Getting started with IBM TPF Toolkit

Objectives

This tutorial provides a sample TPF 4.1 application and will guide you through the steps required to bring this application into the TPF Toolkit V3.4.0 environment so that you can build, maintain, and debug it using TPF Toolkit.
Note: This tutorial is intended as a general guide to getting started using TPF Toolkit V3.4.0. The specific characteristics of your current environment might require set up and steps that are not mentioned in this tutorial. Contact your IBM® Representative if you require extra assistance.

This tutorial only covers moving a TPF 4.1 application into TPF Toolkit V3.4.0. The result will be a TPF 4.1 application that can you can maintain, build, and debug using TPF Toolkit V3.4.0. Migrating TPF 4.1 applications to the new maketpf facility or to run on z/TPF is beyond the scope of this tutorial.

Time required

This tutorial should take approximately 60 minutes to complete.

Before you begin

This tutorial assumes the following:

- Some knowledge of Windows® XP or Vista Business or Enterprise, and how to run applications on these platforms.
- Some knowledge of installing, setting up remote connections, and moving between perspectives and views in the workbench.
- TPF Toolkit has been installed on your machine and a connection to the remote host (on which you complete your development) has been established.
- The source files for the sample application reside on the z/OS® UNIX® System Services file system.

Tip: Refer to the Appendix at the end of this tutorial for details on how to move from a traditional dataset-based environment to z/OS UNIX.
Note: All of the source files for the sample TPF application reside in the **extras\NewUserTutorial\** folder. This folder is located in C:\Program Files\IBM\TPF Toolkit V34 and contains the following source files:

- tpfapp.cpp
- stringMethods.cpp
- stringMethods.hpp
- c$zstr.hpp
- zstral.asm
- zstr.mac

You can build this source on the z/OS UNIX environment using cxx commands to build the C++ source and c89 commands to build Assembler source. Your z/OS UNIX environment also has several environment variables that are created and set when you type the omvs command to enter the z/OS UNIX environment. The values of these variables (along with the cxx and c89 commands) are required to build your source.

The cxx and c89 command strings also contain parameter values or options that instruct the compiler to build the source with specific characteristics. For example, the command string might contain options to indicate if debug information should be included in the resulting modules.

The TPF DLL and DLM are built with CBLD build scripts to generate the JCL or by coding the JCL directly. The JCL is submitted as a job on the MVS™ system. For the purposes of this tutorial, the build scripts reside on z/OS UNIX in the same folder as the TPF application source code, in files with .bsc and .jcl extensions. For the sample TPF application, these files are:

- ztad.bsc
- ztad.jcl
- ztam.bsc
- ztam.jcl

You can build the Loadset by coding the appropriate JCL statements and input control cards to create the Loadset and submit the job. The JCL source is stored in the **tpfapp.jcl** file.
Note: All of these files reside in the **extras\NewUserTutorial\** folder. This folder is located in C:\Program Files\IBM\TPF Toolkit V34.

Description

The sample TPF application that you will move is a small application that outputs an EBCDIC string on the console when it is run on TPF. This application contains:

- Two C/C++ modules (tpfapp.cpp and stringMethods.cpp).
- One Assembler module (ZSTR).

These modules are built into a TPF DLM and a TPF DLL, and then packaged into a Loadset.

The application consists of the following source files:

- **tpfapp.cpp** - The tpfapp C++ module that acts as the mainline program for the TPF application. This module calls the **getString** and **cleanString** functions to retrieve, print, and exit the string.
- **stringMethods.cpp, stringMethods.hpp, c$zstr.hpp** - The stringMethods C++ module provides the **getString** and **cleanString** functions. These are wrapper functions that call the ZSTR module, and then extract the results of those calls from register R1.
- **zstral.asm, zstr.mac** - The ZSTR module is an assembler module that provides the **GETSTR** function. This function generates a string, places this string into register R1, and then into CLNSTR which zeros out Register R1. GETSTR is invoked by the getString function and CLNSTR is invoked by the cleanString function in the stringMethods C++ module.

When the application is built, TPF DLL ZTAD contains the stringMethods C++ module and a stub for the ZSTR assembler module. DLM ZTAM contains the tpfapp module and a definition side deck input definition for the ZTAD TPF DLL. The TPFAPP loadset contains the TPF DLL ZTAD, the DLM ZTAM, and the ZSTR assembler module. It is transported to the TPF test system using a general dataset (GDS).

Once the Loadset is loaded onto the TPF test system, complete the following steps to invoke the Loadset:

1. Activate the Loadset using the following command:
   ```
   ZOLDR ACT TPFAPP
   ```
2. Create a new user defined command to invoke the program by typing:
   ```
   ZFMSG ADD ZTAPP PROG-ZTAM
   ```
3. Run the program using the user-defined command by typing:
   ```
   ZTAPP
   ```

If the program runs successfully, output similar to this sample displays on the console:

```
String: E7E8E9BA2FE151B5F7D02DC1C2C3C4C5+
```

Overview

After you install TPF Toolkit and establish a connection to the remote host on which you complete your development, you need to configure several items before you can develop this TPF application using TPF Toolkit.
Tip: For more information on how to create remote connections to the remote hosts required for development, see the Related concepts section below.

1. Set up the Remote Action header file.
2. Set up the target environment.
3. Set up the build and link options.
4. Set up the load options.
5. Create the target environment.

6. Create a TPF project.
7. Create a TPF DLL build script.
8. Create a TPF DLM build script.
9. Create a TPF loadset generation script.
10. Build the application.

Part 1: Setting up the Remote Action header file

Remote actions are host z/OS UNIX commands or programs that are invoked from the TPF Toolkit environment. They are used extensively in the development of TPF applications. Each compile, assemble, module, and loadset generation is performed by a remote action.
These remote actions generate a script file that is transferred to the host z/OS UNIX subsystem and executed. You must set the environment information and variables in the Remote Action header file of the script before the action is run. The default header file (bbshtpf.bbs) resides in the TPFSHARE directory.
Note: By default, the TPFSHARE directory is located in C:\Program Files\IBM\TPF Toolkit V34\Config\.

For more information on how remote actions work in TPF Toolkit, see the Related concepts section below.

If your TPF Toolkit Administrator has not set up this file, you must edit the file and ensure that the environment variables are set to the same values that are contained in your current z/OS UNIX-based development environment.
Tip: To obtain the current settings for your environment, type the following command from a z/OS UNIX command line:

```
env
```

You must also set the STEPLIB environment variable to the values contained in the STEPLIB DD statement in the tpfapp.jcl file.

Part 2: Setting up the target environment

After the header file for remote actions is set up, you must define your target environment. A target environment in TPF Toolkit allows you to group properties that define a development or deployment environment in your organization. You can capture different environments by creating target environment definitions using the Target Environments preference pages. You can apply these target environments to your TPF projects to customize the edit, build, load, and debug settings for these projects.

Applications that are developed in the same environment and deployed on the same system have similar properties; for example, compiler, assembler, prelinker, linker and loader. Many applications have the same SYSLIB or OBJLIB. Target environments allow you to define these options in one place, making your applications easier to maintain.

To determine how to define your target environments, examine your applications. If the properties of your applications are not that similar, you might need to define a separate target environment for each application. Since you are only bringing one application into TPF Toolkit in this tutorial, you only need to look in one place for this information.

A target environment is defined as a set of options. You define your set of options first, and then include these options in your target environment. For more information on target environments, see the Related concepts section below.

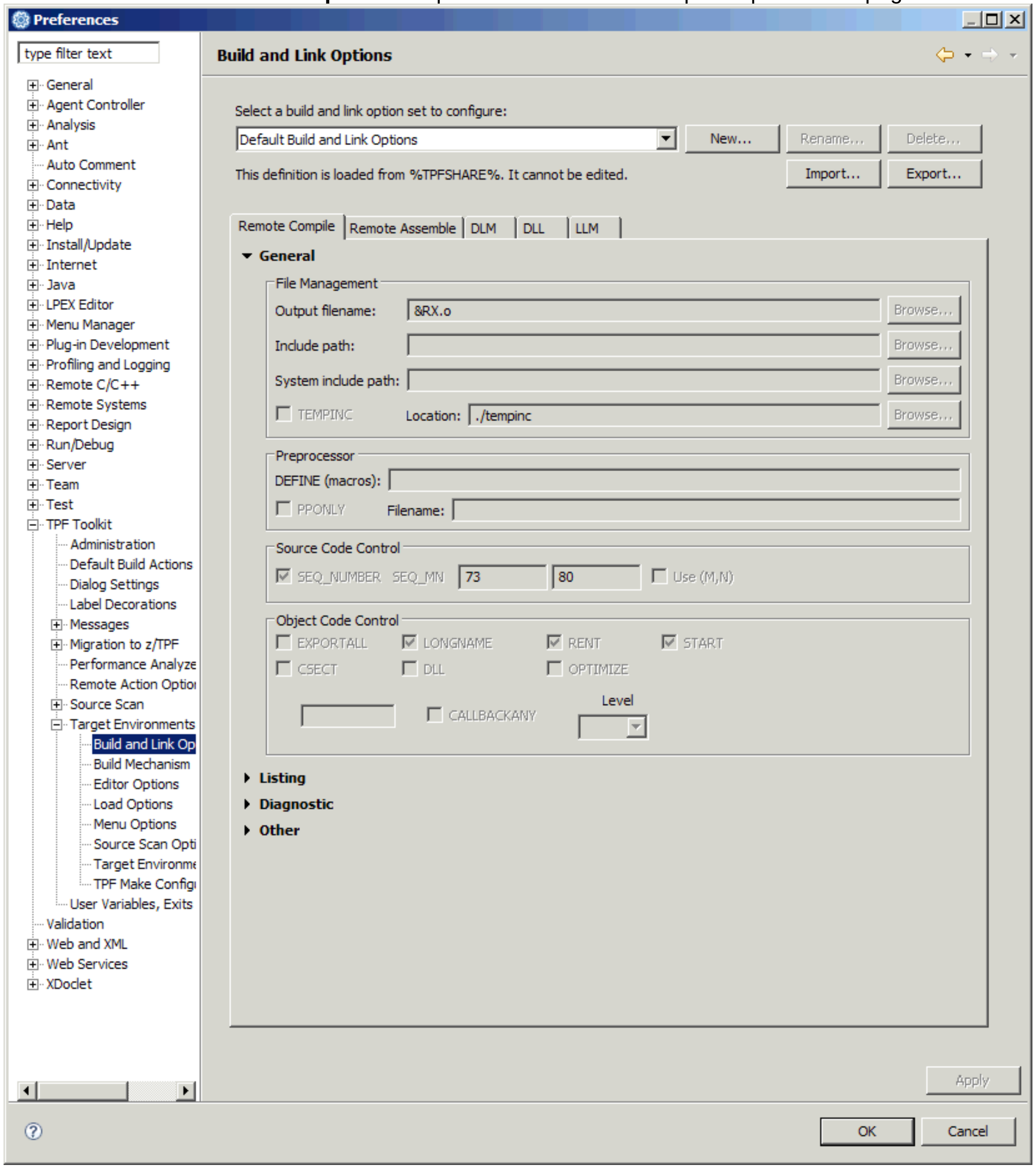
Part 3: Setting up the build and link options

Global compile, assemble, prelinker, and linker options are specified for C, C++, Assembler, TPF DLLs, and DLMs in the build and link options. When you submit a request to compile (that is, build) a source file, TPF Toolkit uses the values specified in the build and link options to compile the file. TPF Toolkit is shipped with default build and link options already set.

Note: Since you might want to debug this sample application using the TPF Toolkit debugger, the following example ensures that the correct build and link options are set to make tpfapp debuggable. These values are preset at the TPF Toolkit or workspace level, which means that all projects created in TPF Toolkit inherit these values. You can also specify default values for a specific TPF project or individual source files.

To create build and link options for tpfapp, complete the following steps:

1. In the workbench, select **Window** > **Preferences** to open the Preferences window.
2. In the left navigation pane, double-click the **TPF Toolkit** node to open the tree of available preference pages.
3. From the **TPF Toolkit** tree, double-click the **Target Environments** node to open the tree of available preference pages.
4. Click **Build and Link Options** to open the Build and Link Options preference page.



Build and Link Options

Select a build and link option set to configure: Default Build and Link Options

This definition is loaded from %TPFSHARE%. It cannot be edited.

General

File Management

Output filename: &RX.o

Include path:

System include path:

☐ TEMPINC Location: ./tempinc

Preprocessor

DEFINE (macros):

☐ PPONLY Filename:

Source Code Control

☑ SEQ_NUMBER SEQ_MN 73 80 ☐ Use (M,N)

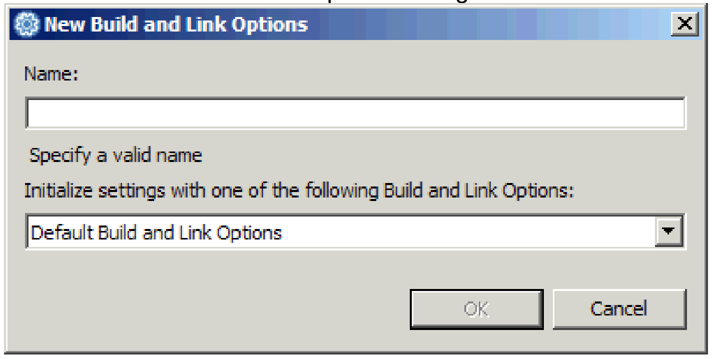
Object Code Control

☐ EXPORTALL ☑ LONGNAME ☑ RENT ☑ START

☐ CSECT ☐ DLL ☐ OPTIMIZE

☐ CALLBACKANY Level

5. Click **New** to open the New Build and Link Options dialog box.



New Build and Link Options

Name:

Specify a valid name

Initialize settings with one of the following Build and Link Options:

Default Build and Link Options

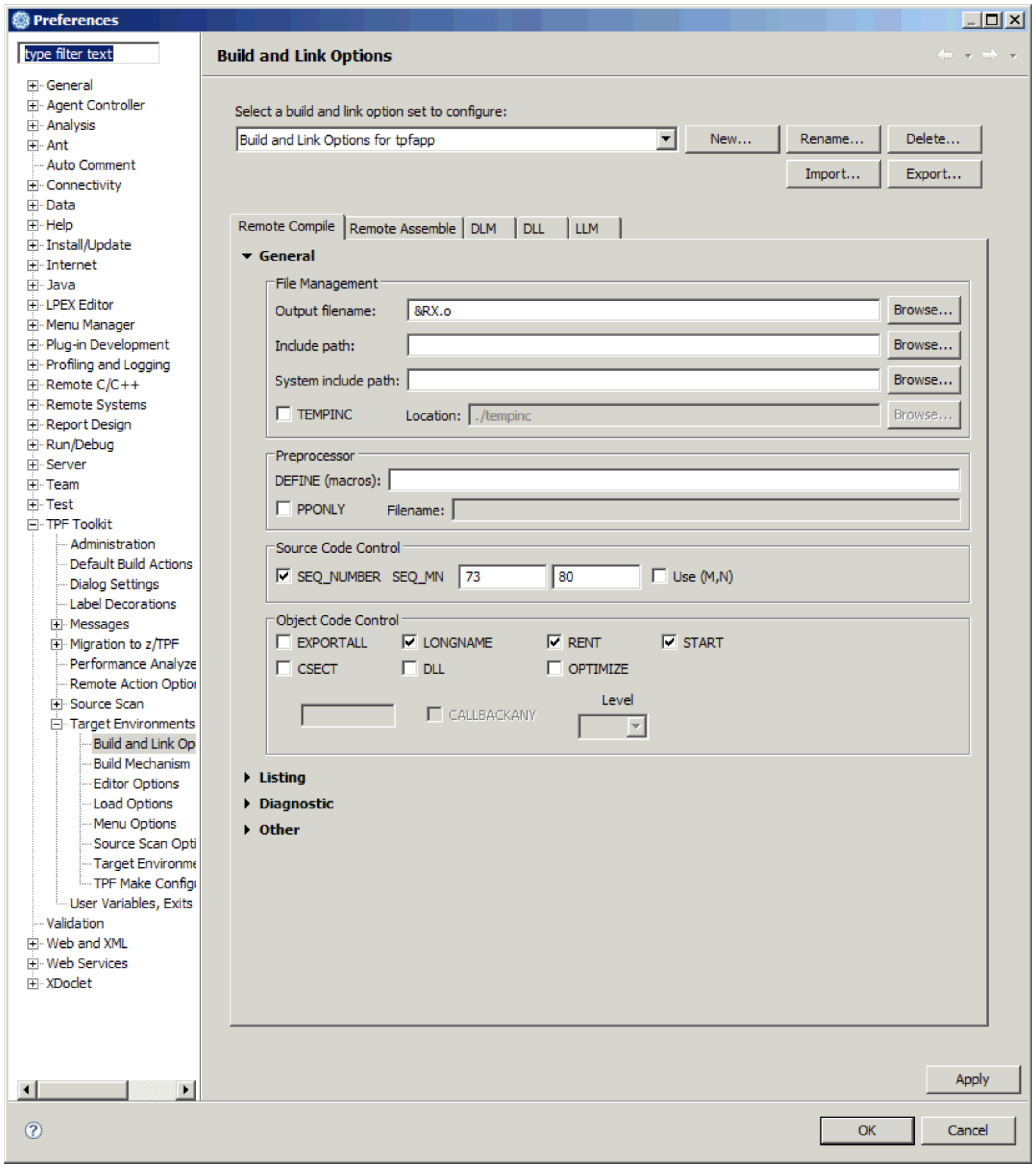
6. In the **Name** field, type **Build and Link Options for tpfapp**.
7. From the drop-down list, select **Default Build and Link Options with Debug Options** to use the initial settings from this set of options for the new set that you are creating.
8. Click **OK** to save your selections and close the dialog box. The name that you assigned to this new set of build and link options is displayed in the list of available sets of build and link options.

Now that you have created a set of build and link options (Build and Link Options for tpfapp), you can set the following build and link options:

- Remote compile
- Remote assemble
- DLM
- DLL

To set the remote compile options, complete the following steps:

1. In the Build and Link Options preference page, click the **Remote Compile** tab.



Build and Link Options

Select a build and link option set to configure: Build and Link Options for tpfapp

General

File Management
Output filename: &RX.o
Include path:
System include path:
TEMPINC Location: ./tempinc

Preprocessor
DEFINE (macros):
PPONLY Filename:

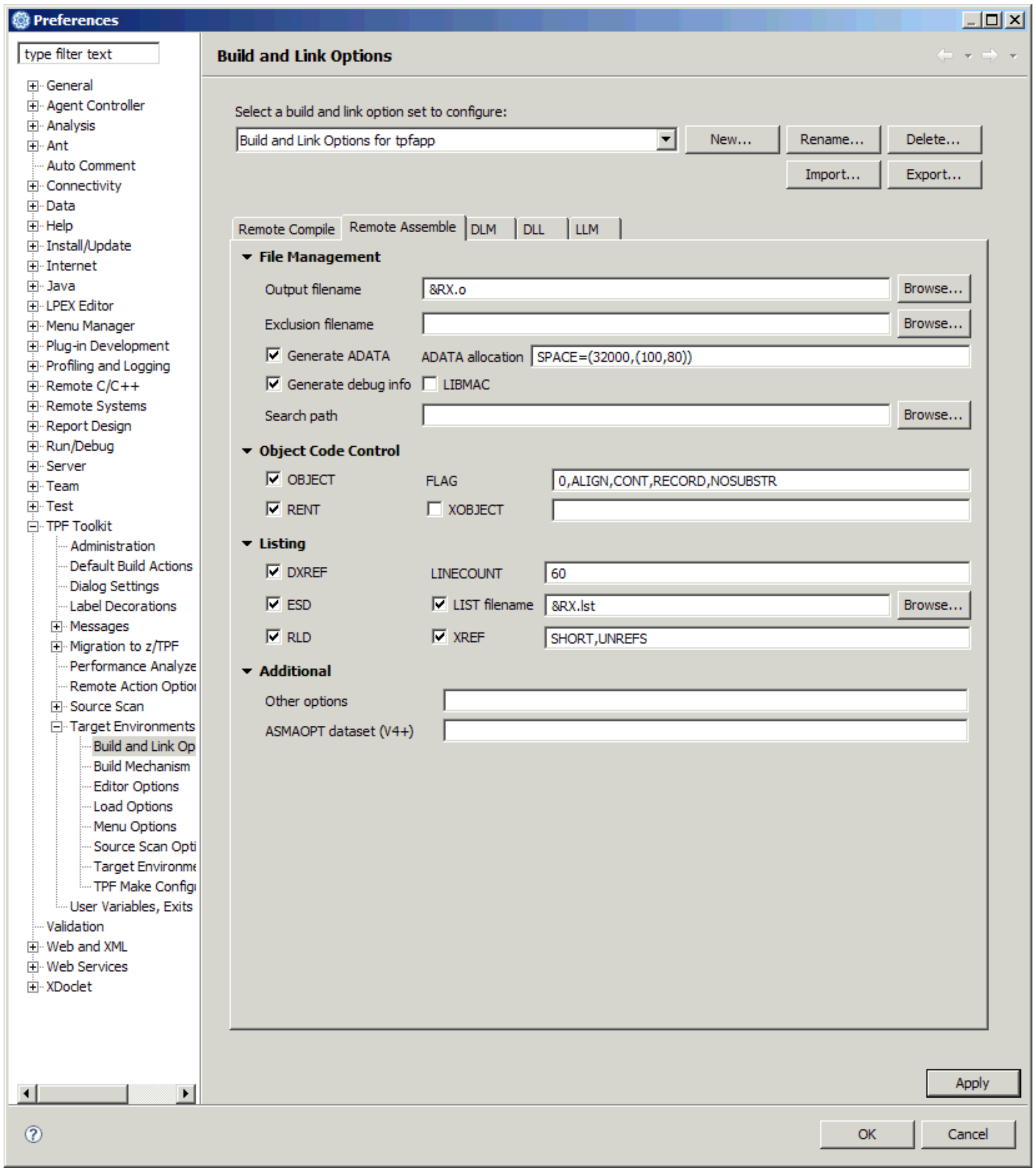
Source Code Control
[✓ SEQ_NUMBER] SEQ_MN 73 80 Use (M,N)

Object Code Control
EXPORTALL [✓ LONGNAME] [✓ RENT] [✓ START]
CSECT DLL OPTIMIZE
CALLBACKANY Level

2. Expand the **Diagnostic** section and select the following check boxes:
   - **TEST**
   - **HOOK**
   - **SYM**
   - **BLOCK**
   - **LINE**
   - **PATH**
3. Expand the **Other** section.
4. In the **Common options** field, type **-Wc,LANGLVL'(EXTENDED)'** to allow the **$** character in identifiers in the compile.
5. Click **Apply** to save your selections.

To set the remote assemble options, complete the following steps:

1. In the Build and Link Options preference page, click the **Remote Assemble** tab.



Build and Link Options

Select a build and link option set to configure: Build and Link Options for tpfapp

File Management

Output filename: &RX.o
Exclusion filename:
[✓] Generate ADATA   ADATA allocation: SPACE=(32000,(100,80))
[✓] Generate debug info   [ ] LIBMAC
Search path:

Object Code Control

[✓] OBJECT   FLAG: 0,ALIGN,CONT,RECORD,NOSUBSTR
[✓] RENT   [ ] XOBJECT

Listing

[✓] DXREF   LINECOUNT: 60
[✓] ESD   [✓] LIST filename: &RX.lst
[✓] RLD   [✓] XREF: SHORT,UNREFS

Additional

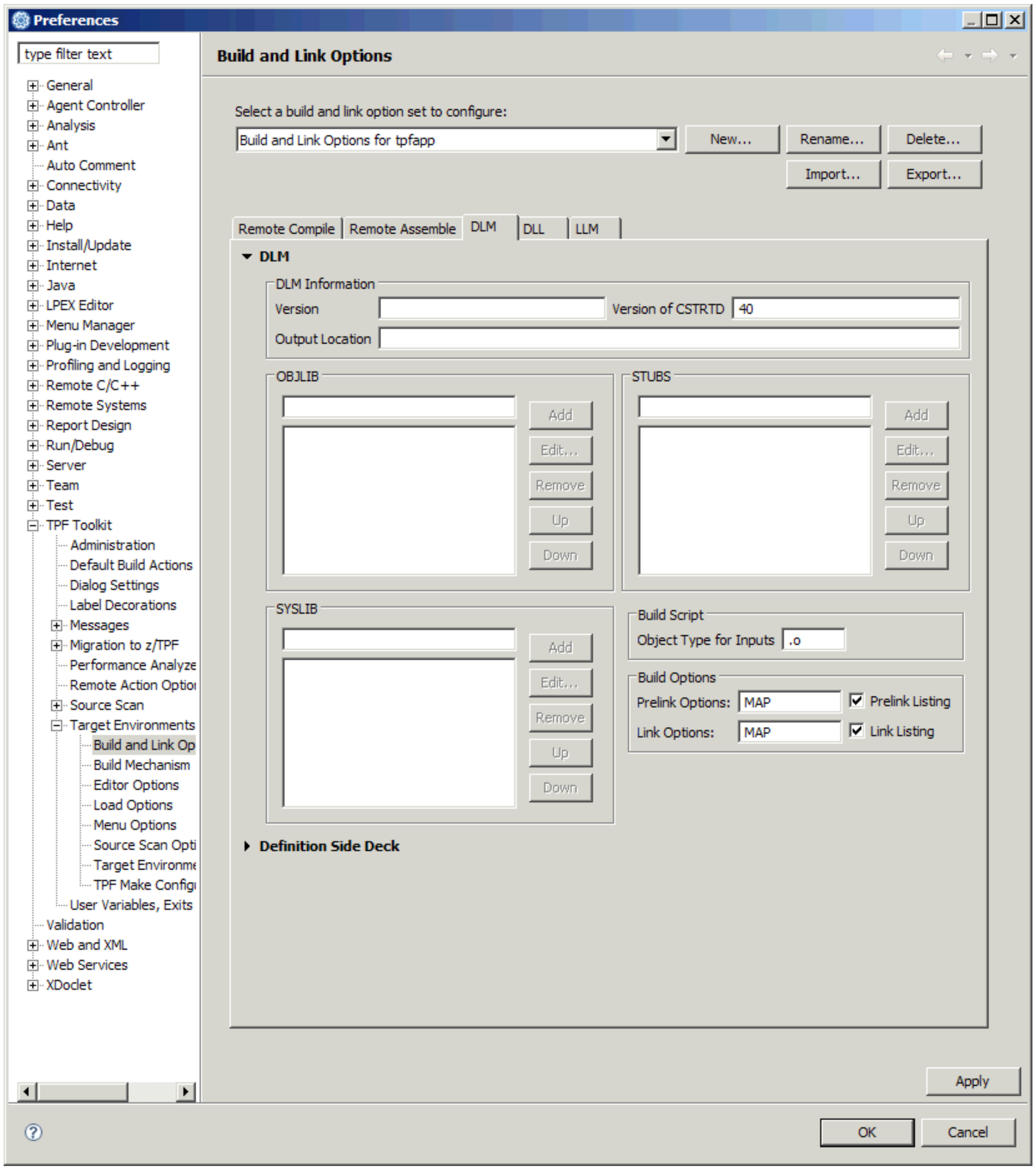
Other options:
ASMAOPT dataset (V4+):

2. Ensure that the **Generate ADATA** and **Generate Debug Info** check boxes are selected.
3. In the **Search Path** field, type **/u/myuser/tpfapp** to specify the location of zstr.mac.
4. Click **Apply** to save your selections.

Next, set the DLM options for the target environment. Typically, the DLM options are values that are global to your environment and rarely change. For example, OBJLIB and SYSLIB concatenations; this information comes from the JCL used to build the DLM. You should store program specific information in the corresponding build script file (.dlm, .dll, or .llm file).

To set the DLM options, complete the following steps:

1. In the Build and Link Options preference page, click the **DLM** tab.



Build and Link Options

Select a build and link option set to configure: Build and Link Options for tpfapp

DLM

DLM Information

Version:

Version of CSTRTD: 40

Output Location:

OBJLIB

STUBS

SYSLIB

Build Script

Object Type for Inputs: .o

Build Options

Prelink Options: MAP ☑ Prelink Listing

Link Options: MAP ☑ Link Listing

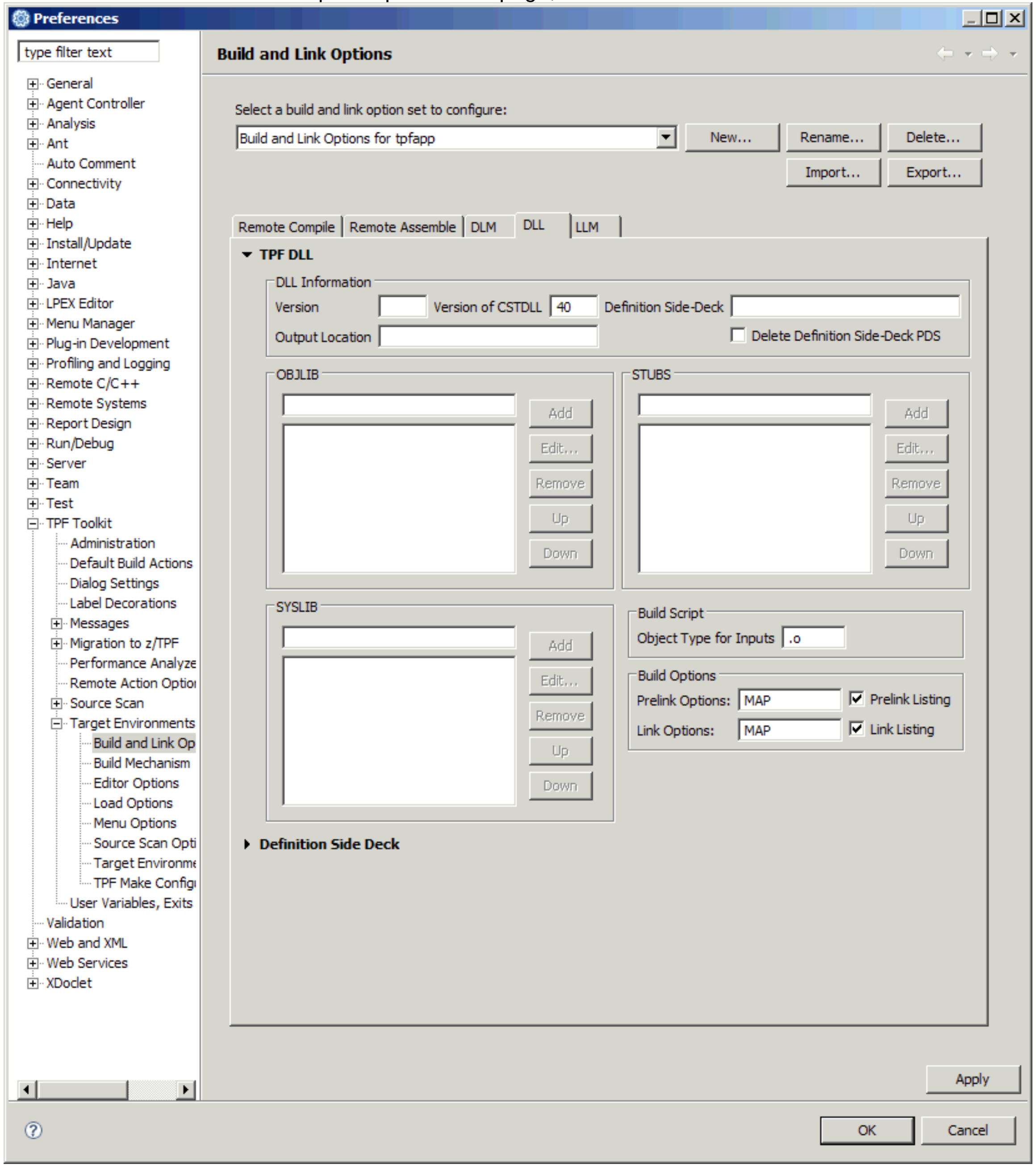
2. Expand the **DLM** section:
   - o In the **OBJLIB** section, add **ACP.OBJ.RLSEVP40.BSS**.
   - o In the **SYSLIB** section, add **ACP.CLIB.RLSEVP40.BSS** and **ACP.STUB.RLSEVP40.BSS**.
   - o In the **Prelink Options** field, type **MAP**.
   - o In the **Link Options** field, type **AMODE=31 RMODE=ANY LIST XREF MAP**.
3. Expand the **Definition Side Deck Concatenation** section. In the **Definition Side Deck Concatenation** section, add **ACP.IMPORTS.RLSEVP40.BSS**.
   Note: The values for each of these options are derived from the JCL that was used to create the ZTAM DLM:
   - o The OBJLIB value comes from the OBJLIB DD statement.
   - o The SYSLIB values come from the SYSLIB DD statement.
   - o The Definition Side Deck Concatenation value comes from the DSD DD statement.

Not all of the values in the JCL are used. The MYUSER.TPFAPP.LK1 dataset specified in the SYSLMOD DD statement and the MYUSER.TPFAPP.EXP1 dataset specified in the DSD DD statement are not specified since they do not apply to all applications using this target environment. These options will be specified later.

4. Click **Apply** to save your selections.

To set the DLL options, complete the following steps:

1. In the Build and Link Options preference page, click the **DLL** tab.



Select a build and link option set to configure: Build and Link Options for tpfapp

Version:
Version of CSTDLL: 40
Definition Side-Deck:
Output Location:
Delete Definition Side-Deck PDS: ☐

Object Type for Inputs: .o

Prelink Options: MAP ☑ Prelink Listing
Link Options: MAP ☑ Link Listing

2. Expand the **TPF DLL** section:
   - o In the **OBJLIB** section, add **ACP.OBJ.RLSEVP40.BSS**.
   - o In the **SYSLIB** section, add **ACP.CLIB.RLSEVP40.BSS** and **ACP.STUB.RLSEVP40.BSS**.
   - o In the **Prelink Options** field, type **MAP**.
   - o In the **Link Options** field, type **AMODE=31 RMODE=ANY LIST XREF MAP**.

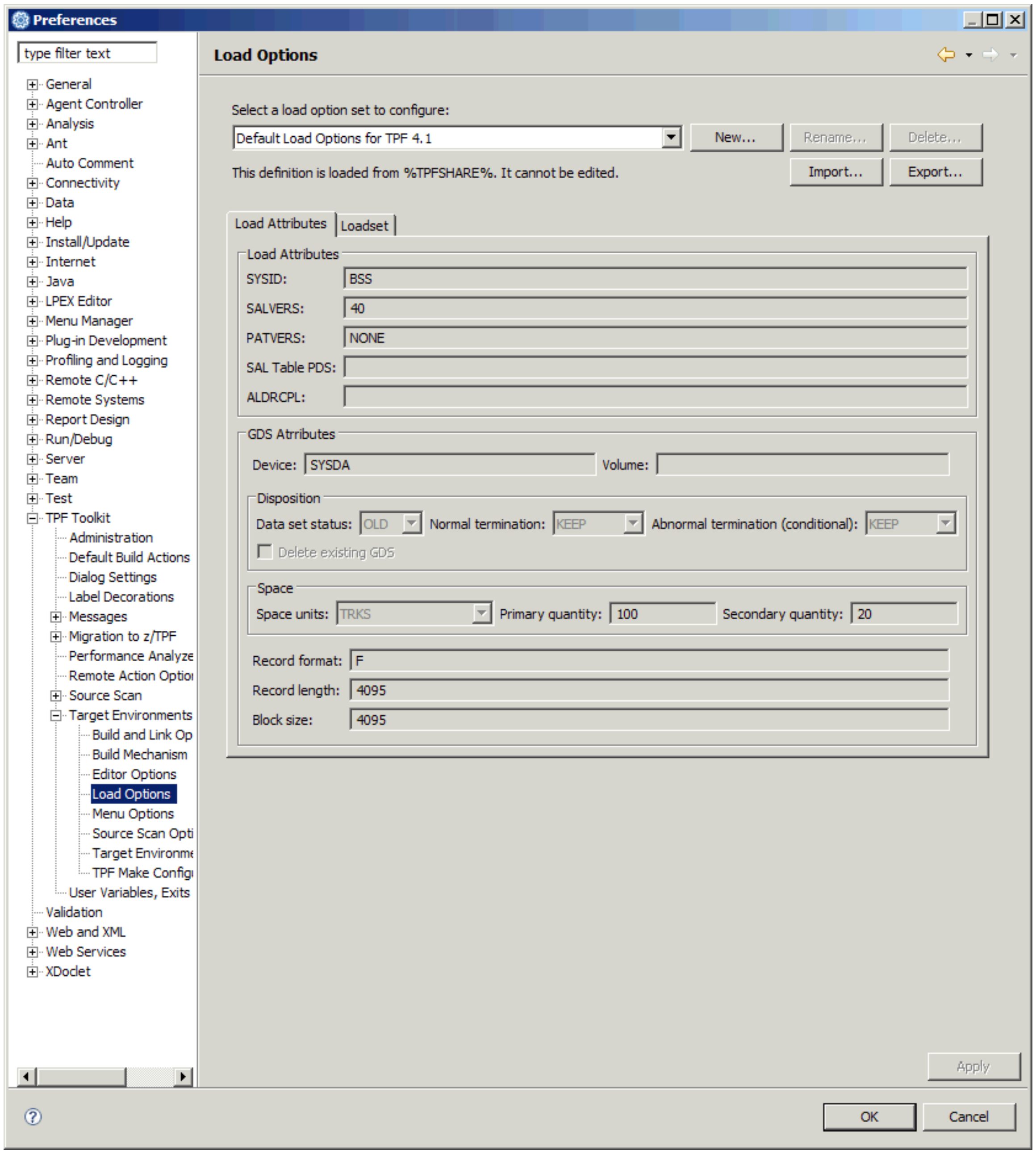
3. Expand the **Definition Side Deck Concatenation** section. In the **Definition Side Deck Concatenation** section, add **ACP.IMPORTS.RLSEVP40.BSS**.
   Note: The values for each of these options are derived from the JCL that was used to create the ZTAD DLL:
   - The OBJLIB value comes from the OBJLIB DD statement.
   - The SYSLIB values come from the SYSLIB DD statement.
   - The Definition Side Deck Concatenation value comes from the DSD DD statement.

   Not all of the values in the JCL are used. The MYUSER.TPFAPP.LK1 dataset specified in the SYSLMOD DD statement and the MYUSER.TPFAPP.EXP1 dataset specified in the SYSDEFSD DD statement are not specified since they do not apply to all TPF DLLs using this target environment. To capture default options that generally apply to all TPF DLLs for an application, create a set of build and link options.
4. Click **Apply** to save your selections.

Part 4: Setting up the load options

The load options are global options used in the generation and transfer of loadsets to your TPF test system. The values used for these options are derived from the loadset generation JCL script, **tpfapp.jcl**.
To create load options for tpfapp, complete the following steps:

1. In the left navigation pane of the Preferences window, double-click the **TPF Toolkit** node to open the tree of available preference pages.
2. From the TPF Toolkit tree, double-click the **Target Environments** node to open the tree of available preference pages.
3. Click **Load Options** to open the Load Options preference page.



Load Options

Select a load option set to configure: Default Load Options for TPF 4.1

This definition is loaded from %TPFSHARE%. It cannot be edited.

Load Attributes

SYSID: BSS

SALVERS: 40

PATVERS: NONE

SAL Table PDS:

ALDRCPL:

GDS Atrributes

Device: SYSDA Volume:

Disposition

Data set status: OLD Normal termination: KEEP Abnormal termination (conditional): KEEP

Delete existing GDS

Space

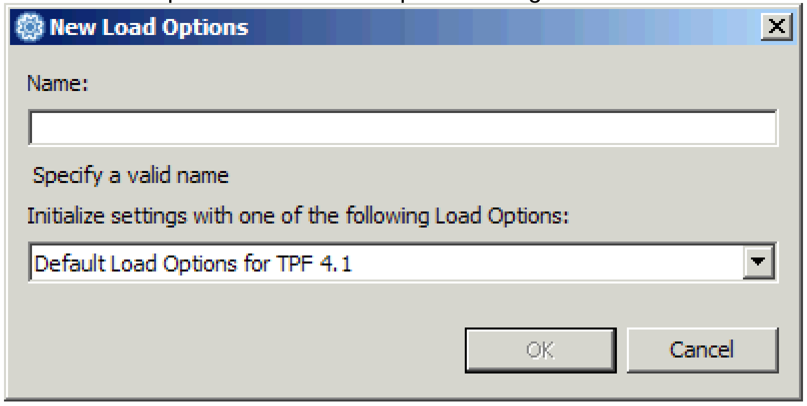
Space units: TRKS Primary quantity: 100 Secondary quantity: 20

Record format: F

Record length: 4095

Block size: 4095

4. Click **New** to open the New Load Options dialog box.



Name:

Specify a valid name

Initialize settings with one of the following Load Options:

Default Load Options for TPF 4.1

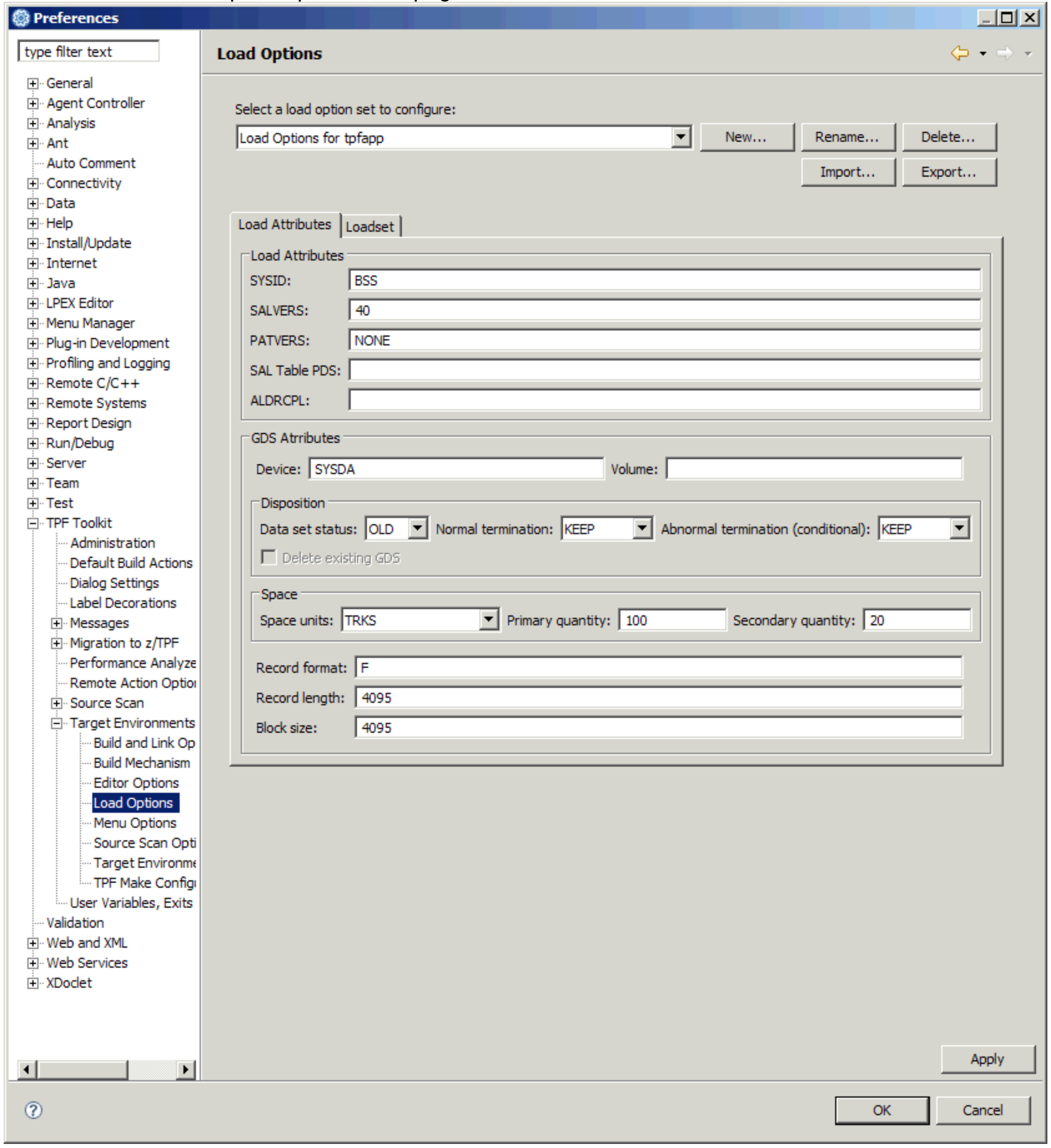
5. In the **Name** field, type **Load Options for tpfapp**.
6. From the drop-down list, select **Default Load Options for TPF 4.1** to use the initial settings from this set of options for the new set that you are creating.
7. Click **OK** to save your selections and close the dialog box. The name that you assigned to this new set of load options is displayed in the list of available sets of load options.

Now that you have created a set of load options (Load Options for tpfapp), you can specify the following load options:

- Load Attributes
- Loadset

To set the Load Attributes options, complete the following steps:

1. In the Load Options preference page, click the **Load Attributes** tab.



Select a load option set to configure: Load Options for tpfapp

Load Attributes
SYSID: BSS
SALVERS: 40
PATVERS: NONE
SAL Table PDS:
ALDRCPL:

GDS Atrributes
Device: SYSDA Volume:

Disposition
Data set status: OLD Normal termination: KEEP Abnormal termination (conditional): KEEP
Delete existing GDS

Space
Space units: TRKS Primary quantity: 100 Secondary quantity: 20

Record format: F
Record length: 4095
Block size: 4095

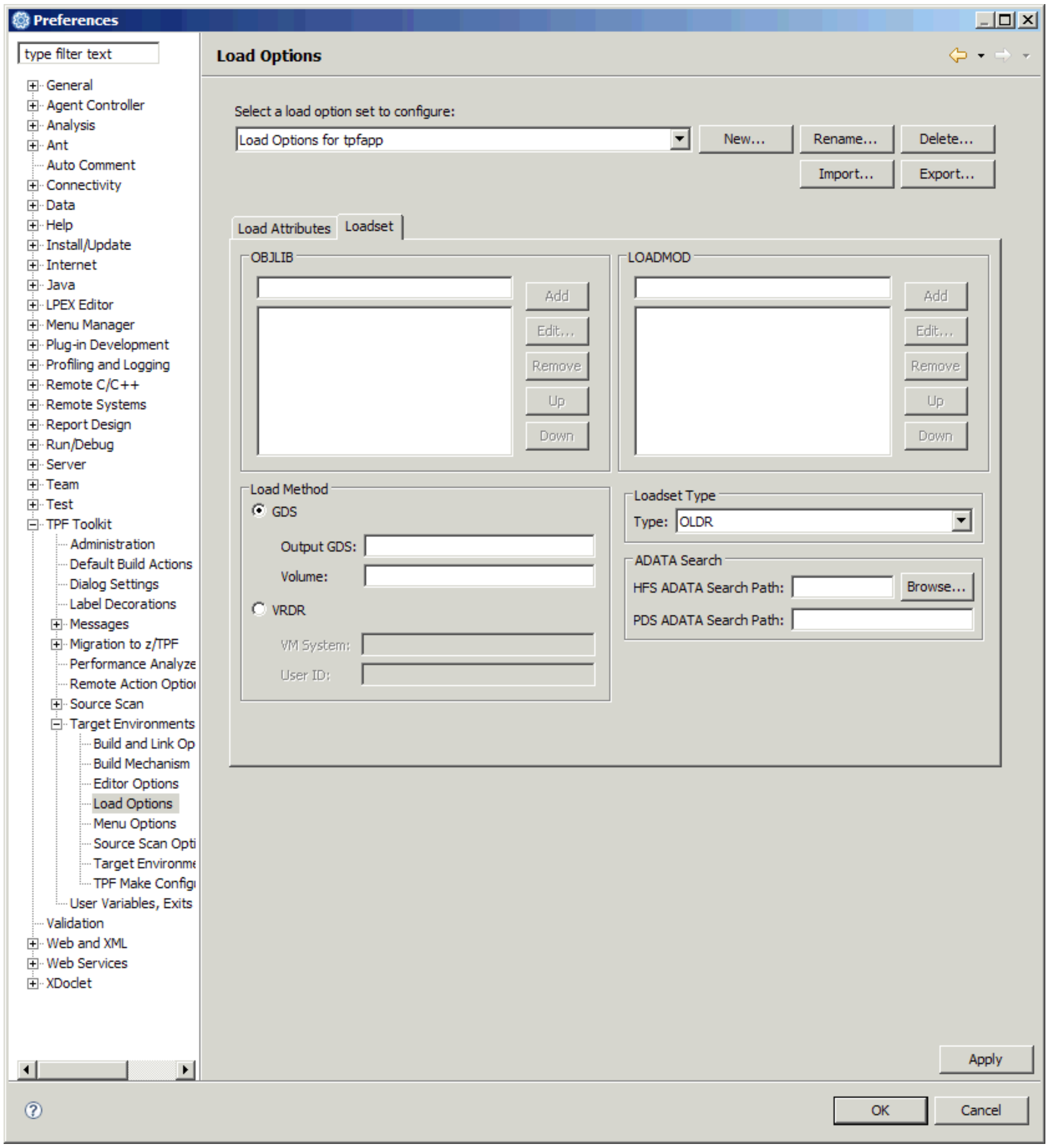
2. To specify the load attributes:
   - In the **SYSID** field, type **BSS**.
   - In the **SALVERS** field, type **40**.
   - In the **PATVERS** field, type **NONE**.

- In the **SAL Table PDS** field, type **ACP.SAL.RLSEVP40.BSS**.

3. To specify the GDS attributes, type **40** in the **Primary quantity** field.
   Note: The values for all other default options are equivalent to the following statements in tpfapp.jcl:
   - SYSID, SALVERS, and PATVERS come from the like-named control statements.
   - SAL Table PDS comes from the SALTB DD statement.
   - Device, Disposition, Space, Record format, Record length, and Block size come from the UNIT, DISP, SPACE, RECFM, LRECL, and BLKSIZE parameters specified in the OUTPUT DD statement.
4. Click **Apply** to save your selections.

To set the Loadset options, complete the following steps:

1. Click the **Loadset** tab.



Select a load option set to configure: Load Options for tpfapp

Load Method: [✓ GDS] VRDR

Output GDS:

Volume:

Type: OLDR

HFS ADATA Search Path:

PDS ADATA Search Path:

2. In the **OBJLIB** section, add **ACP.LINK.RLSEVP40.BSS**.
3. In the **Load Method** section, specify the GDS to which the loadset should be generated:
   a. In the **Output GDS** field, type **LDR.MYUSER**.
   b. In the **Volume** field, type **LDR**.

Note: The values for each of these options are derived from the following statements in tpfapp.jcl:

- OBJLIB comes from the OBJLIB DD statement.
- Output GDS and Volume come from the DSN and VOLUME parameters specified in the OUTPUT DD statement.

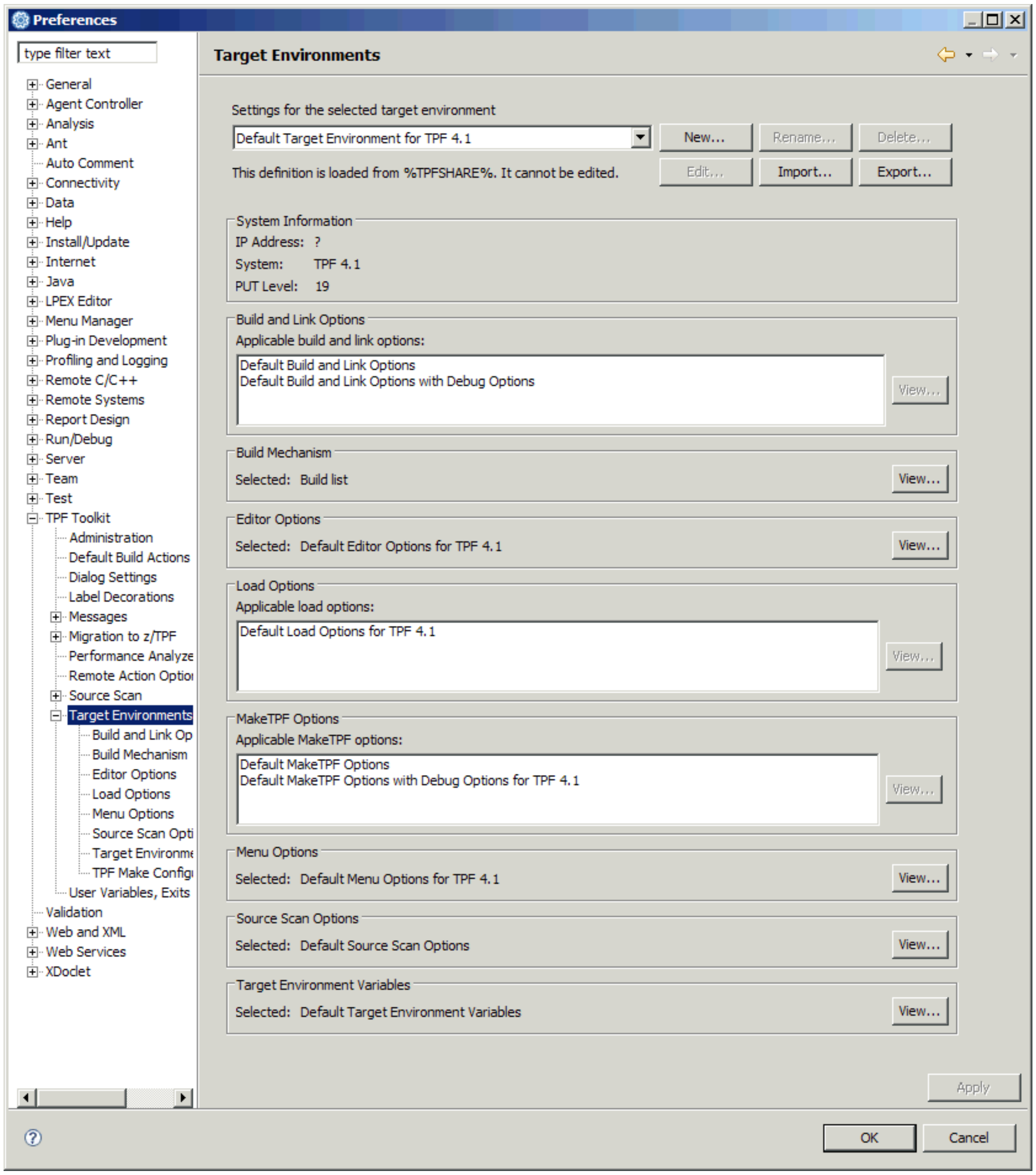
Much of the information in tpfapp.jcl is application specific and will be specified when you define the TPF project for the sample application.

4. Click **Apply** to save your selections.

You do not need to define any other options for your target environment. The default options shipped with TPF Toolkit are used. You can now create your target environment.

Part 5: Creating the target environment

1. In the left navigation pane of the Preferences window, double-click the **TPF Toolkit** node to open the tree of available preference pages.
2. From the TPF Toolkit tree, double-click the **Target Environments** node to open the tree of available preference pages.
3. Click **Target Environments** to open the Target Environments preference page.


Preferences

type filter text

General
Agent Controller
Analysis
Ant
Auto Comment
Connectivity
Data
Help
Install/Update
Internet
Java
LPEX Editor
Menu Manager
Plug-in Development
Profiling and Logging
Remote C/C++
Remote Systems
Report Design
Run/Debug
Server
Team
Test
TPF Toolkit
Administration
Default Build Actions
Dialog Settings
Label Decorations
Messages
Migration to z/TPF
Performance Analyze
Remote Action Optio
Source Scan
Target Environments
Build and Link Op
Build Mechanism
Editor Options
Load Options
Menu Options
Source Scan Opti
Target Environme
TPF Make Config
User Variables, Exits
Validation
Web and XML
Web Services
XDoclet

Target Environments

Settings for the selected target environment

Default Target Environment for TPF 4.1 — New... Rename... Delete...

This definition is loaded from %TPFSHARE%. It cannot be edited. — Edit... Import... Export...

System Information
IP Address: ?
System: TPF 4.1
PUT Level: 19

Build and Link Options
Applicable build and link options:
Default Build and Link Options
Default Build and Link Options with Debug Options

Build Mechanism
Selected: Build list

Editor Options
Selected: Default Editor Options for TPF 4.1

Load Options
Applicable load options:
Default Load Options for TPF 4.1

MakeTPF Options
Applicable MakeTPF options:
Default MakeTPF Options
Default MakeTPF Options with Debug Options for TPF 4.1

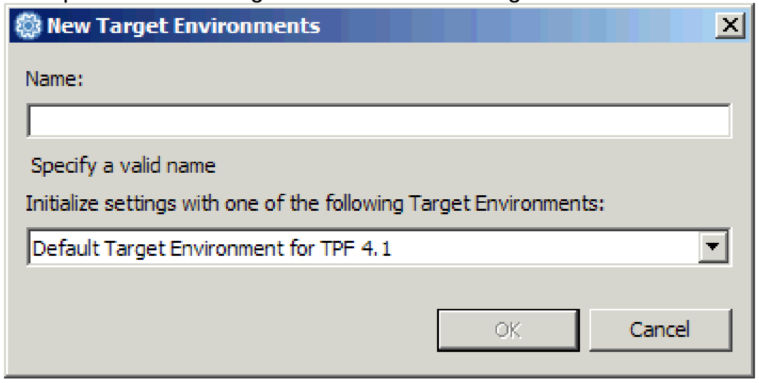
Menu Options
Selected: Default Menu Options for TPF 4.1

Source Scan Options
Selected: Default Source Scan Options

Target Environment Variables
Selected: Default Target Environment Variables

View... Apply OK Cancel

4. Click **New** to open the New Target Environments dialog box.


New Target Environments

Name:

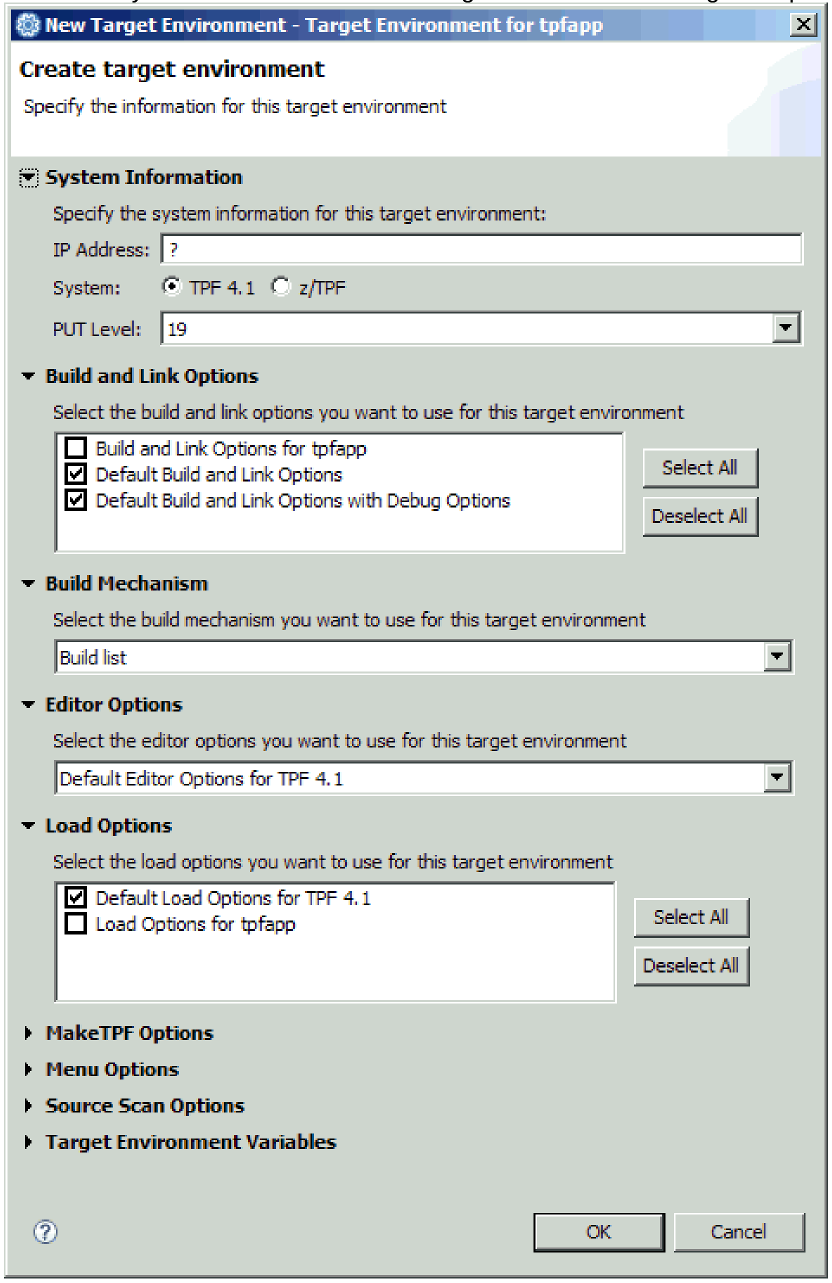
Specify a valid name

Initialize settings with one of the following Target Environments:

Default Target Environment for TPF 4.1

OK Cancel

5. In the **Name** field, type **Target Environment for tpfapp**.
6. From the drop-down list, select **Default Target Environment for TPF 4.1** to use the initial settings from this target environment for the new one that you are creating.
7. Click **OK** to save your selections. The New Target Environment dialog box opens.



New Target Environment - Target Environment for tpfapp

Create target environment

Specify the information for this target environment

System Information

Specify the system information for this target environment:

IP Address: ?

System: [✓ TPF 4.1] z/TPF

PUT Level: 19

Build and Link Options

Select the build and link options you want to use for this target environment

- ☐ Build and Link Options for tpfapp
- ☑ Default Build and Link Options
- ☑ Default Build and Link Options with Debug Options

Build Mechanism

Select the build mechanism you want to use for this target environment

Build list

Editor Options

Select the editor options you want to use for this target environment

Default Editor Options for TPF 4.1

Load Options

Select the load options you want to use for this target environment

- ☑ Default Load Options for TPF 4.1
- ☐ Load Options for tpfapp

8. In the **IP address** field, type the IP address of the system that you will use to test your application.
9. Click the **TPF 4.1** radio button.
10. From the **PUT level** drop-down list, select the PUT level that your test system is running at.
11. Expand the **Build and Link Options** section and select the **Build and Link Options for tpfapp** check box.
12. Expand the **Load Options** section and select the **Load Options for tpfapp** check box.
13. Click **OK** to create your new target environment definition and close the dialog box. The target environment that you created is added to the list of available target environments in the **Settings for the selected target environment** drop-down list.
14. Click **OK** to close the Preferences window.

Part 6: Creating a TPF project

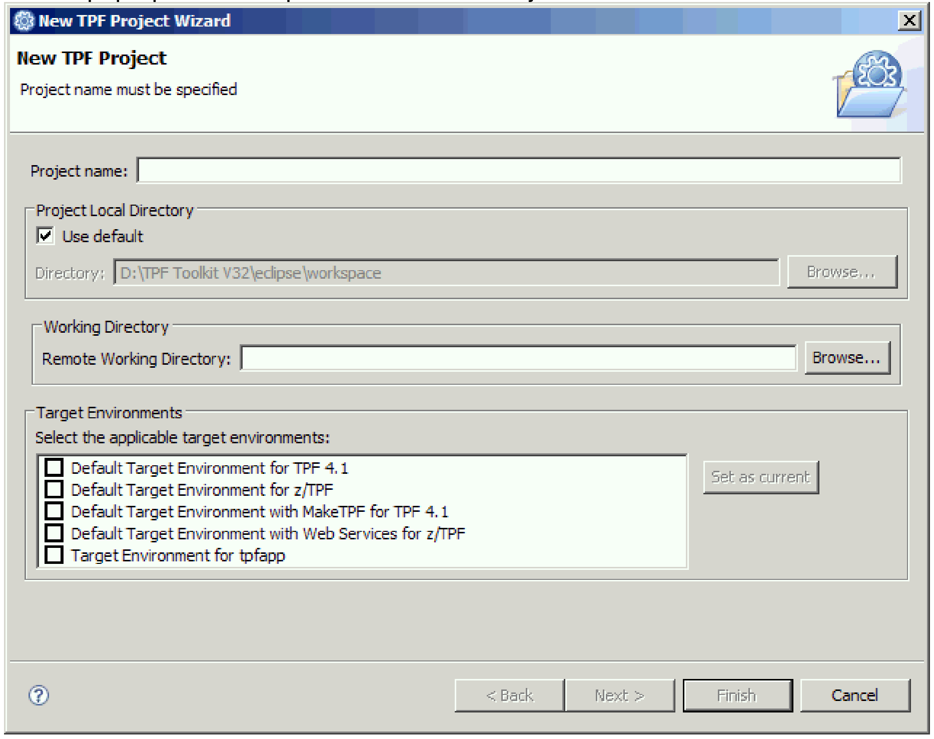
Once you define your target environment to capture the development and deployment environment for your application, you can create a TPF project. A TPF project is used to group resources (such as files or folders) from one or more systems that you need to perform a task. A TPF project can mean different things in different environments. For example, a TPF project can contain:

- Everything that you need to develop a TPF application.
- A version of a TPF application.
- An application that targets a specific TPF system.

TPF projects can also contain subprojects. You can use a TPF subproject to logically group different components of an application in TPF Toolkit. For the purposes of this tutorial, the TPF project represents the entire application, as it is too small to break up into components.

To create a TPF project, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, right-click anywhere in the view and select **New** > **Project** from the pop-up menu to open the New TPF Project Wizard.



New TPF Project Wizard

New TPF Project

Project name must be specified

Project name:

Project Local Directory

[✓] Use default

Directory: D:\TPF Toolkit V32\eclipse\workspace

Working Directory

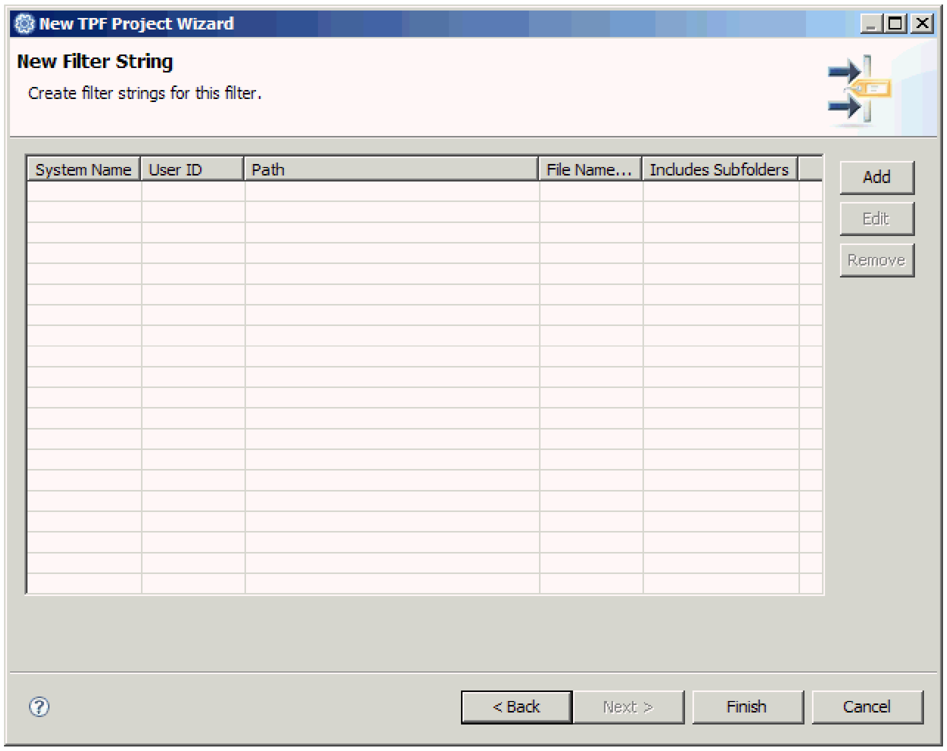
Remote Working Directory:

Target Environments

Select the applicable target environments:

- [ ] Default Target Environment for TPF 4.1
- [ ] Default Target Environment for z/TPF
- [ ] Default Target Environment with MakeTPF for TPF 4.1
- [ ] Default Target Environment with Web Services for z/TPF
- [ ] Target Environment for tpfapp

3. In the **Project name** field, type the name of the new project. For example, **tpfapp**.
4. In the **Project Local Directory** section, select the **Use default** check box to specify that you want to create the project in your local workspace.
5. In the **Working Directory** section, click **Browse** to browse for the location of your remote system.
6. Expand the tree to locate the /u/myuser/tpfapp folder where the sample TPF application currently resides, select this folder, and then click **OK**.
7. In the **Target Environments** section, select the **Target Environment for tpfapp** check box.
8. Click **Next** to proceed to the New TPF Filter page to create a filter for the TPF project. Filters allow you to populate the project with files and folders by grouping filter strings together to logically group similar resources.
9. In the **Filter Name** field, type **Filter**, and then click **Next** to proceed to the New Filter String page to populate the filter.



10. In the New Filter String page, click **Add** to open the TPF Toolkit Browse Dialog.
11. Select the **/u/myuser/tpfapp** folder to create a filter string that contains all of the files in the tpfapp application.
12. Click **Finish** to create the new project with the resources that you specified in the filter strings and close the New TPF Project Wizard.

The tpfapp project is displayed in the TPF Project Navigator view. When you expand the tpfapp project and its filters, all of the files contained in the tpfapp application are displayed.

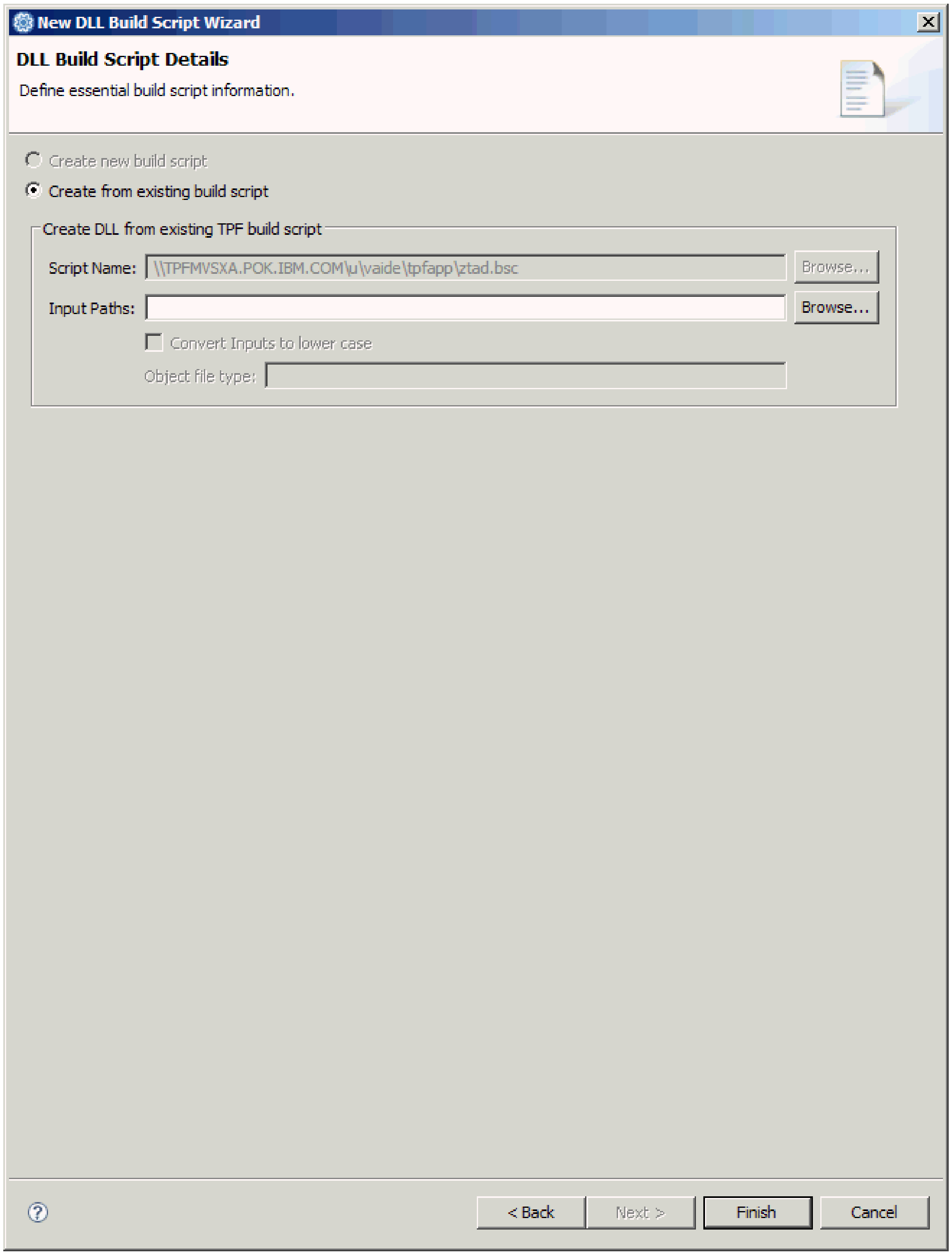
Part 7: Creating a TPF DLL build script

TPF Toolkit creates a special file to store the information needed to generate the TPF Dynamic Link Library (DLL) object required to create TPF applications.
The definition for a TPF DLL is stored in a file with the .dll extension.
Note: These are not the same build scripts as those used by the CBLD tool.

You need to create a TPF DLL build script file (.dll file) for the ZTAD TPF DLL for the tpfapp application. You can use the CBLD build scripts as input to the creation process. TPF Toolkit parses these files and uses the information to create the .dll file.

To create the ZTAD.dll file, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, double-click the tpfapp project to expand it, and then double-click the Filter to display all of the files in the tpfapp project.
3. Right-click **ztad.bsc** to open the pop-up menu.
4. Select **Create build script** to open the New DLL Build Script Wizard.


New DLL Build Script Wizard

DLL Build Script Details

Define essential build script information.

( ) Create new build script

(●) Create from existing build script

Create DLL from existing TPF build script

Script Name: \\TPFMVSXA.POK.IBM.COM\u\vaide\tpfapp\ztad.bsc

Input Paths:

[ ] Convert Inputs to lower case

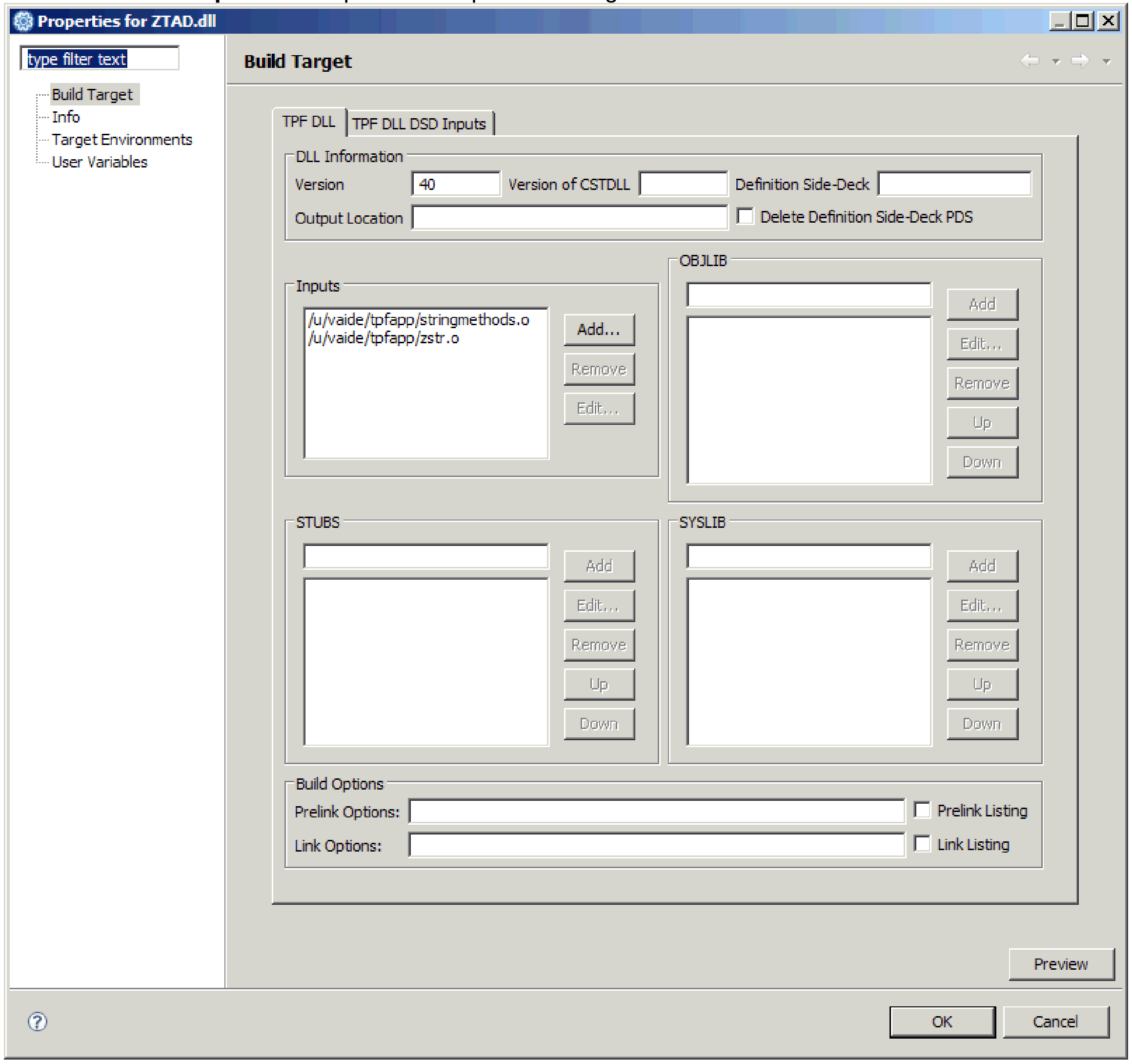
Object file type:

5. Since the inputs specified in the ztad.bsc build script file reside on z/OS UNIX, you must specify the z/OS UNIX folder in which these inputs reside. In the **Inputs path** field, type **/u/myuser/tpfapp**. This path is prefixed to all of the inputs mentioned in the CBLD build script.
6. Clear the **Convert inputs to lower case** check box since the ztad.bsc file already has the inputs specified in the correct case.
7. In the **Object file type** field, type **o** as the file name extension for the inputs on z/OS UNIX. This file name extension is appended to each input.

8. Click **Finish** to create the ZTAD.dll.

The ZTAD.dll build script file is displayed in the TPF Project Navigator view. Now you can set the properties for this file. To set the properties for the ZTAD.dll file, complete the following steps:

1. In the TPF Project Navigator view, right-click the ZTAD.dll file to open the pop-up menu.
2. Select **Properties** to open the Properties dialog box.


Properties for ZTAD.dll

Build Target

TPF DLL | TPF DLL DSD Inputs

DLL Information
Version: 40
Version of CSTDLL:
Definition Side-Deck:
Output Location:
☐ Delete Definition Side-Deck PDS

Inputs
/u/vaide/tpfapp/stringmethods.o
/u/vaide/tpfapp/zstr.o

Build Options
Prelink Options:
☐ Prelink Listing
Link Options:
☐ Link Listing

You need to populate some of the information in this properties page. This information is contained in the JCL that you did not specify in the set of build and link options in the target environment.

3. In the **Definition Side-Deck** field, type **/u/myuser/tpfapp**. This is the name of the z/OS UNIX folder into which the exported functions and variables are generated.
4. In the **Output Location** field, type **MYUSER.TPFAPP.LK1** to specify the dataset that contains the built DLL. This dataset originates from the SYSLMOD DD statement.
5. The reference to the Assembler module ZSTR in this DLL is a stub. Since the ztad.bsc build script file does not make a distinction between real objects and stubs, the stub definition is converted to an input. To correct this stub definition in the **Inputs** list, select **ZSTR.o**, and then click **Remove** to remove the input from the list.
6. In the **STUBS** section, type **ZSTR**, and then click **Add** to add ZSTR to the list of stubs.
7. Click **OK** to save your changes and complete the creation of the ZTAD.dll build script file based on information from ztad.bsc.

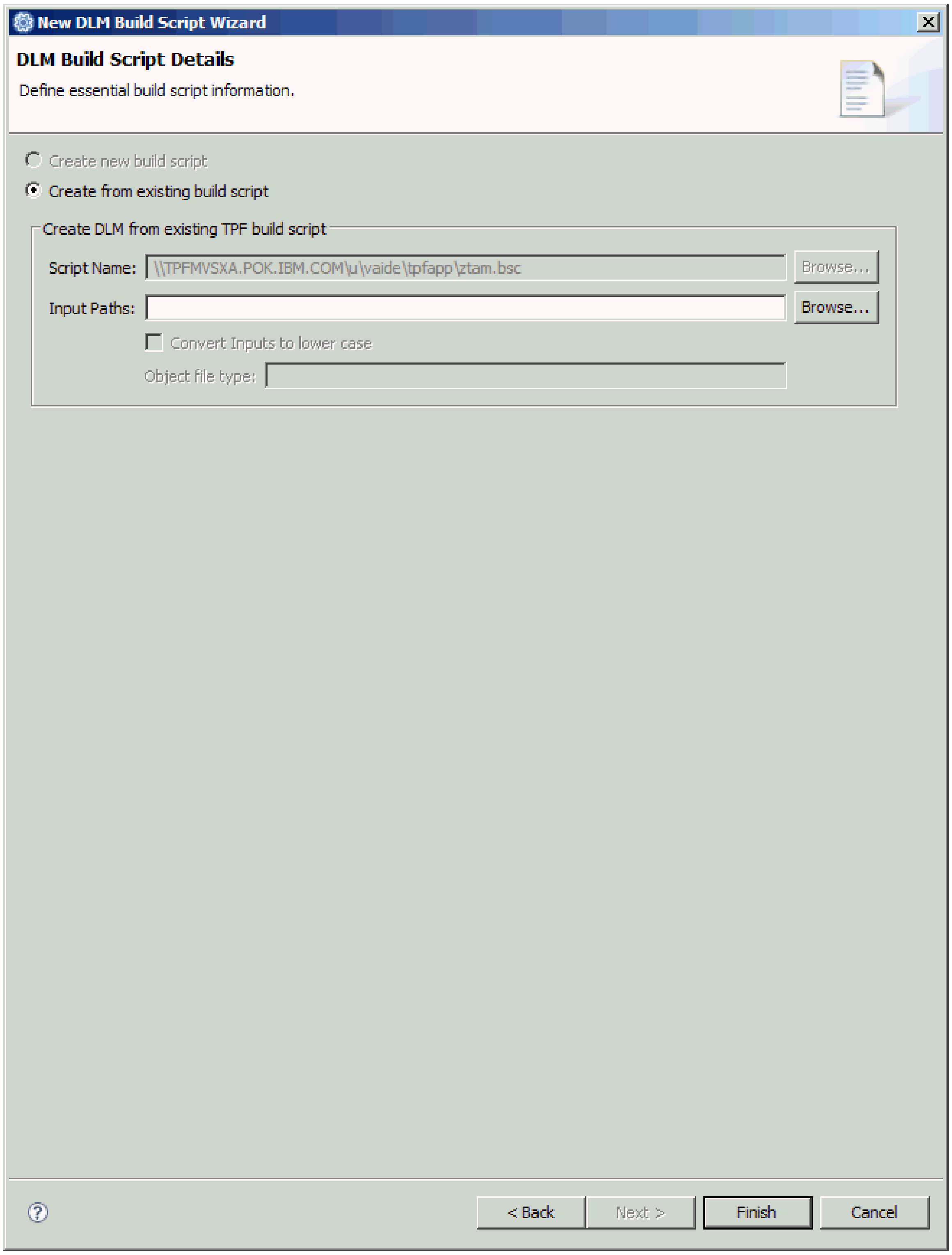
Part 8: Creating a TPF DLM build script

TPF Toolkit creates a special file to store the information needed to generate the Dynamic Load Module (DLM) object required to create TPF applications.
The definition for a DLM is stored in a file with the .dlm extension.
Note: These are not the same build scripts as those used by the CBLD tool.

You need to create a TPF DLM build script file (.dlm file) for the ZTAM DLM for the tpfapp application. You can use the CBLD build scripts as input to the creation process. TPF Toolkit parses these files and uses the information to create the .dlm files.

To create the ZTAM.dlm file, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, double-click the tpfapp project to expand it, and then double-click the Filter to display all of the files in the tpfapp project.
3. Right-click **ztam.bsc** to open the pop-up menu.
4. Select **Create build script** to open the New DLM Build Script Wizard.

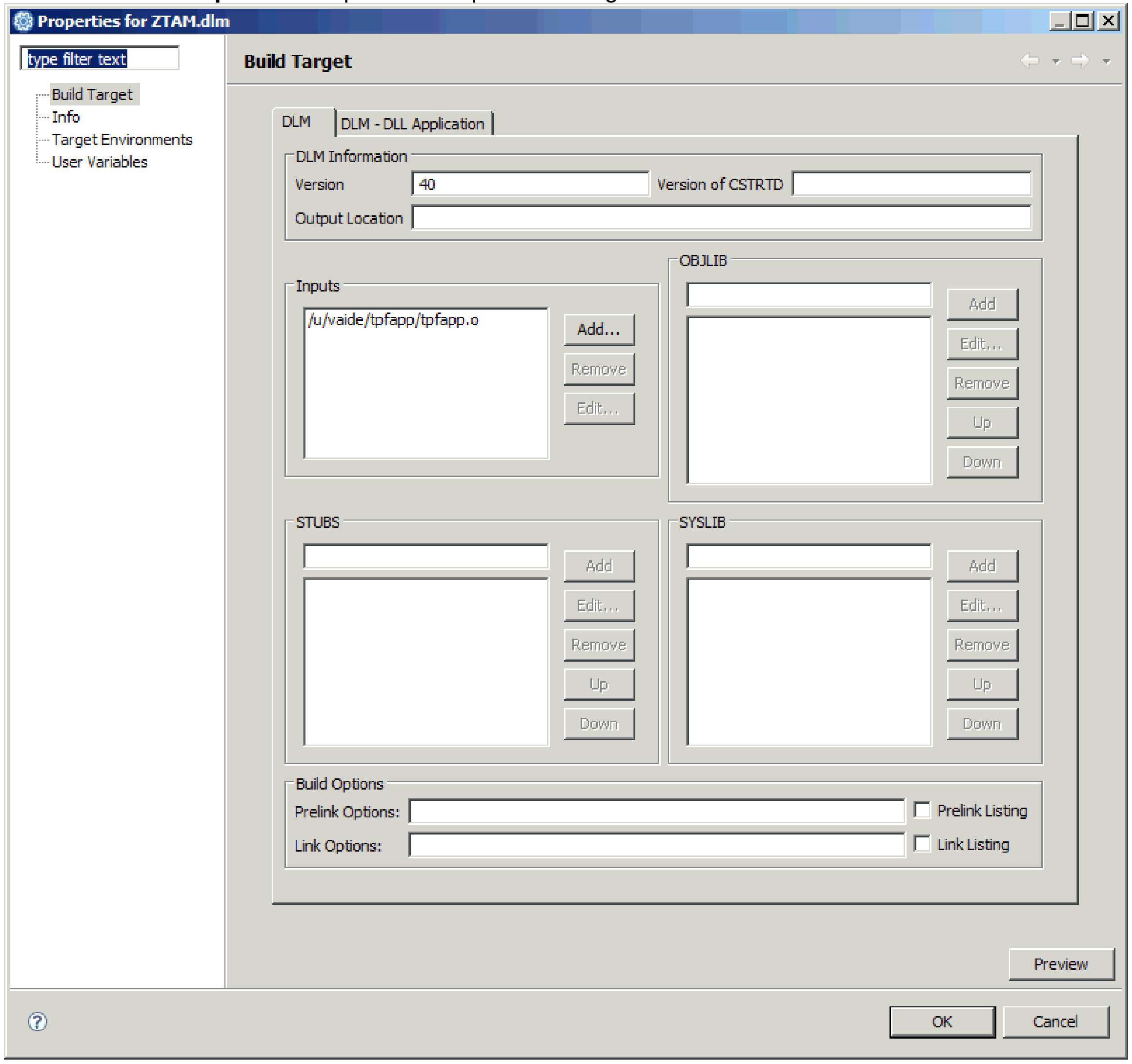

New DLM Build Script Wizard
DLM Build Script Details
Define essential build script information.
Create new build script
Create from existing build script
Create DLM from existing TPF build script
Script Name: \\TPFMVSXA.POK.IBM.COM\u\vaide\tpfapp\ztam.bsc
Browse...
Input Paths:
Browse...
Convert Inputs to lower case
Object file type:
< Back
Next >
Finish
Cancel

5. Since the inputs specified in the ztam.bsc build script file reside on z/OS UNIX, you must specify the z/OS UNIX folder in which these inputs reside. In the **Inputs path** field, type **/u/myuser/tpfapp**. This path is prefixed to all of the inputs mentioned in the CBLD build script.
6. Select the **Convert inputs to lower case** check box to convert inputs in the ztam.bsc file to lower case.
7. In the **Object file type** field, type **o** as the file name extension for the inputs on z/OS UNIX. This file name extension is appended to each input.

8. Click **Finish** to create the ZTAM.dlm.

The ZTAM.dlm build script file is displayed in the TPF Project Navigator view. Now you can set the properties for this file. To set the properties for the ZTAM.dlm file, complete the following steps:

1. In the TPF Project Navigator view, right-click the ZTAM.dlm file to open the pop-up menu.
2. Select **Properties** to open the Properties dialog box.



Version: 40
Version of CSTRTD:
Output Location:
Inputs: /u/vaide/tpfapp/tpfapp.o
Prelink Options:
Link Options:

You need to populate some of the information in this properties page. This information is contained in the JCL that you did not specify in the set of build and link options in the target environment.

3. In the **Output Location** field, type **MYUSER.TPFAPP.LK1** to specify the dataset that contains the built DLM. This dataset originates from the SYSLMOD DD statement.
4. Click **OK** to save your changes and complete the creation of the ZTAM.dlm build script file based on information from ztam.bsc.

Part 9: Creating a TPF loadset generation script

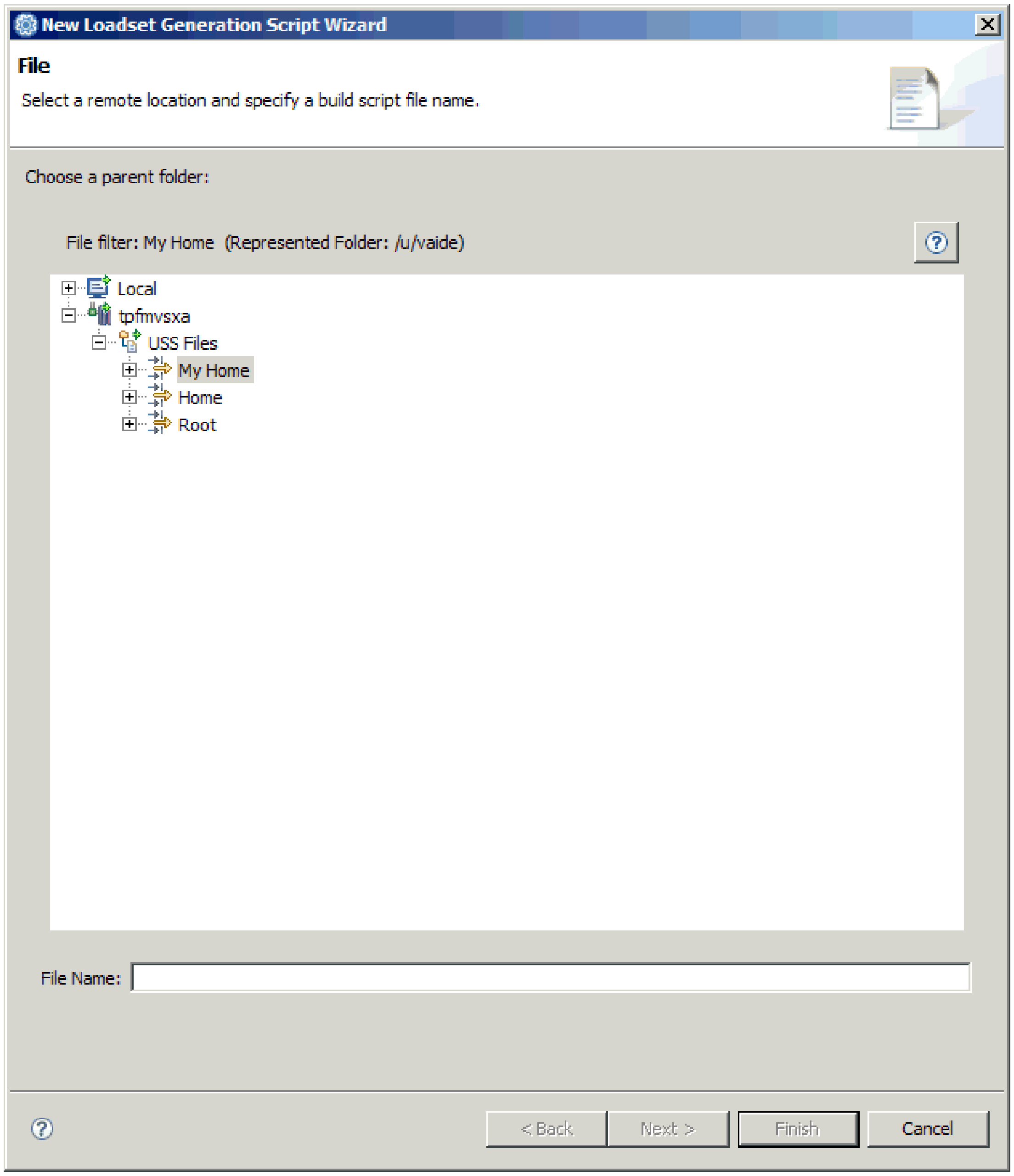
The loadset was originally generated by submitting a JCL script that combined all of the parts of the application and output it to a general dataset (GDS). The tpfapp application used tpfapp.jcl to create that loadset.

TPF Toolkit requires that you create a loadset generation script to create the loadset for the application. This file resides in the TPF project folder and has a .lset extension. All of the information required to create the loadset generation script comes the JCL script that you originally used to create a loadset.

You already specified some of the general information for creating loadsets in the Load Options preference page and included it in your target environment definition. The loadset generation script that you create will contain additional information that is specific to this loadset. When you are ready to generate the loadset, you can select the appropriate set of load options that you want to use to provide the general options. The information in both the loadset generation script (.lset file) and the selected load options are used to generate the loadset.

To create a loadset generation script, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, right-click anywhere in the view to open the pop-up menu.
3. Select **New** > **Loadset Generation Script** to open the New Loadset Generation Script Wizard.


New Loadset Generation Script Wizard

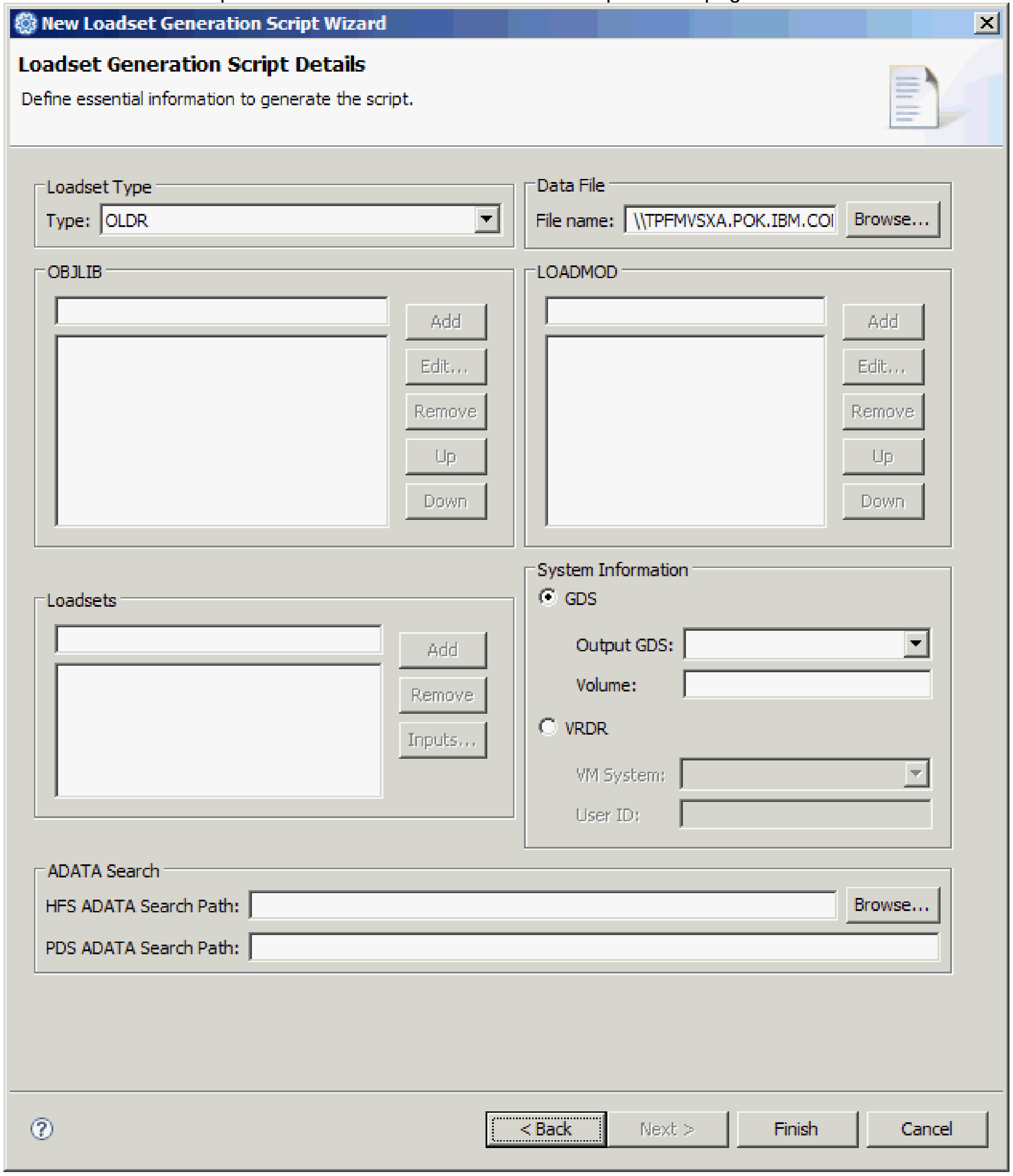
File

Select a remote location and specify a build script file name.

Choose a parent folder:

File filter: My Home (Represented Folder: /u/vaide)

4. From the tree of available resources, select the **/u/myuser/tpfapp** folder. This is the folder for the tpfapp application.
5. In the **File Name** field, type **tpfapp**.
   Note: If you omit the file name extension, the .lset file name extension is automatically appended to the file name.
6. Click **Next** to proceed to the TPF Project Filter page.
7. Select **Filter** as the TPF project filter that you want the new Loadset build script to appear in.
8. Click **Next** to proceed to the Loadset Generation Script Details page.


New Loadset Generation Script Wizard

Loadset Generation Script Details

Define essential information to generate the script.

Loadset Type
Type: OLDR

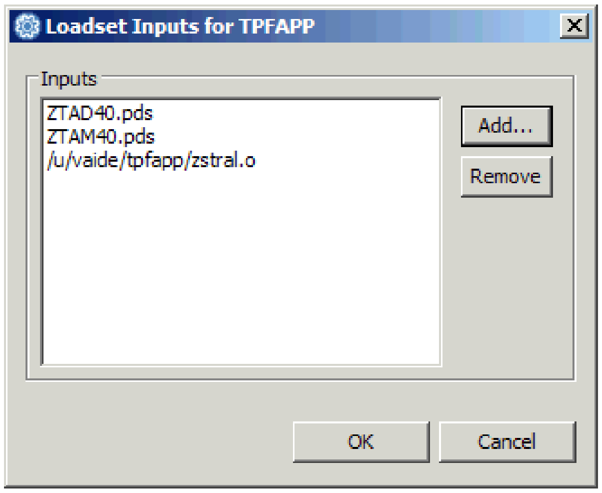
Data File
File name: \\TPFMVSXA.POK.IBM.COI

System Information
GDS [✓]
VRDR

9. From the **Type** drop-down list, select **OLDR**.
10. In the **OBJLIB** section, add the following values:
    - MYUSER.TPFAPP.LK1
    - MYUSER.TPFAPP.OBJ

These values come from the OBJLIB DD statement in tpfapp.jcl.

11. In the **LOADMOD** section, add **MYUSER.TPFAPP.LK1**. This value comes from the LOADMOD DD statement in tpfapp.jcl.
12. Use the **Loadset** section to specify the loadset modules defined in the control statements in tpfapp.jcl. In the **Loadsets** field, type **TPFAPP** to specify the name of your loadset, and then click **Add** to add the TPFAPP loadset to the list of loadsets.
13. From the list of loadsets, select **TPFAPP** to specify the loaddeck, and then click **Inputs** to open the Loadset Inputs dialog box.
14. To add modules to the TPFAPP loadset, click **Add** to open the Add Inputs dialog box.
15. In the **PDS Members** field, type **ZTAD40, ZTAM40** and then click **OK** to add ZTAD40 and ZTAM40 to the loadset.
    Note: The ZTAD40 DLL is generated into the MYUSER.TPFAPP.LK1 dataset and the ZTAD40 DLM is generated into the MYUSER.TPFAPP.LK1 dataset.
16. Click **Add**. In the **Files** field, type **/u/myuser/tpfapp/zstral.o** or click **Browse** to browse for the location of the file.
    Note: The ZSTRAL assembler module is now generated into the /u/myuser/tpfapp z/OS UNIX folder, to the zstral.o file.
17. Click **OK** to add ZSTRAL to the loadset.



ZTAD40.pds
ZTAM40.pds
/u/vaide/tpfapp/zstral.o

18. In the Loadset Inputs dialog box, click **OK** to save your changes and close the dialog box.
19. Click the **GDS** radio button to specify that you want to write the loadset to a GDS.
20. In the **HFS ADATA Search Path** field, type **/u/myuser/tpfapp**. The assembler ADATA file is generated into this z/OS UNIX folder.
    Note: The ADATA file must be included in the loadset to debug the application.
21. Click **Finish** to create the loadset generation script.

The following files are displayed in the tpfapp project:

- **tpfapp.lset** - the new loadset generation script.
- **tpfapp.data** - contains the loaddeck information.

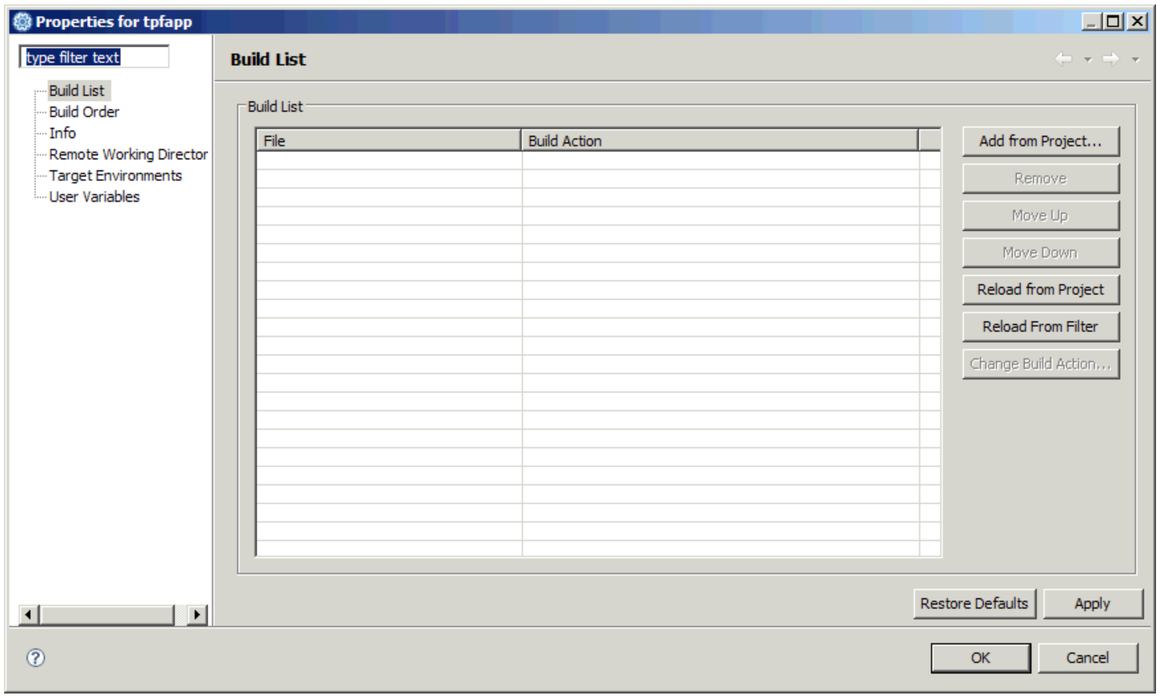
Part 10: Building the sample TPF application

Now that you have moved the application to TPF, you can set up the tpfapp project to build the tpfapp application. This is a two step process:

1. Set up the build list for the tpfapp project. The project build list is a list of files and build targets to be processed when a TPF project is built.
2. Build the tpfapp project.

To set up the build list for the tpfapp project, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, right-click the tpfapp project, and then select **Properties** from the pop-up menu to open the Properties dialog box.
3. In the left navigation pane, select **Build List** to open the Build List properties page.

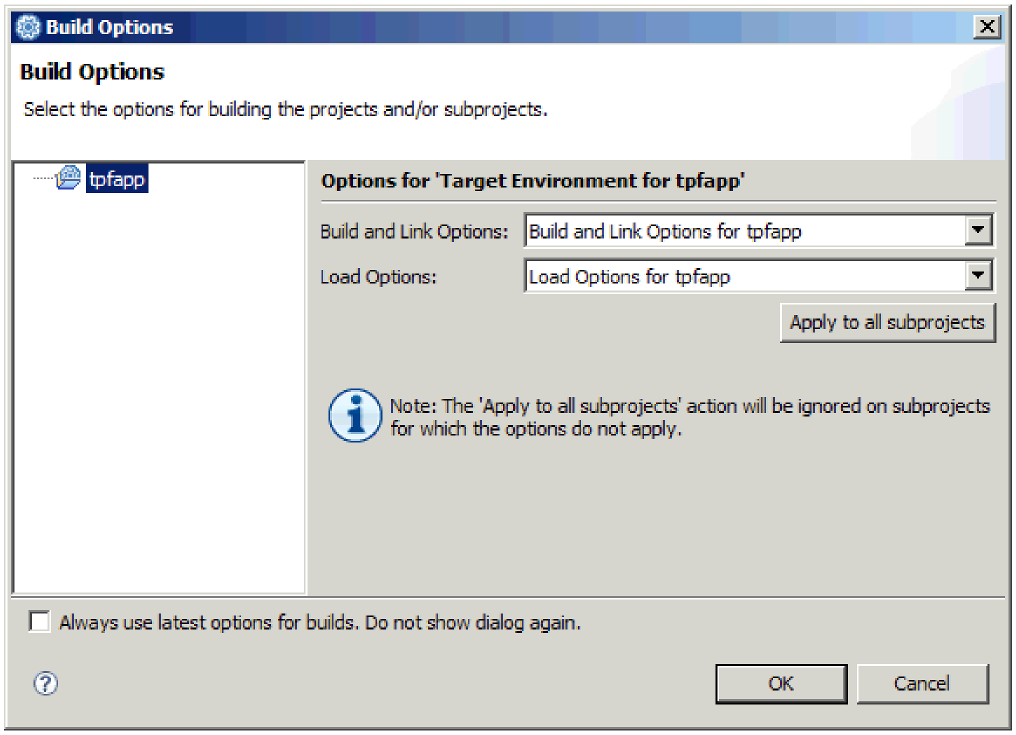


4. Click **Reload from Project** to populate the build list with all of the files in the project that can be built. These include *.s, *.asm, *.c, *.cpp, *.llm, *.dll, *.dlm, and *.lset files.
5. The order of the files in the build list might need to be modified. The .asm and .cpp files must be built first, then the ZTAD40 DLL, ZTAM40 DLM, and finally the TPFAPP loadset. To move these files into the correct order, select the file that you want to move, and then click **Move Up** or **Move Down** to change the position of the file within the build list.
   Note: Your final list should appear in the following order:
   - zstral.asm
   - stringMethods.cpp
   - tpfapp.cpp
   - ztad.dll
   - ztam.dlm
   - tpfapp.lset

   The default build action associated with each file type in the Default Build Actions preference page is assigned to each file in the build list.
6. Click **OK** to save your settings and close the Change Build Action dialog box.
7. Click **Apply** to save your settings, and then **OK** to close the Properties dialog box.

You are now ready to build the tpfapp project. To build the project, complete the following steps:

1. Switch to the TPF Toolkit perspective.
2. In the TPF Project Navigator view, right-click the tpfapp project to open the pop-up menu.
3. Select **Build** to open the Build Options dialog box.


Build Options

Build Options

Select the options for building the projects and/or subprojects.

Options for 'Target Environment for tpfapp'

Build and Link Options: Build and Link Options for tpfapp

Load Options: Load Options for tpfapp

Note: The 'Apply to all subprojects' action will be ignored on subprojects for which the options do not apply.

☐ Always use latest options for builds. Do not show dialog again.

4. From the **Build and Link Options** drop-down list, select the set of Build and Link Options for tpfapp.
5. From the **Load Options** drop-down list, select the set of Load Options for tpfapp.
6. Click **OK** to start the build.

Each item in the build list is built individually using the associated build action. The build items are built in the order in which they appear in the build list.

The set of build and link options that you specified for the tpfapp project are used to build the project. For example, the compile and assemble options are passed to the compiler to build your programs, as specified. The DLM and DLL options are combined with the information in the individual build script files (.dlm and .dll files) to link your programs. Finally, the default load options associated with the target environment for the tpfapp project are used to provide the general options for generating the loadset, as defined by the loadset generation script (.lset file).

You can view the build progress of each item in the Progress view. Build messages are displayed in the Remote Console view and compile errors are displayed in the Remote Error List.

If a C/C++ or Assembler program item fails to build, you can select the result in the Progress view. You can filter the errors in the Remote Error List to only display errors associated with that build. Right-click any errors in the Remote Error List to open the associated source file in the editor, at the location of the error.

Correct all of the errors. Once you correct the errors, you can rebuild the project using the Build action. After a successful build, the tpfapp application loadset is in the specified GDS, ready to load onto the TPF system to be run or debugged.
Tip: For more information on debugging a TPF application, see the Related topics section below.

Summary

In this tutorial, you have learned how to bring a sample TPF 4.1 application into the TPF Toolkit V3.4.0 environment so that you can build, maintain, and debug it using TPF Toolkit.

Appendix A: Moving files from datasets to z/OS UNIX System Services

Typically, TPF application development done in a z/OS environment uses several partitioned datasets (PDS). Each PDS is used to store a different source type – one for assembler source, another for C++ source, and another for JCL scripts. Each member of the PDS for that type is source for a different program or module.

z/OS UNIX is a POSIX-based environment. Files are stored in directories; directories can also contain other directories. This type of organization is called a Hierarchical File System (HFS) and starts with a single root directory that contains all of the other files and directories on the system. Typically, this directory is called /.

Each user on z/OS UNIX is given their own branch of the directory tree to work in, typically under the directory /u. Their home directory is typically the same name as their user ID, for example, **myuser**. The home directory for myuser is /u/myuser. TPF Toolkit only supports development in the z/OS UNIX environment.

You should structure your application so that all of the files for the application are rooted under one directory. You can create subdirectories within that root application directory to further categorize your source files by component or by file type.

For example, you can create a subdirectory called **src** that contains all of the source files for your application and a directory called **objs** that contains all of the include files (or header files) for your application. For the tpfapp application used in this tutorial, create a folder called **tpfapp** in the **myuser** directory. Since the sample application is a small application with only a few source files, you do not need to create any subdirectories.

The type of a source file in a PDS environment is determined by the PDS it resides in. z/OS UNIX uses an extension to determine the type of the file. The extension is typically three characters in length and is appended to the file name using the period (**.**) character. For example, an assembler source file in the tpfapp application that resides in a PDS is referenced as MYUSER.SOURCE.ASM(ZSTRAL). When this assembler source file is moved to z/OS UNIX, it is referenced as /u/myuser/tpfapp/zstral.asm.

You can physically copy the source from a PDS member to a z/OS UNIX directory using the TSO OPUT command. For example, to copy the ZSTRAL assembler source to z/OS UNIX, type the following command:

```
OPUT SOURCE.ASM(ZSTRAL) '/u/myuser/tpfapp/zstral.asm'
```

Note: For more information on the OPUT command, see the TSO command help.

Once all of the source files required for the application are copied to a z/OS UNIX directory, you can begin to use TPF Toolkit to develop, build, and debug your TPF application.