Altair SLC Installation and User Guide

for z/OS

Version: 2023.5

www.altair.com

Contents

Introduction

This document is intended to help guide you through installing Altair SLCon the z/OS platform, including sections on how to use Altair SLC, and what to do if you have SAS language programs with or without associated data.

z/OS or MVS datasets are generally referred to as *z/OS datasets*.

Dataset on its own refers to a WPD dataset (the proprietary binary dataset storage format used by Altair SLC) held within a *data library*.

Installation and configuration process

The installation and configuration of Altair SLC on z/OS consists of the following sequence of steps:

- Obtain the Altair SLC software
- Ensure that all pre-requisites are met
- Upload the distribution file to z/OS
- Unpack the distribution file
- Extract the z/OS components
- Extract the USS components (necessary if support for TrueType fonts is required)
- Extract the SDK components (optional)
- Modify the WPSPROC JCL member
- Apply the Altair SLC licence key
- Verify that the installation was successful
- Initialise the TrueType font cache (required for any output using TrueType fonts)
- Configure Altair SLC and the environment
- Migrate existing data and code libraries
- Execute SAS language programs.

Altair SLC software requirements

To install Altair SLC on the z/OS platform, you require:

- The distribution package (zip file)
- The licence key file specific to your site

The distribution package and licence key file are obtained directly from World Programming; please contact sales@worldprogramming.com

Distribution package

Altair SLC is distributed as a single `.zip` file. This archive contains a single z/OS installation file in IBM's TERSE format, and has a name that ends with `-zos-s390.dlib.ter`.

When processed, this package installs both the application and associated text files. Once installed, the end-user licence agreement may be found as a member in the `<WPSPFX>.LICENCE` library. The naming convention of the members is a 2-character language code, such as EN, FR or JA, followed by an EBCDIC (Extended Binary Coded Decimal Interchange Code) code page name, for example:

- `<WPSPFX>.LICENCE(EN1047)`
- `<WPSPFX>.LICENCE(FR297)`, or
- `<WPSPFX>.LICENCE(JA930)`

Licence key file

As part of the installation process, you will need to apply a licence key to the Altair SLC installation. Without a valid licence key, you will not be able to run SAS language programs.

On commencement of a trial, or when a license has been purchased, a licence key is provided in a file ending with `.wpskey` separately from the distribution package.

The licence key file is supplied in plain text and contains information specific to your site, together with an encrypted password that enables Altair SLC to be executed by you. It is applied using the `SETINIT` procedure described in the section on Applying the Licence Key.

Prerequisites

Altair SLC uses z/OS services to implement some functionality. Some of these services are a part of z/OS Unix System Services (USS) and, as such, require the user to have a valid OMVS segment defined in their user profile. This will avoid errors that may otherwise occur during usage.

The process of installing Altair SLC on the z/OS platform requires the creation of several z/OS datasets. It may also require the installer to specify z/OS dataset names and (possibly) USS directory and path names for some existing system software components.

Consideration should be given to the following locations prior to starting the installation process:

z/OS TSO USERID of the person installing Altair SLC	*<userid>*
z/OS HLQ for Altair SLC software libraries/datasets	*<wpspfx>*
WPSHOME USS HFS directory for Altair SLC	*<wpshome>*
Target z/OS dataset name for the Altair SLC Distribution Library	*<wpsdlib>*

Altair SLC is supported on z/OS version 2.2, or later, on Architecture 10, or later.

Language Environment (LE) settings

Altair SLC makes use of the z/OS Language Environment (LE). It is important to be aware of your system's LE configuration. This information is only relevant when running Altair SLC.

Security considerations

As part of the installation process, you may need to alter USS directory and file settings. For more information see Permission Settings.

You should also consider protection of your resources using whatever security control software is installed (RACF, ACF2, Top Secret, and so on)

Installing Altair SLC

Upload the TERSE file from the PC to z/OS

After unzipping the distribution file, you must transfer the file that is named with a suffix of `-zos-s390.dlib.ter` i to your z/OS system.

Using whatever available file transfer mechanism (such as IND$FILE, FTP or the transfer file utility option of your 3270 emulator application) upload this file from your PC to z/OS. The DCB information for the target file should be .

```
DSORG=PS,RECFM=FB,BLKSIZE=27648,LRECL=1024
```

The `BLKSIZE` value may be different, depending on your SMS parameters, but the `DSORG`, `RECFM`, and `LRECL` values must be as specified above; the TERSE program will not process a file that has different DCB characteristics.

The upload process must take place in BINARY mode; there must not be any insertion or removal of carriage-return or line-feed characters; nor should there be any ASCII-to-EBCIDIC translation. Disregarding any of these conditions will result in a corrupted installation file that cannot be decompressed. The file currently requires approximately 7000 tracks on a device-type 3390 disk volume.

Unpacking the TERSE file

Having successfully uploaded `<wpspfx>.DLIB.TER` to your z/OS host system, you will need to run a job to unpack the DLIB from the TERSE file. The resulting PDS file currently requires approximately 13000 tracks of 3390 disk space, with two directory blocks defined. An example JCL to perform the unpacking is shown below:

```
// <add a suitable JOB statement here>
//*
//* Unpack the Altair SLC DLIB TERSE file
//*
//* (1) Add a suitable JOB statement
//* (2) Change <wpspfx> to the chosen filename prefix
//* (3) Submit this job and examine the results
//*
//UNPACK   EXEC PGM=TRSMAIN,PARM=UNPACK
//SYSPRINT DD   SYSOUT=*
//INFILE   DD   DSN=<wpspfx>.DLIB.TER,DISP=SHR
//OUTFILE  DD   DSN=<wpspfx>.DLIB,
//          DISP=(NEW,CATLG,DELETE),SPACE=(TRK,(15000,1,2),RLSE),
//          DSORG=PO,RECFM=FB,BLKSIZE=27920,LRECL=80
```

Copy this example into a file on the target system and make the changes detailed in the comments at the top. When ready, submit the job to the system for processing. If the job runs to successful completion, the resulting `SYSPRINT` output should be similar to:

```
** AMA572I STARTING TERSE DECODE   UNPACK       hh:mm:ss  mm/dd/yyyy ****
** AMA527I  INPUT  - DDNAME : INFILE   DSNAME: <wpspfx>.DLIB.TER
** AMA528I  OUTPUT - DDNAME : OUTFILE  DSNAME: <wpspfx>.DLIB
** AMA555I  THE VALUES ARE:  BLKSIZE= 27920   LRECL=80      PACKTYPE=PACK
 RECFM=FIXED
** AMA583I  INPUT DATASET SIZE IN BYTES: 347019264 OUTPUT DATASET SIZE IN BYTES:
 707626096 COMPRESSION RATIO: 49%
** AMA573I TERSE COMPLETE DECODE   UNPACK       hh:mm:ss  mm/dd/yyyy ****
** AMA504I  RETURN CODE: 0
```

The file extracted from `<wpspfx>.DLIB.TER` file is named `<wpspfx>.DLIB`. It is a PDS that contains members that are associated with installing and executing Altair SLC on the z/OS platform. Those members are described in the $README member.

Extract the z/OS components

You must extract the z/OS components to use Altair SLC software. Extraction of z/OS components is achieved by modifying and submitting the JCL provided in <wpspfx>.DLIB(@INSTALL). An example of this JCL is shown below:

```
// <add a jobcard here>
//*
//*-------------------------------------------------------------*/
//* @INSTALL : INSTALL THE Altair SLC DISTRIBUTION LIBRARIES
//*-------------------------------------------------------------*/
```

```
//*
//* (1) ADD A SUITABLE JOB CARD
//* (2) CHANGE <wpsdlib> TO THE DISTRIBUTION LIBRARY NAME
//* (3) CHANGE <wpspfx> IN ALL PLACES BELOW TO THE D/S PREFIX
//*     FOR Altair SLC INSTALL LIBRARIES
//*     OR ..
//* (3) CHANGE:
//*       *AUTOLIB, *CLIST, *CNTL, *FONTS, *INSTALL, *LICENSE, *LOAD,
//*       *MAPS, *README, *RELNOTE, *SASHELP, *THANKS and *USS
//*     TO:
//*       @AUTOLIB, @CLIST, @CNTL, @FONTS, @INSTALL, @LICENSE, @LOAD,
//*       @MAPS, @README, @RELNOTE, @SASHELP, @THANKS and @USS
//*     IF THESE HAVE BEEN PREALLOCATED
//* (4) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//*
//*
//STEP01   EXEC PGM=IEFBR14
//SETINIT  DD DISP=(NEW,CATLG),DSN=<wpspfx>.SETINIT,
//            DSORG=PS,RECFM=FS,LRECL=27998,BLKSIZE=27998,
//            SPACE=(TRK,1),UNIT=SYSDA
//*
//STEP02   EXEC PGM=IKJEFT1B,DYNAMNBR=999,COND=(0,NE)
//SYSEXEC  DD DISP=SHR,DSN=<wpsdlib>
//*AUTOLIB DD DISP=SHR,DSN=<autolib>
//*CNTL    DD DISP=SHR,DSN=<cntl>
//*CLIST   DD DISP=SHR,DSN=<clist>
//*FONTS   DD DISP=SHR,DSN=<fonts>
//*LOAD    DD DISP=SHR,DSN=<load>
//*MAPS    DD DISP=SHR,DSN=<maps>
//*SASHELP DD DISP=SHR,DSN=<sashelp>
//*USS     DD DISP=SHR,DSN=<usspax>
//*LICENSE DD DISP=SHR,DSN=<license>
//*INSTALL DD DISP=SHR,DSN=<install>
//*README  DD DISP=SHR,DSN=<readme>
//*RELNOTE DD DISP=SHR,DSN=<relnote>
//*THANKS  DD DISP=SHR,DSN=<thanks>
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSTSIN  DD DATA,DLM='++'
PROF NOPREFIX
RINST 'PFX(<wpspfx>)'
++ END OF //SYSTSIN
```

This member must be modified before being submitted to the system. Follow the instructions at the top of the file, substituting appropriate values where required.

The JCL invokes the batch-mode `TSO` program `IKJEFT1B` which in turn schedules the use of the `REXX` procedure named `RINST` which is supplied in `<WPSPFX>.DLIB` This `REXX` uses the `TSO RECEIVE` command to restore each of the files listed in the DD statements that appear as comments in the JCL.

The commented DD statements (for example, `//*CNTL...`, `//*LOAD...`) should be left as they are; the DCB and SPACE attribute information for each of the files will be supplied by the RINST procedure. If a greater level of control over the specification of these is required, the relevant files may be:

- Pre-allocated by using ISPF option 3.2, or perhaps a suitably-crafted IEFBR14 jobstream. In this case, the asterisk in the associated DD statement must be changed to a'@' character, or

- fully specified in the jobstream by adding the required DSN, DISP, SPACE and DCB clauses to the associated DD statement(s). In this case, the DD statements need to be uncommented.

The increased level of control has a cost associated with it; Altair SLC continually develops and grows in functionality. As a result, any SPACE values that are explicitly specified may be insufficient for use with future installations or upgrades; indeed, entire files may be added or deleted. The fact that a set of explicitly-defined attributes are good for one particular install of Altair SLC must not be taken as a guarantee that the same values will work in future upgrades of the software.

When the required changes have been made, the jobstream should be submitted to the system. With the latest version of Altair SLC, there are 13 files to be restored, of which 10 are PDS or PDSE libraries and the remaining three are physical sequential files. As the job progresses through the required work, a separate `SYSOUT` section for each of the library files being restored will be generated on the JES spool; the restores of the sequential files do not cause an output section to be generated. When the job is complete, there will be 11 outputs on the spool. There will be one piece of output for each library file that is restored, plus an extra piece for the overall progress log for all the restores. Each of these outputs will be identified by the same job name and number.

Warning:

When the job completes, a return code of zero **must not** be taken as an indication that all the restores have been successful.

To check for successful job completion, examine the `SYSTSPRT` section of the progress log.

The `SYSTSPRT` section is split into easily-identified parts, one for each file restored, for example:

```
/*============================*/
/* RECEIVING CNTL COMPONENTS */
/*============================*/

INMR901I Dataset WPCUK02.WPS.V4102.CNTL from BUILD on N1
INMR154I The incoming data set is a 'DATA LIBRARY'.
INMR906A Enter restore parameters or 'DELETE' or 'END' +
INMR908A The input file attributes are: DSORG=PARTITIONED, RECFM=FB, BLKSIZE=27920,
 LRECL=80, File size=281K bytes +
INMR909A You may enter DSNAME, SPACE, UNIT, VOL, OLD/NEW, or RESTORE/COPY/DELETE/END
INMR001I Restore successful to dataset 'WPS.V4102.B10692.CNTL'
```

The final `restore successful` message indicates that all is well for that particular file restore. There needs to be one such message for each of the files that are restored. A problem encountered with any single restore may also affect subsequent restores, even though the relevant message(s) indicate success. One single problem requires complete re-run of this job – do not attempt partial re-runs for individual files.

Extract the USS components

You do not need to extract the USS components if neither GRAPH or ODS output is used on your system, *and* Altair SLC is not used in a USS environment.

Support for TrueType fonts in Altair SLC is supplied via the USS components held within the <WPSPFX>.DLIB file. Without this support, any use of STYLE clauses in ODS (Output Delivery System) or in GRAPH specifications will result in ugly output. Additionally, PDF output generated by the ODS functions will be unreadable.

If you require the USS components, then the JCL held in <WPSPFX>.DLIB (@INSTUSS) must be modified and submitted to the system. An example of the jobstream, appears below:

```
// <add a jobcard here>
//*
//*-------------------------------------------------------------*/
//* @INSTUSS : INSTALL COMPONENTS INTO THE USS HOME DIRECTORY*/
//*-------------------------------------------------------------*/
//*
//* (1) ADD A SUITABLE JOB CARD
//* (2) CHANGE <wpsdlib> TO THE Distribution library name
//* (3) CHANGE <wpspfx> TO THE D/S PREFIX FOR INSTALL LIBRARIES
//* (4) CHANGE <wpshome> TO THE USS HOME Directory name
//* (5) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//*
//STEP01   EXEC PGM=IKJEFT1B,DYNAMNBR=999
//SYSEXEC  DD DISP=SHR,DSN=<wpsdlib>
//@USS     DD DISP=SHR,DSN=<wpspfx>.USS
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSTSIN  DD DATA,DLM='++'
PROF NOPREFIX
RINSTUSS WPSHOME(<wpshome>) WPSPFX(<wpspfx>)
++ END OF //SYSTSIN
```

Follow the instructions at the top of the file, substituting appropriate values where required. The *<WPSHOME>* value is a case-sensitive USS pathname. Care must be taken to ensure that this case-sensitivity is maintained when modifying the file.

The JCL invokes the batch-mode TSO program IKJEFT1B which in turn schedules the use of the REXX procedure named RINSTUSS supplied in <wpspfx>.DLIB. This REXX attempts to create the pathname specified by *<WPSHOME>*. When that is complete, the USS 'pax' program is employed to extract data from the .tar file held in <WPSPFX>.USS, and populate folders in the *<WPSHOME>* folder.

After the job is complete, review the output to ensure that everything was successful.

When done, the *<WPSHOME>* value must be made available to all users. The required pathname is specified as a start-up option in *<WPSPFX>.CNTL* (CONFIG). Examine that file and locate the . Modify the USSWPSHOME="" line in the *<WPSPFX>.CNTL* file and specify the *<WPSHOME>* value between the quotation marks, ensure that case-sensitivity is maintained.

Extract the SDK components

Use of the Software Development Kit may be necessary if the creation of specialised functions and possibly esoteric formats, amongst other things, is required. If there is no such requirement, then this step is completely optional.

A jobstream is supplied in <WPSPFX>.DLIB (@INSTSDK) to help accomplish this task. An example of this jobstream is shown below:

```
// <add a jobcard here>
//*
//*-------------------------------------------------------------------*/
//* @INSTSDK : INSTALL THE Altair SLC SDK LIBRARIES
//*-------------------------------------------------------------------*/
//*
//* (1) ADD A SUITABLE JOB CARD
//* (2) CHANGE <wpsdlib> TO THE DISTRIBUTION LIBRARY NAME
//* (3) CHANGE <wpspfx> IN ALL PLACES BELOW TO THE D/S PREFIX
//*     FOR  INSTALL LIBRARIES
//* (4) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//*
//STEP01   EXEC PGM=IKJEFT1B,DYNAMNBR=999,COND=(0,NE)
//SYSEXEC  DD DISP=SHR,DSN=<wpsdlib>
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//USRLD    DD DISP=(NEW,CATLG),DSN=<wpspfx>.USER.LOAD,
//            DSORG=PO,RECFM=U,LRECL=0,BLKSIZE=27998,
//            SPACE=(CYL,(1,1)),UNIT=SYSDA,DSNTYPE=LIBRARY
//SYSTSIN  DD DATA,DLM='++'
PROF NOPREFIX
RINSTSDK 'PFX(<wpspfx>)'
++ END OF //SYSTSIN
```

Follow the instructions at the top of the file, substituting appropriate values where required, and then submit the job for processing. On job completion, examine the output to ensure that the process was successful.

A total of eight files are created and populated by this job, all named with a high-level qualifier of <WPSPFX>.SDK,. An empty load library named <WPSPFX>.USER.LOAD is also created to be used for programs generated by the client.

Configure Altair SLC

The configuration tasks use members in the <WPSPFX>.CNTL library to make the installation ready for use. The <WPSPFX>.CNTL library also contains a number of parameter files that are used by Altair SLC.

Modify the supplied WPSPROC JCL member

Member WPSPROC in the <WPSPFX>.CNTL library is a suggested JCL procedure to be used for invoking Altair SLC in batch jobs. A copy of it appears below:

```
//*-------------------------------------------------------------------*/
//* WPSPROC : BATCH INTERFACE TO Altair SLC  */
//*-------------------------------------------------------------------*/
//*                                                                   */
//* (1) CHANGE <wpspfx> BELOW TO THE INSTALLATION DATASET PREFIX */
//*                                                                   */
//*-------------------------------------------------------------------*/
//*
//*-------------------------------------------------------------------*/
//* DEFINE WPSPROC AND DEFAULT ARGUMENTS
//*-------------------------------------------------------------------*/
//WPSPROC  PROC WPSPFX='<wpspfx>',           /* DATASET PREFIX   */
//  CONFIG=NULLFILE,                        /* USER CONFIG FILE     */
//  LOAD='*.NULLLOAD,VOL=REF=*.NULLLOAD',   /* DUMMY LOAD CONCAT    */
//  OPTIONS='',                             /* WPS OPTIONS          */
//  SASAUTO='*.NULLAUTO,VOL=EF=*.NULLAUTO', /* DUMMY SASAUTOS CONCAT*/
//  SYSPARM='',                             /* PROGRAM PARAMETERS   */
//  WORKDSN='&&WPSWORK',                    /* WORK DATASET NAME    */
//  WORKUNI=TRK,WORKPRI=450,WORKSEC=450     /* DEFAULT WORK SPACE   */
//*
//* EXECUTE WPSHOST
//WPS      EXEC PGM=WPSHOST,REGION=0M,
//         PARM=('SYSPARM=''&SYSPARM'' &OPTIONS')
//*
//* DEFINE NULL DDNAMES
//NULLLOAD DD DISP=(MOD,PASS),DSN=&&MTLOAD,UNIT=SYSDA,
//            SPACE=(TRK,(1,1,1)),LIKE=&WPSPFX..LOAD
//NULLAUTO DD DISP=(MOD,PASS),DSN=&&MTAUTO,UNIT=SYSDA,
//            SPACE=(TRK,(1,1,1)),LIKE=&WPSPFX..AUTOLIB
//*
//* DEFINE STEPLIB
//STEPLIB  DD DISP=(SHR,PASS),DSN=&LOAD
//         DD DISP=SHR,DSN=&WPSPFX..LOAD
//*
//* DEFINE WORK DDNAME
//WORK     DD DISP=(NEW,DELETE),DSN=&WORKDSN,
//            SPACE=(&WORKUNI,(&WORKPRI,&WORKSEC))
```

```
//*
//* DEFINE Altair SLC-SPECIFIC DDNAMES
//CONFIG   DD DISP=SHR,DSN=&WPSPFX..CNTL(CONFIG)
//         DD DISP=SHR,DSN=&CONFIG
//MAPS     DD DISP=SHR,DSN=&WPSPFX..MAPS
//NEWS     DD DISP=SHR,DSN=&WPSPFX..CNTL(NEWS)
//SASAUTOS DD DISP=(SHR,PASS),DSN=&SASAUTO
//         DD DISP=SHR,DSN=&WPSPFX..AUTOLIB
//SASHELP  DD DISP=SHR,DSN=&WPSPFX..SASHELP
//SASLIST  DD SYSOUT=*
//SASLOG   DD SYSOUT=*,RECFM=VBA,LRECL=137,BLKSIZE=141
//SETINIT  DD DISP=SHR,DSN=&WPSPFX..SETINIT
//WPSFONTS DD DISP=SHR,DSN=&WPSPFX..FONTS
//WPSTRACE DD SYSOUT=*
//*
//* DEFINE LANGUAGE ENVIRONMENT (LE) DDNAMES
//CEEDUMP  DD SYSOUT=*
//CEEOPTS  DD DISP=SHR,DSN=&WPSPFX..CNTL(CEEOPTS)
//CEERPT   DD SYSOUT=*
//*
//* DEFINE SORT DDNAMES
//DFSPARM  DD DISP=SHR,DSN=&WPSPFX..CNTL(DFSPARM)
//SORTMSGS DD SYSOUT=*
//*
//* DEFINE DB2 DDNAMES
//WPSAOINI DD DISP=SHR,DSN=&WPSPFX..CNTL(WPSAOINI)
//DSNAOINI DD DISP=(NEW,DELETE),DSN=&&DSNAOINI,
//            DSORG=PS,RECFM=FB,LRECL=80,
//            SPACE=(TRK,1),UNIT=SYSDA
//*
//* DEFINE SYSPRINT AND SYSOUT
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//         PEND
```

Follow the instruction at the top of the member to change the *<WPSPFX>* value and then save the member back in the `<WPSPFX>.CNTL` library.

This member should be available to all system users to invoke Altair SLC; for this reason, the file should be copied to a library on the system-wide `PROCLIB` concatenation. The member should not be moved from the `<WPSPFX>.CNTL` library since it is expected to be present there in subsequent jobs.

The JCL in this member does not change very often, but it will change as time goes by. It is therefore inadvisable to rely upon a version of the procedure from a previous version of Altair SLC. If basic JCL-type errors are experienced after an upgrade of Altair SLC, one of the earliest checks to be carried out is to test the currency of the WPSPROC being used.

Apply the licence key

A valid licence key must be applied to activate your installed copy of Altair SLC. This requires the special execution of the `SETINIT` procedure using the values contained in your site-specific licence key file.

New licence keys may be issued from time to time and re-applied using the same process discussed in the section.

The license key is a plain text file available at the Altair web site. License keys are individual to each client and may be retrieved for use by a person who is identified as an *authorised user* for the client. The authorised user can log on to the Altair web site and register themselves with an ID and password, and then Altair will grant access to the license key page for the specified user.

When the authorised user subsequently logs on, they will be able to download a copy of the licence key file to their desktop system. From here, the contents of the license key file must be copied in to the `<WPSPFX>.CNTL` (XSETINIT) member.

Note:

Ensure that the modified XSETINIT contains text that is *exactly* the same as the contents of the downloaded file. There must be no transposing of upper-case text to lower-case and vice-versa, and there must be no truncation or splitting of the lines of text.

Most important of all, do not modify the value in the `PASSWORD` line.

A valid key file normally contains more lines of text than the example file held in `<WPSPFX>.CNTL` (XSETINIT), so care must be taken when copying the file contents. Ignoring any of these conditions will result in a corrupted license key, which will prevent subsequent use of Altair SLC.

Having modified the XSETINIT file, the key must now be applied by using the JCL held in `<WPSPFX>CNTL`(@SETINIT). A copy of this JCL appears below:

```
// <add a jobcard here>
//PROCLIB  JCLLIB ORDER=(<wpspfx>.CNTL)
//*
//*--------------------------------------------------------------------*/
//* WPS SETINIT JOB
//*--------------------------------------------------------------------*/
//*
//* (1) ADD A SUITABLE JOBCARD
//* (2) CHANGE <wpspfx> TO THE WPS INSTALLATION DATASET PREFIX
//* (3) CORRECTLY CONFIGURE <wpspfx>.CNTL(WPSPROC)
//* (4) PLACE THE SETINIT LICENSING CODE, OBTAINED FROM WORLD
//*     PROGRAMMING,  INTO THE 'XSETINIT' MEMBER OF THIS DATASET
//* (5) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//* (6) CHECK FOR A JOB RETURN CODE OF ZERO
//*
//*--------------------------------------------------------------------*/
//*
//@SETINIT EXEC WPSPROC,OPTIONS='SETINIT'
//SYSIN    DD DISP=SHR,DSN=<wpspfx>.CNTL(XSETINIT)
```

Follow the instructions at the start of this file, then submit the job for processing. The file named `<WPSPFX>.SETINIT` will be updated to reflect the data held in the license key file. The integrity of `<WPSPFX>.SETINIT` is vital to continued use of Altair SLC, and any subsequent modification or corruption of file will cause Altair SLC to cease working. We therefore recommend the file needs to be strongly protected and/or backed-up.

While applying a new licence key or re-installing an existing licence key is uncomplicated, the process is not something that should be performed in an ‘emergency recovery’ situation.

The `//PROCLIB JCLLIB ORDER=(<wpspfx>.CNTL)` statement, indicates the required WPSPROC JCL is in the `<WPSPFX>.CNTL` library.

When complete, examine the output to ensure that the job terminated with return code 0 (zero) and that the message

```
NOTE: Setinit applied successfully
```

appears in the `SASLOG` output section.

Installation verification

Member @VERIFY in `<WPSPFX>.CNTL` provides JCL that runs a sample Installation Verification script. The script is also held in `<WPSPFX>.CNTL`, as member XVERIFY. A copy of the @VERIFY JCL appears below:

```
// <add a jobcard here>
//PROCLIB  JCLLIB ORDER=(<wpspfx>.CNTL)
//*
//*-------------------------------------------------------------------*/
//* SAMPLE JOB TO VERIFY Altair SLC INSTALLATION                        */
//* BY RUNS THE INSTALLATION VERIFICATION PROGRAM (XVERIFY)           */
//*-------------------------------------------------------------------*/
//*
//* (1) ADD A SUITABLE JOBCARD
//* (2) CHANGE <wpspfx> TO THE Altair SLC INSTALLATION DATASET PREFIX
//* (3) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//* (4) CHECK FOR A JOB RETURN CODE OF ZERO
//*
//*-------------------------------------------------------------------*/
//*
//@VERIFY  EXEC WPSPROC
//SOURCLIB DD DISP=SHR,DSN=<wpspfx>.CNTL
//SYSIN    DD DATA,DLM='++'
OPTIONS SOURCE2;
ODS LISTING;
%INCLUDE WPSIN;
++ END OF //SYSIN
//*
//WPSIN    DD DATA,DLM='++'
%INCLUDE SOURCLIB(XVERIFY);
++ END OF //WPSIN
```

Follow the instructions at the top of the member, and submit the job for processing. The XVERIFY script is intended to show that Altair SLC can be used in batch mode.

If the return code of the job is non-zero, then one or more steps of the install procedure performed so far have not been completed correctly. Any such problems must be corrected before attempting to continue with use of Altair SLC. Re-run the job, look at the output and ensure that there are no errors or other important diagnostic messages.

Initialise the TrueType font cache

This step is only necessary if the ODS (Output Delivery System) or the Graphics output facility of Altair SLC are used. If PDF output is generated via ODS, then availability of TrueType fonts is a pre-requisite. Support for TrueType fonts is provided via use of USS facilities, so the USS Components must be installed..

The Altair SLC software includes a single TrueType font family named 'Vera', provided in `<WPSPFX>.FONTS`. Other TrueType fonts and families are available on z/OS systems, and further fonts may be separately licensed. Any fonts to be used by Altair SLC must be identified in the font cache before first use. Failure to do this will result in output that varies in quality from untidy to completely unreadable.

Member `@FONTCFG` in the `<WPSPFX>.CNTL` library is intended for (re)initialisation of the shared font cache, in the USS Altair SLC installation. A copy of the JCL appears below:

```
//*<add a jobcard here>
//PROCLIB  JCLLIB ORDER=(<wpspfx>.CNTL)
//*
//*-------------------------------------------------------------------*/
//* SAMPLE JOB TO PERFORM SHARED FONT CONFIGURATION AND CACHING      */
//* IN THE USS Altair SLC INSTALLATION FOR USE BY ODS PDF                  */
//*-------------------------------------------------------------------*/
//*
//* (1) ADD A SUITABLE JOBCARD
//* (2) CHANGE <wpspfx> TO THE INSTALLATION DATASET PREFIX
//* (3) CHANGE <wpshome> TO THE USS HOME DIRECTORY NAME
//* (4) SUBMIT THIS JOB AND THEN CHECK THE OUTPUT
//* (5) CHECK FOR A JOB RETURN CODE OF ZERO
//* (6) LOOK IN THE <wpshome>/etc/fonts/fonts.conf FILE TO VALIDATE
//*     THE FONTS THAT HAVE BEEN IDENTIFIED
//* (7) VALIDATE THAT THE <wpshome>/fontconfig/ DIRECTORY HAS BEEN
//*     CREATED
//*
//* IN ORDER FOR CLIENT JOBS TO USE THIS SHARED FONT CACHE
//* THEY MUST BE RUN WITH THE SYSTEM OPTION USSWPSHOME=<wpshome>
//*
//*-------------------------------------------------------------------*/
//*
//@FONTCFG  EXEC WPSPROC,
//            OPTIONS='CONFIGFONT CONFIGFONTMVS=<wpspfx>
//              USSWPSHOME="<wpshome>"'
//PDFOUT    DD DSN=&PDFOUTF,DISP=(NEW,DELETE,DELETE),
//             RECFM=VB,BLKSIZE=100,LRECL=50,DSORG=PS,
//             SPACE=(TRK,(10,10))
//SYSIN    DD DATA,DLM='++'
ODS PDF FILE=PDFOUT;
DATA A; A=1; RUN;
PROC PRINT DATA=A; RUN;
ODS _ALL_ CLOSE;
++ END OF //SYSIN
```

Follow the instructions at the top of the file to modify this JCL before submission. Ensure you correctly specify the `OPTIONS` clause that follows `EXEC WPSPROC`, an incorrect OPTIONS clause will result in a JCL error, `EXCESSIVE PARAMETER LENGTH IN THE PARM FIELD`.

When the job runs successfully, the centralised, shared, font cache in the Altair SLC installation is initialised and ready for use. For more information on continuing JCL statements refer to the IBM manual *z/OS MVS JCL Reference*, catalogue number SA23-1385, and specifically the section titled *Continuing parameter fields enclosed in apostrophes*.

If this JCL is used to be used to reinitialise the font cache to make new fonts available, then those fonts must be identified in the `<wpshome>/etc/fonts/fonts.conf.in` file prior to submitting the job. The first few lines of this file are shown below:

```
<?xml version="1.0"?>
<!DOCTYPE fontconfig SYSTEM "fonts.dtd">
<!-- /etc/fonts/fonts.conf file to configure system font access -->
<fontconfig>

<!-- Font directory list -->

  <!-- to locate fonts in the USS Altair SLC installation, uncomment the line below
 -->
  <!-- dir>WPS_INSTALL_DIR/fonts</dir -->

  <!-- to locate the fonts shipped with Altair SLC -->
  <file>//'WPS_MVS_LOCN.FONTS(VERA)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERABD)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERABI)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERAIT)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERAMOBD)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERAMOBI)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERAMOIT)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERAMONO)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERASE)'</file>
  <file>//'WPS_MVS_LOCN.FONTS(VERASEBD)'</file>
...
```

Any new fonts to be made available must be identified in this part of the file. Do not alter any other part of the file.

You must not modify the `fonts.conf` file held in the same folder. That file must be left untouched – it is the output of the cache charging process.

There are two different ways of identifying new fonts, either by specifying individual fonts between `<file>…</file>` tags, or an entire folder of fonts, specified between `<dir>…</dir>` tags.

On z/OS systems, a set of fonts are made available in `/usr/lpp/fonts/worldtype`. If this folder is not present,contact your local system administrators for assistance.

PDSs such as SYS1.FONTLIB, SYS1.FONTLIBB and SYS1.FONT300 may be present and their contents may also be used, but in these cases, the files within will need to be copied into a USS folder and that folder specified in the `<wpshome>/etc/fonts/fonts.conf.in` file.

With IBM-supplied font libraries of either type, the accepted font file naming convention has been replaced such that the actual font held within a file is not immediately identifiable. File-name to font-name equivalence tables are available in IBM manual *z/OS Font Collection*, publication number GA32-1048.

Once the new fonts are identified in `<wpshome>/etc/fonts/fonts.conf.in`, then re-run the `<WPSPFX>.CNTL` (@FONTCFG) job to re-initialise the font cache and make the new fonts available for use.

Once the @FONTCFG job has completed, examine the output to ensure that it ran to successful completion.

The font cache file can be found at `<wpshome>/fontconfig`. This file needs to be identified to make it available to all Altair SLC users who make use of the `STYLE` option in various statements, and/or need to generate PDF documents. The relevant pathname is specified as a start-up option in `<WPSPFX>.CNTL` (CONFIG). Examine this member to locate the `FONTCACHEDIR=""` line, and insert the `<wpshome>/fontconfig` pathname specification between the quotation marks. Ensure that case-sensitivity is maintained.

The default CONFIG file

Most of the system options available may be specified in an `OPTIONS` statement in the SAS language program source. Some options, however, modify the run-time environment of Altair SLC, so they must be specified before the SAS language program starts. This is achieved by the required options being specified via the OPTIONS parameter in the WPSPROC JCL and/or via a file of such options identified by the CONFIG DD statement in the WPSPROC JCL. This statement itself features a possible override that can be specified via the `CONFIG` option in WPSPROC, so there are a number of different ways of specifying options that may be required. The `CONFIG` file does not need to be the home of system-wide start-up options exclusively. Other available options may be specified here too.

To list the current SAS language system option settings during an Altair SLC session, use the `OPTIONS` procedure statement.

Example 1

```
// <job statement>
//<stepname> EXEC WPSPROC,CONFIG=<my_configs_dsn>
//SYSIN DD *
<program script>
```

will result in a CONFIG DD statement of:

```
//CONFIG   DD DISP=SHR,DSN=&WPSPFX..CNTL(CONFIG)
//         DD DISP=SHR,DSN=<my configs dsn>
```

<my configs dsn> may be a physical sequential file or it may be a member within a PDS. If any options are specified in both the files, the later specification takes effect and overrides the earlier one(s)

Example 2

In this example, the install-supplied list of default options will be completely ignored. It is to be hoped that *<my configs dsn>* contains sufficient detail to allow the program script to run to expected completion.

```
// <job statement>
//<stepname> EXEC WPSPROC
//CONFIG DD DISP=SHR,DSN=<myconfigs dsn>
//SYSIN DD *
<program script>
```

Example 3

In this example, the `WORKINIT` option will override any similar specification in files associated with the CONFIG DD statement.

```
// <job statement>
//<stepname> EXEC WPSPROC,OPTIONS='WORKINIT'
//SYSIN DD *
<program script>
```

The default DFSPARM file

`<WPSPFX>.CNTL` (DFSPARM) is a list of options to be used by the host `SORT` program, if it is invoked. The very basic set of options provided by default may be enhanced if considered necessary.

There are three options associated with `SORT` that are held in the `CONFIG` file.

- `SORTCUTP` - works in conjunction with the `SORTPGM` option value. The value defines the point at which the `WPS` internal sort is used in preference to whichever sort utility is specified by the `SORTPGM` option.
- `SORTPGM` - may be set to `HOST`, `BEST` or `WPS`. If `WPS` is specified, the internal sort program provided with Altair SLC is always used. If `HOST` is specified, then whichever sort utility is installed on the host system is used. If `BEST` is specified, then the decision to use `HOST` or `WPS` sort is based on the `SORTCUTP` option value.
- `SORTSIZE` - defines the amount of memory that may be used for sorting data. This option applies to both the `WPS` sort and whichever sort utility is installed on the host system. The amount specified will be taken from the region size specified for the batch job.

A `DFSPARM` DD statement may be used to override the specification in the WPSPROC JCL.

The default NEWS file

`<WPSPFX>.CNTL` (NEWS) contains some text that is copied to the `SASLOG` output section at the start of every Altair SLC session. It may easily be changed to feature something more site-specific.

The file location may be overridden by a NEWS DD statement

The default CEEOPTS file

The default <WPSPFX>.CNTL (CEEOPTS) provides a list of Language Environment options, all of which are commented out. Modifying any of these options is possible but mostly inadvisable.

On z/OS, Altair SLC makes use of the LE (Language Environment) heap storage pool facility to improve the performance of memory management activities. In general, this is beneficial, but it does come at the cost of a slightly elevated baseline memory usage, as more memory is required to manage the heap storage pools.

If you wish to switch off the usage of heap storage pools, ensure that the content read from the CEEOPTS DD contains the line HEAPPOOLS (OFF).

The heap storage pool facility can be dynamically tuned to improve the performance associated with particular application memory usage patterns and footprint. The default settings been selected to provide a reasonable balance of performance for the majority of Altair SLC usage. However, it may be that, for a particular workload, the performance can be improved.

The heap storage pool facility can be configured to report the optimum settings associated with a particular run of the application. Once an alternative heap storage pool configuration has been obtained, the CEEOPTS can be modified so that future job runs always use this configuration. The CEEOPTS can either be tuned for a single job, so that those settings are always used for that job, or the required settings can be applied to the default CEEOPTS for all Altair SLC runs using WPSPROC.

For more information about how to tune the heap storage pool facility configuration, see the *Tuning heap storage* section of the *IBM z/OS LE programming guide*.

Using Altair SLC

Executing SAS language programs

The @verify member of PDS `<wpspfx>.CNTL` that was used for installation verification may be used as a starting point for running any other SAS language programs.

Files created during execution

The Altair SLC Batch Job interface creates and references a number of z/OS datasets, in addition to any that may be created by user-written programs.

The default files are `SASLOG` for the progress log generated as a script is processed and `SASLIST` for the default output listing.

There are other `SYSOUT` DD statements in the WPSPROC JCL that may contain diagnostic output from Altair SLC.

Examples of SAS language programs and the required execution JCL

The `<wpspfx>.CNTL` library contains some example programs, together with the JCL required to execute them to demonstrate the capabilities of Altair SLC.

For example, `@SEQFB` is a simple program that reads and writes file data that has a record format of 'fixed, blocked'. Similarly, `@SEQVB` reads and writes data of a 'variable, blocked' nature.

Using SAS System data libraries

Altair SLC can read z/OS-based SAS System dataset libraries directly using the `SASDASD` library engine. However, Altair SLC is unable to write to the `SASDASD` format. Therefore, persistent SAS System dataset libraries that get updated will need to be migrated to Altair SLC dataset libraries, which can be achieved using the `COPY` procedure. For more information see the *Migration Guide for z/OS*.

World Programming is able to provide consultancy on the migration of data. It is important to consider data migration before any existing environment that is to be run in parallel becomes unavailable. The following sections are a summary of the data formats that can be accessed.

Using XPORT files

Altair SLC is able to read and write XPORT files using the `XPORT` data library engine.

Using SAS7BDAT files

Altair SLC is able to read and write SAS7BDAT (SAS v7/8) files using the `SASBDATA` data library engine.

Using SD2 files

Altair SLC is able to read SD2 (SAS v6) files using the `SD2` data library engine.

Using SASTAPE files

Altair SLC is able to read `SASTYPE` (SAS v6) files using the `SASTAPE` data library engine.

Using RDBMS (DB2, SQLServer)

Altair SLC is able to read and write data for a variety of relational database management systems (RDBMS) such as SQL Server (Microsoft) and DB2 (IBM). For more information about using DB2 withAltair SLC, see Using DB2

Using CPORT files

Altair SLC is able to read and write CPORT files using `CPORT` and `CIMPORT` procedures.

Running existing SAS language programs

Many existing SAS language programs will run unaltered. However, other programs may require modification depending on the complexity and nature of the programs.

Migrating existing Altair SLC data

Typically, data produced by earlier versions of Altair SLC can be accessed transparently. There may, however, be performance benefits associated with moving the data to a new library created by the latest version of Altair SLC. Refer to the release notes for information.

Migrating existing SAS language programs

Existing SAS language programs written/run with previous versions of Altair SLC are typically compatible with the latest version of Altair SLC. However, you should first refer to the relevant release notes before running older programs.

Using MXG

MXG is an application written by Merrill Consultants. If you are considering using Altair SLC with MXG, you will find relevant information in the *Migration Guide for z/OS*

Using DB2

Altair SLC can connect to DB2 using the DB2 Call Level Interface (CLI) provided by IBM.

Altair SLC is a 31-bit XPLINK application, and specifically requires the XPLINK version of the CLI (DSNAOCLX). To make a connection to DB2, the target DB2 installation must include support for the DB2 CLI.

The CLI must be bound into the DB2 sub-system in use. Refer to the DSNTIJCL member in the DB2 sample library DSN*x*10.SDSNSAMP for an example job that accomplishes this process. The ‘x’ in this file name must be replaced by character associated with the specific DB/2 version number in question (‘A’ for version 10, ‘B’ for 11, and so on).

For detailed information on installing and setting up the DB2 runtime environment to enable support for the DB2 CLI, refer to the *ODBC Guide and Reference* available from the IBM website. Links to the different versions of the DB2 for z/OS manuals are given at: http:// www-01.ibm.com/support/docview.wss?uid=swg27039165.

Once you have installed the DB2 CLI on your mainframe, there is an additional configuration step required to connect to DB2: three additional libraries must be included in your steplib. (We recommend that you do this on an installation-wide basis by modifying the steplib in the WPSPROC member of the <wpspfx>.CNTL library).

Before you edit the steplib in wpsproc, it will look like this:

```
//* DEFINE STEPLIB
//STEPLIB DD DISP=(SHR,PASS),DSN=&LOAD
//         DD DISP=SHR,DSN=&WPSPFX..LOAD
```

You need to add the following three libraries to the steplib:

```
//* DEFINE STEPLIB
//STEPLIB DD DISP=(SHR,PASS),DSN=&LOAD
//         DD DISP=SHR,DSN=&WPSPFX..LOAD
//         DD DISP=SHR,DSN=DSNx10.SDSNLOD2
//         DD DISP=SHR,DSN=DSNx10.SDSNEXIT
//         DD DISP=SHR,DSN=DSNx10.SDSNLOAD
```

Once again, the lower-case '*x*' in the above file names must be replaced by the DB/2 version number indicator

Once you have completed this step, you should be able to connect to DB2. Please make sure you check for DB2/CLI connectivity on your z/OS system before you attempt to access the database using a `LIBNAME` statement specifying `DB2` as the engine name, or by using a `PROC SQL` connect statement.

Using sequential engines on tape devices

Altair SLC supports the writing of complete sequential libraries (wpsseq, sasseq, xport) to tape, although such a library can only be written to as a complete entity in one step, be it a `DATA` or a procedure step.

To work around this limitation, multiple `PROC COPY` statements or `COPY` statements in a `DATASETS` procedure can be merged into `COPY` statements in a `DATASETS` procedure copies datasets from multiple input libraries to one tape sequential library. For example:

```
PROC COPY IN=INLIB1 OUT=SEQLIB; RUN;
PROC COPY IN=INLIB2 OUT=SEQLIB; RUN;
```

could be rewritten as:

```
PROC DATASETS LIB=INLIB1 NOLIST NODETAILS;
COPY IN=INLIB1 OUT=SEQLIB;
COPY IN=INLIB2 OUT=SEQLIB;
RUN;
```

Without making this change each subsequent `DATA` or procedure step would write a completely new version of the library, overwriting the previous version, containing only the content placed into the library by that step.

Sharing data between multiple Altair SLC users

The processing of data files in Altair SLC data libraries in sequential z/OS datasets is controlled through the `DISP` option of the relevant dd or libname statement.

If the contents of a data library are to be changed in any way, then the `DISP` value should be specified as either new (when creating a new data library) or old (when modifying the contents of an existing one). Specifying `DISP=SHR` automatically prevents modification of the data library, resulting in a read only library within Altair SLC. Data files in a read only library, allocated with `DISP=SHR`, may only be used as input to the program.

The concurrent processing of data files in a data library held in a VSAM linear z/OS dataset is controlled through the `SHAREOPT` option. Nevertheless, only one user should be allowed to update data files in the library at any point in time. For that reason, the first ('crossregion') parameter value should be set to '1',for example `shr(1, n)`.

For USS-based data libraries, the access mode setting for the directory, and contained files is key. Altair SLC uses the permissions granted to the library files when running a SAS language program and coordinates the concurrent access to the files using system-level file locking, to prevent multiple programs from writing to the same file.

The WORK library

The `WORK` data library for each user should be unique and not shared.

When using Altair SLC on z/OS, the supplied JCL procedure will allocate a temporary work data library each time it is executed. This can be overridden within the JCL.

When using a USS directory-based `WORK` library, when a SAS language program is submitted, a new unique temporary directory is created below the USS work directory specified. This temporary directory will be deleted on completion of the program.

To change the work library to use a USS directory location instead of a z/OS dataset, edit the wpsproc member of the cntl library. Change the work dd name to point to a USS directory. The entry should be similar to the example below:

```
//WORK DD PATHDISP=(KEEP,KEEP),PATH='/u/wps/work'
```

Placement of data libraries

an Altair SLC data library can be held within a sequential z/OS dataset, a VSAM linear z/OS dataset, or a USS (UNIX System Services) HFS/zFS directory. Whilst using USS files offers certain advantages over sequential z/OS or VSAM linear z/OS datasets, particularly when copying, renaming, and transferring using UNIX utilities, such files are far from ideal, as HFS volumes cannot be allocated, used, and de-allocated within a JCL script in the same way that a z/OS file can be.

The default Altair SLC dataset engine is called `WPD`. Altair SLC will automatically detect whether the library holding a dataset is contained within a z/OS dataset, a VSAM linear z/OS dataset, or a USS directory, and operate on the library accordingly. All three types of library can be used in a single SAS language program.

For details of the syntax related to data libraries supported by Altair SLC, refer to the *Reference for Language Elements*.

The default `WORK` library is a z/OS sequential dataset-based library. The default `SASHELP` library is provided as a permanent z/OS sequential dataset-based library.

Sequential z/OS dataset-based data libraries

an Altair SLC data library in a sequential z/OS dataset is represented by a single binary structure. Each individual dataset is held in this structure. The structure of the library and its member datasets is a format proprietary to Altair SLC; datasets can only be added, deleted and moved from a native z/ OS file data library using Altair SLC.

For sequential z/OS dataset libraries, we recommend attributes of half-track blocking and the record format should be undefined. As such, the record length is therefore effectively immaterial. For example, when initially creating the file on a 3390 disk, specify the following for the best results:

```
DCB=(DSORG=PS,RECFM=U,BLKSIZE=27998,LRECL=27998)
```

Storage of a dataset using z/OS datasets is best configured through use of DD statements in the JCL used to launch Altair SLC. The DDNAME on such a statement becomes an implicit Altair SLC Library identifier in the program and can therefore be used directly as if a `LIBNAME` statement has been used.

For instance if a DDNAME *mylib* is declared, then a dataset called *dataset1* in the z/OS dataset referred to by *mylib* can be referenced as *mylib.dataset1*

VSAM Linear z/OS dataset-based data libraries

An Altair SLC data library within a VSAM linear z/OS dataset is very similar to a z/OS sequential dataset in terms of internal structure. Each individual dataset is held within the structure, which is in a format that is proprietary to Altair SLC; datasets can only be added, deleted and moved by using Altair SLC.

Initially, defining the VSAM linear z/OS dataset would normally be performed via an idcams define similar to:

```
DEFINE CLUSTER(NAME(<LDS_name>) LINEAR CYLINDERS(<pri sec>) SHAREOPTIONS(1,3))
```

Subsequently, a DD statement is used to form the association between a library name and the LDS_name.

USS directory-based data libraries

an Altair SLC data library can be represented by a USS directory, with each dataset within the library being represented by a single file with a file extension of `.wpd` z/OS datasets can be added to, and removed from, the library by use of file manipulation tools within USS such as cp (copy), mv (move), rm (delete), and so on.

When the contents of a USS file-based data library are listed, the list of members returned is the list of files in the associated directory with the following extensions:

- `wpd` = Altair SLC Dataset
- `wpccat` = Altair SLC Catalogue
- `wpcvw` = Altair SLC View
- `wpdidx` = Altair SLC Index

Before using a USS directory-based data library, an HFS or zFS volume must be allocated and mounted into the USS file system. The user must also have sufficient privileges to perform the operations they require on the library.

libraries can be defined using, for example, the libname statement. Before issuing a libname statement, the USS directory to which the statement refers must exist.

USS permissions

To create a dataset, you must have read, write and execute permissions on the USS directory.

To read a dataset, you must have read and execute permissions on the USS directory and read permissions for the `.wpd` file.

Altair SLC data files

Datasets are stored in either a z/OS dataset or a USS directory. The `WORK` library may be defined as either of these types of library. Consider the following example program :

```
LIBNAME mylib '/u/<userid>/wpsdata';
DATA mylib.data1;
  A = 1;
RUN;
```

This will create a data file called `/u/<userid>/wpsdata/mydata.wpd`.

Under the USS file system, path, directory and file names are case sensitive.

Data can be imported to and exported from files and DB2 tables if necessary. For more information about using DB2 with Altair SLC, see Using DB2.

Alternatively, assuming that a ddname of mylib1 is defined in the JCL used to launch the program, and the ddname points to a z/OS dataset containing a data library, then the following program could be used to access a dataset within the mylib1 z/OS dataset:

```
DATA mylib1.data1;
  A = 1;
RUN;
```

Font configuration

Using TrueType fonts for ODS PDF or Graphing.

Altair SLC uses TrueType fonts for ODS PDF and graphing output on z/OS. To use TrueType fonts, you must install the USS component of Altair SLC. One font family (Vera) is supplied with Altair SLC, but other font families can be used.

TrueType font files can be used with many applications on virtually all operating systems. The *OpenType* standard is the successor of TrueType, and *WorldType* fonts are TrueType and OpenType fonts that are supplied in a Microsoft Unicode format.

Fonts may be supplied by software vendors for use with application software subject to licence terms and conditions. Fonts can also be purchased and downloaded from various font vendors.

To use TrueType fonts with Altair SLC

The `<wpspfx>.CNTL(@FONTCFG)` job is automatically run as part of the @CONFIG job during Altair SLC installation. The user who submits the `<wpspfx>.CNTL(@FONTCFG)` job requires write permissions; other users only require read permissions.

Before the `@FONTCFG` job is submitted, check permissions on the following folders and files:

- `<wpshome>/fontconfig`. The location for the font cache. This may not exist before running the job

- `<wpshome>/etc/fonts`. The location for certain font files.
- `<wpshome>/etc/fonts/fonts.conf`. The Altair SLC font configuration file.
- `<wpshome>/etc/fonts/fonts.conf.in`. The Altair SLC font configuration file template.

Where `<wpshome>` is the location of the USS components of Altair SLC.

1. Ensure Altair SLC is installed. For more information see *Installing Altair SLC* (page 8).
2. Install the USS components of Altair SLC. For more information see *Extract the USS components* (page 11).

   Ensure that the correct `<wpshome>` location is specified in the *@INSTUSS* jobstream used to install the USS component.
3. Add details for the TrueType fonts you want to use with Altair SLC to the `<wpshome>/etc/fonts/fonts.conf.in` file. See *Adding fonts* (page 31) for more information.
4. Run the *<wpspfx>.CNTL(@FONTCFG)* job to create the central cache of fonts.

   Where `<wpspfx>` is the Altair SLC prefix for the location of the installation.
5. Any job that uses `ODS PDF` must be specify the `USSWPSHOME` system option set to the location of the USS Altair SLC components.

Once the `<wpspfx>.CNTL(@FONTCFG)` job has successfully completed, `ODS PDF` output or graphing is available.

If fonts are not set up correctly then you may see the message “failed to choose a font, expect ugly output”. In this case, *ugly output* displays unknown characters as small rectangles.

The Font Cache

Fonts to be used by Altair SLC must be in the *font cache*. The default location is for the font cache is `<wpshome>/fontconfig`. This cache must be re-initialized when a new version of Altair SLC is installed, or when new font information is added to the `<wpshome>/etc/fonts/fonts.conf.in` file.

The XML file `<wpshome>/etc/fonts/fonts.conf` defines which fonts are to be loaded into the cache. Do not edit this file as the font config job rewrites the content. A separate file `<wpshome>/etc/fonts/fonts.conf.in` should be used to list the fonts to be loaded into the cache. Typically USS directories containing families of font files are listed, but for z/OS, individual font files are listed.

When the font cache is re-initialized, the files are used to assemble the cache from all those listed directories (explored recursively) and from the individually listed z/OS libraries.

Adding fonts

Although Altair SLC can support any TrueType font, a single font family *Vera* is supplied. The default library <wpspfx>.FONTS generated with @INSTALL JCL contains the following members associated with the Vera font:

- VERA
- VERABD (bold)
- VERABI (bold italic)

- VERAIT (italic)
- VERAMOBD (monospace bold)
- VERAMOBI (monospace bold italic)
- VERAMOIT (monospace italic)
- VERAMONO (monospace)
- VERASE (serif)
- VERASEBD (serif bold)

Any other font files listed in `<wpshome>/etc/fonts/fonts.conf.in` should follow the same naming conventions. A USS directory can be added to this configuration file, using the XML syntax shown in the template, for example `<dir>path-to-fonts-directory</dir>`.

If you have the AFP WorldType fonts in `/usr/lpp/fonts/worldtype`, then this directory path can be included in the `<wpshome>/etc/fonts/fonts.conf.in` file.

IBM's AFP outline and raster fonts – TrueType, OpenType, and WorldType – may be found in locations such as `SYS1.FONTLIB`, `SYS1.SFNTILIB`, `SYS1.FONTLIBB`, `SYS1.FONTLIB`, `SYS1.SFONDLIB`, and `SYS1.FONT300`. Not all fonts and locations have been tested. but any TrueType fonts are expected to be usable in ODS output.

The relationship between file names and the IBM font names are described on the IBM Knowledge Center: https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzalu/rzalucontruetype.htm.

Re-initializing the font cache

To add new TrueType fonts, and the required details to the `<wpshome>/etc/fonts/fonts.conf.ini` and use the supplied `<wpspfx>.CNTL(@FONTCFG)` jobstream to re-initialize the font cache.

The `<wpspfx>.CNTL(@FONTCFG)` jobstream uses two system options that must NOT be specified in the `<wpspfx>.CNTL(CONFIG)` file:

- `CONFIGFONT`, and
- `CONFIGFONTMVS=<wpspfx>`

Specifiying these options in the CONFIG member would result in every individual Altair SLC user having their own independent copy of the font cache, which could lead to synchronisation problems if the *<wpspfx>.FONTS* library is changed.

Further reading

The following guides are available as reference material for Altair SLC:

- Migration Guide for z/OS
- Reference for Language Elements
- Communicate Guide
- What's New document for the current version.

Migration Guide for z/OS

This guide will help you with the process of moving to Altair SLC on the z/OS platform. It has sections on how to migrate programs and data and includes information specifically to help with migrating an MXG environment.

The migration guide is supplied as part of the distribution package in PDF format. All documents are also available on the World Programming website: *https://www.worldprogramming.com*

Appendix A – Running Altair SLC from UNIX Systems Services

Altair SLC can be run from a UNIX Systems Services (USS) session. It is very likely that the USS components of Altair SLC have been installed to support the use of TrueType fonts. For more information see Extracting the USS components.

Launching Altair SLC

A SAS language program in a file called *<filename>* can be processed by Altair SLC running from USS using the command:

```
<WPSHOME>/bin/wps <filename>
```

Output files

Altair SLC uses the standard output (`stdout`) stream for its logging and the standard error (`stderr`) stream to show errors. By default, `stdout` and `stderr` write to the screen. If a file is required for the log and error output then it can be redirected using a command similar to:

```
 <WPSHOME>/bin/wps <filename> >log.txt 2>&1
```

Making it easier

It will make life easier for those users regularly executing Altair SLC on USS if their login scripts contain the following lines:

```
export WPSHOME=<WPSHOME>
alias runwps=$(WPSHOME)/bin/wps
```

This will enable Altair SLC to be run by issuing the command:

```
runwps <filename>
```

Installation verification

The @verify member of PDS `<wpspfx>.CNTL`, supplied with Altair SLC, contains a sample job that can be used to verify that the installation of Altair SLC has been successfully completed for z/OS.

The equivalent process should now be carried out to show that the USS installation of Altair SLC has been successful. Enter the following command to run the xverify source program:

```
runwps “//’<wpspfx>.CNTL(XVERIFY)’” > log.txt 2>&1
```

The `SASLOG` output will appear as `log.txt` and the `SASLIST` output as `xverify.lst`. These files should be studied to verify that the installation has been successfully completed.

Appendix B – Running Altair SLC from TSO

One function of the <wpfpfx>.DLIB(@iNSTALL) job that was used at the start of the installation procedure is to populate z/OS library <wpfpfx>.CLIST with a member named tsowps. This member is a CLIST that enables use of Altair SLC in real time under native TSO (Time Sharing Option) or under TSO/ISPF. The only difference between the two environments is that the final output is presented in slightly different ways.

The clist features a large number of arguments which may either be left set at their default values, or set to comply with site-specific standards, prior to making the facility available to end users. The list of defined arguments is shown below.

The main reason for the length of this list is the number of file allocations that need to be made to enable Altair SLC to run.

Arguments and default settings for the TSOWPS CLIST

- WPSPFX: This string must contain the z/OS dataset prefix for the Altair SLC installation that is to be used. Either set this on the invocation, or ensure that the CLIST specifies the appropriate default for the installation. This prefix is used to locate all the z/OS datasets within the installation. This parameter must have a value, either explicit or an implicit default, in order for the CLIST to be able to initiate the Altair SLC invocation.
- USERPFX: This string defaults to the user's high-level z/OS dataset name qualifier, and will be used when qualifying unquoted z/OS dataset arguments. This is similar to the way that ISPF (Interactive System Productivity Facility) employs the user prefix for unquoted z/OS dataset names. Pass in an alternative hlq string to be used as the initial optional qualification name.
- OUTDSNPFX: An optional parameter that, if specified, will be used to form the default stem name of the names of all output z/OS datasets that are not otherwise specified. If the outdsnpfx name is not a quoted string, then it will be qualified with the userpfx value. If this parameter is not specified, then the default stem name will be formed from the userpfx and dsqual values.
- DSQUAL: An optional parameter that defaults to .wps and is used when outdsnpfx is not specified. It is combined with userpfx to form the default stem name for all output z/OS datasets that are not specified elsewhere.
- OPTIONS: An optional parameter that can be used to pass option values into the invocation of Altair SLC.
- SYSPARM: An optional parameter that can be used to pass any sysparm bindings into the invocation of Altair SLC.
- CONFIG: An optional parameter that can be used to specify a single configuration z/OS dataset name or z/OS dataset member that will be passed to Altair SLC before the installation-based configuration member. If an unquoted name is used, then it will be prefix-qualified with the userpfx value.

- SASAUTOS: An optional parameter that can be used to specify a single z/OS dataset name or z/OS dataset member that will be passed to Altair SLC as the source of AUTOCALL macros ahead of the installation-based source. If an unquoted name is used, then it will be prefix-qualified with the userpfx value.
- SASHELP: An optional parameter that can be used to specify a single z/OS dataset name or z/OS dataset member that will be passed to Altair SLC as the source of the location of the SASHELP library. If an unquoted name is used, then it will be prefix-qualified with the userpfx value.
- WPSFONTS: An optional parameter that can be used to specify a single z/OS dataset name that will be passed to Altair SLC as the source of font data ahead of the installation-based source. If an unquoted name is used, then it will be prefix-qualified with the userpfx value.
- WPSLOAD: An optional parameter that can be used to specify a single z/OS dataset name that will be passed to Altair SLC as the load library ahead of the installation based load library. If an unquoted name is used, then it will be prefix-qualified with the userpfx value.
- SASLIST: An optional parameter that can be used to specify the z/OS dataset name to which the saslist output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to dummy. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist, it will be allocated using the allocation parameters from the saslistsize parameter.
- SASLISTSIZE: An optional parameter that specifies the default size for the saslist output z/OS dataset.
- SASLOG: An optional parameter that can be used to specify the z/OS dataset name to which the saslog output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to dummy. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist, it will be allocated using the allocation parameters from the saslogsize parameter.
- SASLOGSIZE: An optional parameter that specifies the default size for the saslog output z/OS dataset.
- WPSTRACE: An optional parameter that can be used to specify the z/OS dataset name to which the wpstrace output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to dummy. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the wpstracesize parameter.
- WPSTRACESIZE: An optional parameter that specifies the default size for the wpstrace output z/ OS dataset.

- `CEEDUMP`: An optional parameter that can be used to specify the z/OS dataset name to which the ceedump output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to dummy. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the ceedumpsize parameter.
- `CEEDUMPSIZE`: An optional parameter that specifies the default size for the ceedump output z/OS dataset.
- `CEERPT`: An optional parameter that can be used to specify the z/OS dataset name to which the ceerpt output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to dummy. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the USERPFX value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the ceerptsize parameter.
- `CEERPTSIZE`: An optional parameter that specifies the default size for the ceerpt output z/OS dataset.
- `SORTMSGS`: An optional parameter that can be used to specify the z/OS dataset name to which the `SORTMSGS` output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to DUMMY. The special value "*" can be used to cause the output tobe directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix- qualified with the USERPFX value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the sortmsgssize parameter.
- `SORTMSGSSIZE`: An optional parameter that specifies the default size for the sortmsgs output z/ OS dataset.
- `SYSPRINT`: An optional parameter that can be used to specify the z/OS dataset name to which the SYSPRINT output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to DUMMY. The special value "*" can be used to cause the output tobe directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix- qualified with the USERPFX value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the sysprintsize parameter.
- `SYSPRINTSIZE`: An optional parameter that specifies the default size for the sysprint output z/OS dataset.
- `SYSOUT`: An optional parameter that can be used to specify the z/OS dataset name to which the sysout output will be written. The special value *DUMMY* can be used to cause the output to be ignored by binding it to DUMMY. The special value "*" can be used to cause the output to be directed to the terminal session. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing output z/OS dataset will be reused and overwritten. If the z/OS dataset name does not exist it will be allocated using the allocation parameters from the sysouttsize parameter.

- SYSOUTSIZE: An optional parameter that specifies the default size for the sysout output z/OS dataset.
- WORK: An optional parameter that can be used to specify the work z/OS dataset name. If an unquoted z/OS dataset name is used, it will be prefix-qualified with the userpfx value. Any existing z/OS dataset will be reused. If the z/OS dataset name does not exist, it will be allocated using the allocation parameters from the worksize and workap parameters.
- WORKSIZE: An optional parameter that specifies the default size for the work z/OS dataset.
- WORKAP: An optional parameter that specifies other allocation parameters that are passed to the allocation command when the work z/OS dataset is allocated.
- SYSIN: The name of the SAS language programme source, this parameter will be qualified with the userpfx if it is not a quoted z/OS dataset name. If the parameter is not specified on the command line, then the CLIST will prompt the user to supply a suitable name.
- STAMP: When this optional switch parameter is specified, the CLIST will qualify all of the output z/OS dataset names with a date and time of the form Yyyyy.Dnnn.Thhmmss, where yyyy is the four digit year, nnn is the three digit day, and hhmmss represent the current hours, minutes and seconds respectively. The default is to not qualify the output z/OS dataset names.
- NOBROWSE: When this optional switch parameter is specified, the CLIST will behave, when run in TSO/ISPF, as if it was running outside of ISPF, and will not use the ISPF browse and view services to display the saslog and other z/OS datasets output by Altair SLC.
- DDVERBOSE: When this optional switch parameter is specified, the CLIST will display more detailed output prior to invoking Altair SLC. This additional output shows the allocated ddnames with which Altair will be invoked.

Before using the CLIST

It is advisable to modify the CLIST to provide a default <WPSPFX> argument value. This change is aimed at facilitating use of the CLIST. Subsequently, end users will not need to know the value of the argument, and will not need to specify it, other than under special circumstances.

Before modification, the first few lines of the CLIST are similar to:

```
PROC 0 +
WPSPFX() /* PREFIX FOR THE Altair SLC INSTALLATION */ +
USERPFX() /* PREFIX FOR USER DATASET QUALIFICATION */ +
```

After modification, the code will be similar to:

```
PROC 0                                              +
WPSPFX('''WPS.V400.B05993''')                       +
USERPFX() /* PREFIX FOR USER DATASET QUALIFICATION */ +
```

The number of quotation marks surrounding the *WPSPFX* parameter value is forced by CLIST syntax requirements alongside the setting of the PREFIX value in your TSO PROFILE. Take care and ensure the correct number are specified.

Test the CLIST

Following modification, the CLIST should be tested. Firstly, make it available by either:

- Copying the CLIST in to a library on your own SYSEXEC concatenation, or
- Modifying your SYSEXEC concatenation to feature <WPSPFX>.CLIST

To test the CLIST use the XVERIFY script. The required command will be similar to:

```
tsowps sysin(‘‘‘<wpspfx>.CNTL(XVERIFY)’’’)
```

Following successful testing, a copy of the CLIST may be moved to a system-wide CLIST or SYSEXEC concatenation.

Launching Altair SLC

Altair SLC requires a large amount of memory to load and run. For this reason, prospective TSO users will probably need to have their default memory size parameter value changed. This value is normally on the initial TSO logon panel. It represents the KB of above-the-16MB-line storage requested when logging on. We recommend a minimum value of 150000.

You can run Altair SLC from TSO or TSO/ISPF using the following command:

```
tsowps sysin(‘‘‘<wpspfx>.CNTL(XVERIFY)’’’)
```

The command may be entered using:

- the command line of any ISPF panel, by prefixing the command with 'tso', for example:

  ```
  tso tsowps sysin(‘‘‘<wpspfx>.CNTL(XVERIFY)’’’)
  ```

- the command line provided in the ISPF Command Shell (option 6 from the ISPF Primary Option Menu); in this case, the prefix tso is optional;
- native TSO, at the 'READY' prompt. In this case the command must NOT be prefixed with tso.

Once program execution is complete, the resulting files are presented in different ways, depending on whether native TSO or TSO/ISPF was used for the task:

- If TSO or TSO/ISPF was used, you will see the contents of the resulting SASLOG file in 'VIEW' mode. From here, use of the 'END' (PF3) command will result in a full list of all the SYSOUT-type files that have been generated.
- If native TSO was used, you will see alist of generated files. There is no immediate 'VIEW' of the SASLOG.

Installation verification

The <WPSPFX>.CNTL (XVERIFY) file can be used for this process. Use the command:

```
tsowps sysin(‘‘‘<wpspfx>.CNTL(XVERIFY)’’’)
```

and then check the output for successful completion.

The contents of the CLIST may change over time. You should therefore not rely on a version of the CLIST from a previous version of Altair SLC. If basic invocation errors are experienced after an upgrade of Altair SLC, one of the earliest checks to be carried out is to test the currency of the TSOWPS CLIST used.

Appendix C – Sending files to Altair

To aid in the diagnosis and resolution of your support issues, you may need to package data files and send them to Altair. There are numerous ways to do this, depending on the nature of the data, and this section describes some of the more common methods.

The data you might send would typically be:

- SYSOUT output generated by using Altair SLC.
- Altair SLC data libraries.
- Non Altair SLC data files (probably used as input to a script, for example an SMF data file)

Retrieve SYSOUT data from the JES2 spool

If the SYSOUT data generated by a SAS language program is to be sent, you should send the *entire* SYSOUT data, not just parts that are thought to be relevant. This circumvents the repetition of requests for further parts of the SYSOUT data.

In a number of cases there may be a significant amount of output to be handled; for instance, we may ask for a re-run of a problem program with the following SAS language statement specified:

```
OPTIONS SOURCE SOURCE2 MACROGEN SYMBOLGEN MLOGIC;
```

Running a standard MXG BUILDPDB with these options will typically result in over 1.4 million lines of output.

To retrieve output for a specific job from the JES spool, locate the output on the hold queue (or it may be the output queue), then issue the xdc line command against it. Use the command against the entire SYSOUT entry on the panel, not the individual output sections. The result is a panel asking for details of the output file to be used. On this panel, specify the attributes of the file to be created. The values that may be shown on the panel will be the file attributes used the the previous xdc command.

For the DCB attributes, we strongly recommend specifying the RECFM=VBA and BLKSIZE=27998, along with an LRECL value that suits the length of the longest output line. A value of 300 is usually sufficient. The amount of space to be allocated for the output file depends on the number of lines to be processed. The BUILDPDB example above required a file of nearly 2000 tracks to store.

Selecting SMF data for transmission

On rare occasions, we may ask for sample SMF data to be sent to aid problem diagnosis. The program to use for extracting such a sample is either of the IBM-supplied utility programs IFASMFDP or IFASMFDL, depending on whether the SMF data is collected in so-called 'MANx' files or in Coupling Facility file(s). Usage of both programs is described in detail in the IBM manual *IBM System Management Facilities manual*, catalogue number SA22-7630.

Compressing data using AMATERSE

Once the evidence of a problem is collected, we recommend the use of the IBM-supplied `TRSMAIN`/`AMATERSE` program to compress the file(s) before sending to Altair.

`TRSMAIN` is the original name for the current `AMATERSE` utility program. `AMATERSE` is provided with an alias entry point of `TRSMAIN`, to maintain backward compatibility. If must be remembered that *DDNAMES* for the input and output files are different depending on which program is named in the `EXEC PGM = <programname>` statement.

`AMATERSE` is now shipped as part of the z/OS Basic Control Program, so it should be available on all z/OS systems. Whether the `AMATERSE` utiity is enabled for a particular user is a matter for local system administration.

The `AMATERSE` utility and how to use it are fully described in the IBM manual *MVS Diagnosis: Tools and Service Aids*, catalogue number GA32-0905, specifically, chapter 18.

In addition to reducing the amount of data to be transmitted, the compressed file format has a predefined 1024-byte record size and the data in the file is always binary in nature. Input file attributes such as record length, organisation, and so on are held in the compressed file, so do not have to be considered when handling the transmission of the data.

Having created a compressed-data file, it will eventually need to be sent to WPL by some method, probably FTP. Since the data is in compressed format, the *only* transmission type that may be used is BINARY.

Appendix D – Load Modules and LPA Usage

During Altair SLC installation, the *<wpspfx>.LOAD* PDSE library is created, which contains the program load modules.

The names of these modules follow one of two naming conventions:

- names are formed using the four-letter *wpsx* prefix and a unique suffix, or
- names are formed using the three-letter *wps* prefix and a unique suffix.

Altair SLC uses a number of common core modules that will generally be loaded at the time of task initiation. These are the modules are named using the *wpsx* prefix.

All other modules are loaded on demand as part of the Altair SLC plug-in architecture to provide implementation for the functionality required for SQL language procedures, and functions and CALL routines.

When installing Altair SLC at some mainframe sites, the administrator may decide to place some of the load modules into the LPA (Link Pack Area). As all of the modules with the *wpsx* prefix are required when Altair SLC is initiated, these should be considered as initial candidates for placing into the LPA. Other load modules should be placed into the LPA, based on evidence of their usage during routine system workload.

Legal Notices

Trademarks

Altair SLC™, Altair Analytics Workbench™, and Altair SLC Hub™ are registered trademarks or trademarks of Altair Engineering, inc. (r) or ® indicates a Community trademark.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

All other trademarks are the property of their respective owner.

General Notices

Altair Engineering, inc. is not associated in any way with the SAS Institute.

Altair SLC is not the SAS System.

The phrases "SAS", "SAS language", and "language of SAS" used in this document are used to refer to the computer programming language often referred to in any of these ways.

The phrases "program", "SAS program", and "SAS language program" used in this document are used to refer to programs written in the SAS language. These may also be referred to as "scripts", "SAS scripts", or "SAS language scripts".

The phrases "IML", "IML language", "IML syntax", "Interactive Matrix Language", and "language of IML" used in this document are used to refer to the computer programming language often referred to in any of these ways.

Altair SLC includes software developed by third parties. More information can be found in the THANKS or acknowledgments.txt file included in the Altair SLC installation.