIELDS(#L4W_AS400 #L4W_OTHER #L4W_
DEF_LIST NAME(#SAVE2OUT) FIELDS(#L4W_APND #L4W_EARG) TY
```

```

DEF_LIST NAME(#SAVE2IN) FIELDS(#L4W_EARG) TYPE(*WORKING)
*****;
DEF_COND NAME(*L4W_OTHER) COND('#L4W_OTHER = "1"');
DEF_COND NAME(*L4W_AS400) COND('#L4W_AS400 = "1"');
DEF_COND NAME(*OKAY) COND('#L4W_RETC = OK');
DEF_COND NAME(*NOTOKAY) COND('#L4W_RETC *NE OK');
DEF_COND NAME(*TRACEL2) COND('#L4W_TRC2 = "1" ');
DEF_COND NAME(*TRACEL4) COND('#L4W_TRC4 = "1"');
DEF_COND NAME(*TEST1) COND('#L4W_TST1 = "1" ');
DEF_COND NAME(*TEST2) COND('#L4W_TST2 = "1"');
DEF_COND NAME(*NOTEST) COND(('#L4W_TST1 *NE "1"') *AND (#L4
*****;
EXECUTE SUBROUTINE(Load_DFT);
POP_UP FIELDS((#L4W_AS400 *IN) (#L4W_OTHER *IN)) IDENTIFY(*L
EXECUTE SUBROUTINE(SAVE_DFT);
*****;
BEGIN_LOOP;
IF COND(*L4W_OTHER);
REQUEST FIELDS(#L4W_ONAM #L4W_LOCK #L4W_SHOWM #L4W_T
AS400 Server Details ');
EXECUTE SUBROUTINE(BLD_ARGS);
USE BUILTIN(DEFINE_ANY_SERVER) WITH_ARGS(SERVER #L4W_OI
#L4W_ARG #L4W_LOCK #L4W_SHOWM) TO_GET(#L4W_RETC);
ELSE;
REQUEST FIELDS(#L4W_ANAM #L4W_LOCK #L4W_SHOWM #L4W_C
USE BUILTIN(DEFINE_OS_400_SERVER) WITH_ARGS(SERVER #L4W_
ENDIF;
EXECUTE SUBROUTINE(SAVE_DFT);
IF COND(*OKAY);
USE BUILTIN(CONNECT_SERVER) WITH_ARGS(SERVER #L4W_PSWI
IF COND(*OKAY);
USE BUILTIN(CONNECT_FILE) WITH_ARGS('*' SERVER #L4W_BLKs)
IF COND(*TEST1);
CALL PROCESS(*DIRECT) FUNCTION(L4WEX02) EXIT_USED(*NEXT
ENDIF;
IF COND(*TEST2);
CALL PROCESS(*DIRECT) FUNCTION(L4WEX03) EXIT_USED(*NEXT
ENDIF;
IF COND(*NOTEST);

```

```

CALL PROCESS(#L4W_PROC) FUNCTION(#L4W_FUNC) EXIT_USED(*
ENDIF;
USE BUILTIN(DISCONNECT_FILE) WITH_ARGS('*' SERVER);
USE BUILTIN(DISCONNECT_SERVER) WITH_ARGS(SERVER) TO_GET
IF COND(*OKAY);
MESSAGE MSGTXT('Disconnection from server completed normally');
ELSE;
MESSAGE MSGTXT('Error detected when disconnecting from server');
ENDIF;
MENU;
ENDIF;
ENDIF;
END_LOOP;
*****;
SUBROUTINE NAME(BLD_ARGS);
DEFINE FIELD(#L4W_ARG) TYPE(*CHAR) LENGTH(256);
CHANGE FIELD(#L4W_ARG) TO(#L4W_EARG);
IF COND(*TRACEL4);
USE BUILTIN(BCONCAT) WITH_ARGS(#L4W_ARG 'ITRO=Y') TO_GET
USE BUILTIN(BCONCAT) WITH_ARGS(#L4W_ARG 'ITRL=4') TO_GET(
ELSE;
IF COND(*TRACEL2);
USE BUILTIN(BCONCAT) WITH_ARGS(#L4W_ARG 'ITRO=Y') TO_GET
USE BUILTIN(BCONCAT) WITH_ARGS(#L4W_ARG 'ITRL=2') TO_GET(
ENDIF;
ENDIF;
ENDROUTINE;
*****;
SUBROUTINE NAME(LOAD_DFT);
CLR_LIST NAMED(#SAVE1);
CLR_LIST NAMED(#SAVE2IN);
USE BUILTIN(TRANSFORM_FILE) WITH_ARGS(#SAVE1 *FUNCTION '
GET_ENTRY NUMBER(1) FROM_LIST(#SAVE1);
GET_ENTRY NUMBER(1) FROM_LIST(#SAVE2IN);
ENDROUTINE;
***** COMMENT(Routine);
SUBROUTINE NAME(SAVE_DFT);
CLR_LIST NAMED(#SAVE1);
CLR_LIST NAMED(#SAVE2OUT);

```

```
ADD_ENTRY TO_LIST(#SAVE1);  
ADD_ENTRY TO_LIST(#SAVE2OUT);  
USE BUILTIN(TRANSFORM_LIST) WITH_ARGS(#SAVE1 *FUNCTION 5  
ENDROUTINE;
```

[↑ 1.5 Verification Application Code \(L4WEX functions\)](#)

1.5.2 L4WEX02 Exchange Example: Client Portion

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following code example.

```
FUNCTION OPTIONS(*DIRECT);
*****;
DEFINE FIELD(#L4W_TEST) TYPE(*DEC) LENGTH(7) DECIMALS(0) L
DEFINE FIELD(#L4W_COUNT) TYPE(*DEC) LENGTH(7) DECIMALS(0)
DEFINE FIELD(#L4W_FC1) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_FC2) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_RSL1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RSL2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_CMP1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_CMP2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RETC) TYPE(*CHAR) LENGTH(2);
*****;
BEGIN_LOOP;
POP_UP FIELDS((#L4W_TEST *IN)) DESIGN(*DOWN) PANEL_TITL('Pe
BEGINCHECK;
RANGECHECK FIELD(#L4W_TEST) RANGE((1 100000)) MSGTXT('Num
ENDCHECK;
*****;
CHANGE FIELD(#L4W_FC1) TO(1);
CHANGE FIELD(#L4W_FC2) TO(#L4W_TEST);
BEGIN_LOOP USING(#L4W_COUNT) TO(#L4W_TEST);
CHANGE FIELD(#L4W_RSL1 #L4W_RSL2) TO(*NULL);
EXCHANGE FIELDS(#L4W_FC1 #L4W_FC2);
USE BUILTIN(CALL_SERVER_FUNCTION) WITH_ARGS(SERVER L4W
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_RETC *NE OK');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1');
LEAVE IF('#L4W_CMP2 *NE #L4W_RSL2');
CHANGE FIELD(#L4W_FC1) TO('#L4W_FC1 + 1');
CHANGE FIELD(#L4W_FC2) TO('#L4W_FC2 - 1');
END_LOOP;
IF COND(#L4W_COUNT *LT #L4W_TEST);
MESSAGE MSGTXT('Test ***FAILED**');
```



```
ELSE;  
MESSAGE MSGTXT('Test completed normally');  
ENDIF;  
END_LOOP;  
*****;  
,
```

[↑ 1. Deploy LANSAs Applications to a Linux Server](#)

1.5.3 L4WEX52 Exchange Example: Server Portion

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following code example.

```
FUNCTION OPTIONS(*HEAVYUSAGE *DIRECT);
*****;
DEFINE FIELD(#L4W_FC1) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_FC2) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_RSL1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RSL2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
*****;
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
*****;
EXCHANGE FIELDS(#L4W_RSL1 #L4W_RSL2);
RETURN;
*****;
```

[↑ 1. Deploy LANSA Applications to a Linux Server](#)

1.5.4 L4WEX03 List Example: Client Portion

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following code example.

```
FUNCTION OPTIONS(*DIRECT);
*****;
DEFINE FIELD(#L4W_TEST) TYPE(*DEC) LENGTH(7) DECIMALS(0) L
DEFINE FIELD(#L4W_LIST) TYPE(*DEC) LENGTH(7) DECIMALS(0) L/
DEFINE FIELD(#L4W_COUNT) TYPE(*DEC) LENGTH(7) DECIMALS(0)
DEFINE FIELD(#L4W_LISTC) TYPE(*DEC) LENGTH(7) DECIMALS(0);
DEFINE FIELD(#L4W_FC1) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_FC2) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_RSL1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RSL2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_CMP1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_CMP2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RETC) TYPE(*CHAR) LENGTH(2);
DEF_LIST NAME(#L4W_LIST1) FIELDS(#L4W_FC1 #L4W_FC2 #L4W_F
DEF_LIST NAME(#L4W_LIST2) FIELDS(#L4W_FC2 #L4W_FC1 #L4W_F
DEF_LIST NAME(#L4W_LIST3) FIELDS(#L4W_RSL1 #L4W_FC2 #L4W_
DEF_LIST NAME(#L4W_LIST4) FIELDS(#L4W_FC1 #L4W_RSL2 #L4W_
DEF_LIST NAME(#L4W_LIST5) FIELDS(#L4W_RSL1 #L4W_RSL2 #L4W
*****;
BEGIN_LOOP;
POP_UP FIELDS((#L4W_TEST *IN) (#L4W_LIST *IN)) DESIGN(*DOWN
BEGINCHECK;
RANGECHECK FIELD(#L4W_TEST) RANGE((1 100000)) MSGTXT('Num
RANGECHECK FIELD(#L4W_LIST) RANGE((1 100)) MSGTXT('Entrys in
ENDCHECK;
*****;
*****;
BEGIN_LOOP USING(#L4W_COUNT) TO(#L4W_TEST);
CLR_LIST NAMED(#L4W_LIST1);
CLR_LIST NAMED(#L4W_LIST2);
CLR_LIST NAMED(#L4W_LIST3);
CLR_LIST NAMED(#L4W_LIST4);
CLR_LIST NAMED(#L4W_LIST5);
*****;
```

```

CHANGE FIELD(#L4W_RSL1 #L4W_RSL2) TO(0);
CHANGE FIELD(#L4W_FC1) TO(1);
CHANGE FIELD(#L4W_FC2) TO(#L4W_LIST);
BEGIN_LOOP TO(#L4W_LIST);
ADD_ENTRY TO_LIST(#L4W_LIST1);
ADD_ENTRY TO_LIST(#L4W_LIST2);
ADD_ENTRY TO_LIST(#L4W_LIST3);
ADD_ENTRY TO_LIST(#L4W_LIST4);
ADD_ENTRY TO_LIST(#L4W_LIST5);
CHANGE FIELD(#L4W_FC1) TO('#L4W_FC1 + 1');
CHANGE FIELD(#L4W_FC2) TO('#L4W_FC2 - 1');
END_LOOP;
USE BUILTIN(CALL_SERVER_FUNCTION) WITH_ARGS(SERVER L4W
LEAVE IF('#L4W_RETC *NE OK');
*****;
CHANGE FIELD(#L4W_LISTC) TO(0);
SELECTLIST NAMED(#L4W_LIST1);
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1) *OR (#L4W_CMP2 *NE #L4W
CHANGE FIELD(#L4W_LISTC) TO('#L4W_LISTC + 1');
ENDSELECT;
LEAVE IF('#L4W_LISTC *NE #L4W_LIST');
*****;
CHANGE FIELD(#L4W_LISTC) TO(0);
SELECTLIST NAMED(#L4W_LIST2);
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1) *OR (#L4W_CMP2 *NE #L4W
CHANGE FIELD(#L4W_LISTC) TO('#L4W_LISTC + 1');
ENDSELECT;
LEAVE IF('#L4W_LISTC *NE #L4W_LIST');
*****;
CHANGE FIELD(#L4W_LISTC) TO(0);
SELECTLIST NAMED(#L4W_LIST3);
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1) *OR (#L4W_CMP2 *NE #L4W
CHANGE FIELD(#L4W_LISTC) TO('#L4W_LISTC + 1');

```

```

ENDSELECT;
LEAVE IF('#L4W_LISTC *NE #L4W_LIST');
*****;
CHANGE FIELD(#L4W_LISTC) TO(0);
SELECTLIST NAMED(#L4W_LIST4);
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1) *OR (#L4W_CMP2 *NE #L4W
CHANGE FIELD(#L4W_LISTC) TO('#L4W_LISTC + 1');
ENDSELECT;
LEAVE IF('#L4W_LISTC *NE #L4W_LIST');
*****;
CHANGE FIELD(#L4W_LISTC) TO(0);
SELECTLIST NAMED(#L4W_LIST5);
CHANGE FIELD(#L4W_CMP1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_CMP2) TO('#L4W_FC1 / #L4W_FC2');
LEAVE IF('#L4W_CMP1 *NE #L4W_RSL1) *OR (#L4W_CMP2 *NE #L4W
CHANGE FIELD(#L4W_LISTC) TO('#L4W_LISTC + 1');
ENDSELECT;
LEAVE IF('#L4W_LISTC *NE #L4W_LIST');
*****;
END_LOOP;
IF COND('#L4W_COUNT *LT #L4W_TEST');
MESSAGE MSGTXT('Test ***FAILED**');
ELSE;
MESSAGE MSGTXT('Test completed normally');
ENDIF;
END_LOOP;
*****;

```

↑ 1. Deploy LANSAs Applications to a Linux Server

1.5.5 L4WEX53 List Example: Server Portion

Refer to [1.3 Test with the Verification and Sample Applications](#) for details on using the following code example.

```
FUNCTION OPTIONS(*HEAVYUSAGE *DIRECT) RCV_LIST(#L4W_LIS
*****);
DEFINE FIELD(#L4W_FC1) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_FC2) TYPE(*DEC) LENGTH(15) DECIMALS(0);
DEFINE FIELD(#L4W_RSL1) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEFINE FIELD(#L4W_RSL2) TYPE(*DEC) LENGTH(15) DECIMALS(5);
DEF_LIST NAME(#L4W_LIST1) FIELDS(#L4W_FC1 #L4W_FC2 #L4W_F
DEF_LIST NAME(#L4W_LIST2) FIELDS(#L4W_FC2 #L4W_FC1 #L4W_F
DEF_LIST NAME(#L4W_LIST3) FIELDS(#L4W_RSL1 #L4W_FC2 #L4W_
DEF_LIST NAME(#L4W_LIST4) FIELDS(#L4W_FC1 #L4W_RSL2 #L4W_
DEF_LIST NAME(#L4W_LIST5) FIELDS(#L4W_RSL1 #L4W_RSL2 #L4W
*****);
SELECTLIST NAMED(#L4W_LIST1);
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
UPD_ENTRY IN_LIST(#L4W_LIST1);
ENDSELECT;
*****;
SELECTLIST NAMED(#L4W_LIST2);
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
UPD_ENTRY IN_LIST(#L4W_LIST2);
ENDSELECT;
*****;
SELECTLIST NAMED(#L4W_LIST3);
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
UPD_ENTRY IN_LIST(#L4W_LIST3);
ENDSELECT;
*****;
SELECTLIST NAMED(#L4W_LIST4);
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
UPD_ENTRY IN_LIST(#L4W_LIST4);
```

```
ENDSELECT;
*****;
SELECTLIST NAMED(#L4W_LIST5);
CHANGE FIELD(#L4W_RSL1) TO('#L4W_FC1 * #L4W_FC2');
CHANGE FIELD(#L4W_RSL2) TO('#L4W_FC1 / #L4W_FC2');
UPD_ENTRY IN_LIST(#L4W_LIST5);
ENDSELECT;
*****;
RETURN;
```

[↑ 1. Deploy LANSA Applications to a Linux Server](#)

2. Execute Applications with a Linux Server

Once your client and server are communicating, you have verified that lcolist is running on your server, and the required objects have been successfully deployed to the Server, you are ready to test execution.

- When you are first experimenting with using a Linux Server, we recommend using the sample L4WEX01 function to connect to the server to begin with. (Please refer to [Testing with the Verification and Sample Applications](#) for further information.) Later you can write your own connection function, and perhaps automatically call it via the INIT= argument. (See the [INIT= and TERM= Parameters](#) in the *Technical Reference Guide* for more information.)
- When you start executing applications, you will probably use the LANSA owner as your login for testing. However, you will eventually need to allow test or production users access to your application. Users must be able to read, execute, create, or update certain files and directories for successful application execution. Please carefully read [Allow Users Access to LANSA](#) in the *Installing LANSA on Linux* guide.
- You may need to setup some X_RUN arguments as standard. You may use the \$X_RUN environment variable, or an x_lansa.pro profile to set these up. Please refer to [Set up default X_RUN parameters](#) in the *Installing LANSA on Linux Guide* for more details.
- Client sessions may need to set specific X_RUN arguments for connection to a Linux server. Refer to [2.1 Override X_RUN arguments inherited from the Client](#) for details.
- You will need to execute X_RUN from the command line to check your license status, and possibly to execute batch jobs. Refer to [2.2 Start X_RUN from the command line](#) for details.
- Full details of all the X_RUN arguments can be found in [The X_RUN Parameters](#) in the *Technical Reference Guide*.

You can refer to [Troubleshooting](#) for further assistance.

Further Information

[2.1 Override X_RUN arguments inherited from the Client](#)

[2.2 Start X_RUN from the command line](#)

[↑ 2. Execute Applications with a Linux Server](#)

2.1 Override X_RUN arguments inherited from the Client

Several standard X_RUN arguments are automatically inherited by the Server. (See [DEFINE_OTHER_SERVER](#) and [The PSXX= Parameter](#) in the *Technical Reference Guide* for details.) In most cases, these arguments are not appropriate for a Linux connection. Some recommendations for overrides are:

- Override the printer name with PRTR= (and optionally PPTH=) if any RPCs will print.
- Override DBID= and DBII=. For example, if DBID=*NONE is passed over by default, the Linux Server will not be able to access the database.
- Override DBUS= and PSWD= if they do not match your local database.
- Use the special override value *SERVER to use server defaults rather than client settings. Refer to [2.1.1 Override value *SERVER](#).
- If you wish to use separate temporary files (or printer files when PRTR=*PATH) into different directories for different users (for example), override TPTH= (or PPTH=) to a specific full path that is generated at connection time. You could use the system variables *USER and *PATHDELIM to generate TPTH=/home/user1/. (Keep in mind that the Linux file system is case-sensitive.)

For further details on the X_RUN parameters, please refer to [The X_RUN Parameter Summary](#) in the *Technical Reference Guide*.

2.1.1 Override value *SERVER

You may specify the special value *SERVER when you want to override the PC default with the Linux Server's default. This will allow you to utilize the standard Linux defaults or specific defaults that you have set up. Please refer to [Set up default X_RUN parameters](#) in the *Installing LANSA on Linux Guide* for more details.

For example, instead of overriding with

```
DBID=tst1 DBII=tst1 INIT=lnxinitf
```

you could replace this with

```
DBID=*SERVER DBII=*SERVER INIT=*SERVER
```

(DBID=tst1 and INIT=lnxinitf would have to be set in x_lansa.pro or \$X_RUN.)

[↑ 2.1 Override X_RUN arguments inherited from the Client](#)

2.2 Start X_RUN from the command line

As LANSAs does not support an interactive user interface on Linux, only batch jobs can be started from the command line. The X_RUN argument MODE defaults to B (batch) and cannot be changed.

Any user that will be executing X_RUN from the command line will need to have their environment configured correctly. Refer to [Allow Users Access to LANSAs](#) in the *Installing LANSAs on Linux Guide* for details.

Please refer to [Batch Jobs](#) in the *Technical Reference Guide* for further details of the differences between batch jobs on Windows and Linux.

[↑ 2. Execute Applications with a Linux Server](#)

3. Troubleshooting

Please refer to the appropriate section:

- [3.1 Install or Upgrade](#)
- [3.2 Deliver To](#)
- [3.3 Character translation/conversion issues](#)
- [3.4 X_RUN or submitted jobs](#)
- [3.5 Connecting to a Linux Server](#)
- [3.5.1 Database](#)

Need more help?

If you cannot resolve a problem using the advice in this section, please complete the following:

1. As the LANSAXROOT owner, execute the script support.sh (located in \$LANSAXROOT/x_lansa/bin, which should be in the PATH) to create the file support.txt in the current directory.
2. Contact your LANSAX supplier for support and attach the support.txt file.

3.1 Install or Upgrade

Any error messages or warnings during execution of `vinstall.py` will be logged to `stdout`.

The most common install problems are usually database issues.

Oracle SQL errors usually appear as:

ORA-99999: message

Note: The following messages are expected (and can be ignored) for upgrades or reinstalls:

ORA-01921: role name XXXX conflicts with another user or role name

ORA-01920: user name XXXXX conflicts with another user or role name

RUNSQL (table creation errors) appear as:

RUNSQL ended in error. Return code is -1652.

If you have database issues, please refer to [3.5.1 Database](#).

It is safe to re-run the install after you have resolved the issues that caused problems.

[↑ 3. Troubleshooting](#)

3.2 Deliver To

Any error messages or warnings during server-side execution of Deliver To will be logged to a job log, which can be retrieved by clicking on the magnifying glass against the message.

- If you have issues connecting to the server, please refer to [3.5 Connecting to a Linux Server](#).
- If you have database issues, please refer to [3.5.1 Database](#).
- If you have other issues, please refer to [3.4 X_RUN or submitted jobs](#).

Note that default X_RUN parameters will also be used by Deliver To. Please refer to [Set up default X_RUN parameters](#) in the *Installing LANSAs on Linux Guide* to determine where default parameters may be set up.

[↑ 3. Troubleshooting](#)

3.3 Character translation/conversion issues

A locale includes location-specific information such as date and time format, currency symbol, range of characters supported, and so on.

LANSA uses the `setlocale()` and `nl_langinfo()` APIs to retrieve information about your site's locale. LANSA assumes that the locale environment variables are set correctly for your location. For example, when you install Red Hat Enterprise Linux, and choose an Australian timezone, none of the `LC_*` variables are set, and `LANG` is set to `en_AU.UTF-8`. Refer to your operating system manuals on the `setlocale()` and `nl_langinfo` APIs for further information.

[↑ 3. Troubleshooting](#)

3.4 X_RUN or submitted jobs

Where can I find logs of messages and errors?

- LANSAX Fatal errors are logged to a LANSAX file called x_err.log and also to the system log. The x_err.log file contains the exact X_RUN parameter list used (including defaults from x_lansa.pro and the \$X_RUN environment variable). The x_err.log is located in the \$LANSAXROOT/x_lansa/log by default.
- LANSAX messages are logged to standard error (in the case of X_RUN executed from the command line) and the system log.

Refer to *Batch Jobs* in the *Deploying Visual LANSAX Applications Guide* for details on capturing standard error output and accessing the system log.

- LANSAX Communications errors are logged to \$LANSAXROOT/log/lroute.trc by default. Refer to *Linux Configuration* in the *LANSAX Communications Setup Guide* for other possible locations.
- If no log files are being created, and you are not logged in to the Linux Server as the LANSAX owner, file and directory permissions may be causing you problems. Try again, using the LANSAX owner as the login, and refer to [Allow Users Access to LANSAX](#) in the *Installing LANSAX on Linux Guide*.

Where can I find out what SQL error -1017 means?

SQL error -1017 means invalid user id or password when connecting to an ORACLE database.

Refer to [3.5.1 Database](#) for help on resolving other SQL errors.

Where can I find out how my default X_RUN parameters are being set?

Refer to Setting up default X_RUN parameters in the *Installing LANSAX on Linux Guide*.

[↑ 3. Troubleshooting](#)

3.5 Connecting to a Linux Server

Refer to this checklist for help with problems connecting to the Linux Server.

Also see

[3.5.1 Database](#)

[3.5.2 Connection](#)

[3.5.3 Diagnosing ORACLE / SQL run-time errors](#)

Server Problem Check List

Check to be Performed	Comments / Further Actions
Process lcolist shows as two active processes when the active processes are displayed (normal operation).	You can use <code>ps -ef grep lco</code> to list all LANSA processes on the Server. Refer to the <i>LANSA Communications Setup Guide</i> . If you cannot see these processes, the LANSA listener is not running.
O/S user profile and password coming from the client are valid on the server system and in the correct case (if applicable).	
DBMS user profile and password coming from the client are valid on the server system.	
Process lctop shows up as an active process	If this is not true, you have an initial connection problem or an application start up problem. If there is no entry in the x_err.log file, try turning on tracing of

<p>after connection is made. Each connection has its own lctop running.</p> <p>You can use <code>ps -ef grep lctop</code> to check.</p>	<p>Communications on both the Client and the Server. See the <i>LANSAs Communications Setup Guide</i> for details. If you still cannot find any logs, refer to Where can I find logs of messages and errors?</p>
<p>Process lctop starts when a connection is made but it quickly fails (or disappears) and the client gets a communications error.</p>	<p>You have an initial X_RUN environment problem such as DBMS connection problem or a failure in your own application. Look in the <code>x_err.log</code> file on the server system. Refer to Where can I find logs of messages and errors? for the file location.</p>
<p>Process lctop fails (or disappears) while your application is running and the client gets a communications error.</p>	<p>Look in the <code>x_err.log</code> file on the server system. Refer to Where can I find logs of messages and errors? for the file location.</p>
<p><code>x_err.log</code> shows an SQL error</p>	<p>Refer to <i>Database</i> following for help.</p>

3.5.1 Database

For problems connecting to an ORACLE database, refer to [3.5.2 Connection](#).

For help diagnosing other ORACLE problems, refer to *Diagnosing ORACLE / SQL run-time errors* following.

Note: If you have ORACLE database issues that you cannot resolve, please contact ORACLE support for assistance before contacting your LANSAsupplier.

3.5.2 Connection

The most common cause of connection problems are:

- The database user id or password is wrong. This can usually be diagnosed by looking in the x_err.log where the X_RUN parameters are listed.
- The ORACLE listener or database is not started. To check whether the listener is started use the following command on the database server

```
lsnrctl status
```

To see whether the database is started use the following command on the database server

```
ps -ef | grep $ORACLE_SID
```

- The ORACLE listener is not configured correctly. Refer to Oracle support and manuals for possible causes.

3.5.3 Diagnosing ORACLE / SQL run-time errors

You can diagnose run-time SQL errors by looking up the ORACLE message for a given number. For example, if you get SQL error code 942, you can look up ORA-00942 to see that the table does not exist (or the user does not have any privileges to see the table).

A standard ORACLE server nstallation includes the utility oerr. To look up SQL error -942, use the following command:

```
oerr ORA 942
```

The ORACLE guides may provide more information than the oerr utility.

[↑ 3. Troubleshooting](#)