

**DevOps Test Embedded
Target Deployment Port Tutorial**



Special notice

Before using this information and the product it supports, read the information in [Notices on page xxi](#).

Contents

- Chapter 1. Introduction..... 4**
- Chapter 2. Preparing for a tutorial..... 5**
- Chapter 3. Creating a new Target Deployment Port..... 6**
- Chapter 4. Editing a Target Deployment Port 8**
- Chapter 5. Selecting a Target Deployment Port..... 10**
- Chapter 6. Validating a new Target Deployment Port..... 11**
 - Creating a New Configuration..... 11
 - Applying a Configuration to a Project..... 11
 - Validating the Compilation Procedure..... 12
 - Debugging a Target Deployment Port..... 12
- Chapter 7. Customizing a Target Deployment Port..... 14**
 - Using a Debugger (RTRT_USR + debugger)..... 17
 - Using Break Point Mode (RTRT_NONE)..... 18
- Notices..... xxi
- Index.....

Chapter 1. Introduction

The Target Deployment tutorial contains advanced information to help you to customize a new Target Deployment Port (TDP).

HCL DevOps Test Embedded (**Test Embedded**) is a complete runtime analysis and testing solution for real-time and embedded systems. It addresses all runtime analysis needs and all test levels including component and system testing for the C, C++, Ada.

General information about using the product can be found in the DevOps Test Embedded documentation.

If you are using the product for the first time, refer the DevOps Test Embedded Online documentation.

If you are upgrading from a previous version of DevOps Test Embedded, refer to Upgrading a Target Deployment Port.

Audience

This Target Deployment Tutorial is intended for advanced users of the product. Advanced knowledge of the target compiler, platform, development and test environment are required for Target Deployment Port customization tasks. Knowledge of Perl scripts is also required.

Chapter 2. Preparing for a tutorial

The purpose of this example is to demonstrate how to create and validate a new TDP on Windows.

This Tutorial will guide you through the steps of creating, modifying and debugging TDP, using custom I/O functions, a debugger and defining a break point strategy.

An example project for this tutorial, names `add.rtp`, is provided with DevOps Test Embedded in the *<Installation folder>/examples/TDP/tutorial* directory.

The TDP for this tutorial is based on the MinGW (Minimalist Gnu for Windows) C compiler distribution. MinGW is a collection of header files and import libraries that allow one to use GCC and produce native Windows32 programs that do not rely on any 3rd-party DLLs.

The MinGW distribution includes GNU Compiler Collection (GCC), GNU Binary Utilities (Binutils), GNU debugger (Gdb), GNU make, and various other utilities.

Perform the following steps to obtain a copy of the MinGW environment:

1. Go to <https://www.mingw-w64.org/>
2. Locate and download the latest complete MinGW distribution.
3. Follow the instructions provided with the distribution for installation and configuration.

Chapter 3. Creating a new Target Deployment Port

In most cases, you cannot create a TDP from scratch but rather base your new TDP on an existing TDP template. In this example, you can adapt an existing TDP `gccmingw_template.xdp` to your own environment.

The TDP file format is `.xdp`, as in XML Deployment Port. There are file-naming conventions when creating a new TDP:

- `c` for a C or C++ TDP, `a` for an Ada TDP.
- An acronym for the target platform host, in this case call it `wingcc` for Windows GCC.
- The name of the development environment `mingw`. Therefore, your TDP filename must be **`cwingccmingw`**.

All TDPs are in the following directory: `<Installation folder>/targets/xml/<tdp_name>.xdp`

Where `<Installation folder>` is the installation directory, and `<tdp_name>` is the name of the TDP.

Perform the following steps to start the TDP Editor:

1. In DevOps Test Embedded from the **Tools** menu, select **TDP Editor** and then Start.
- or
2. From the command line, type `tdpedit`.

Perform the following steps to open a TDP template

1. Select **Open**, from the **File** menu in the **TDP Editor**.
2. In the target subdirectory, select the `gccmingw_template_tutorial` TDP file.
3. Right click the Top-level node in the tree-view pane: **Gnu 6.3 (minGW)**.
4. Select **Rename** and enter a new name for this TDP:

For example: **`gcc 6.3 (minGW) Win`**

This name identifies the TDP in the DevOps Test Embedded GUI.

In the Comment for the root node section, enter the following information such as TDP developer and target environment.

Host Machine : Windows 10

Compiler : Gnu Compiler 6.3 (mingw)

Linker : Gnu Linker 6.3 (mingw)

Debugger : Gnu Debugger 7.6.1 (mingw)

OS : Native

Target : Native

This makes things easier when sharing the TDP with other users.

Perform the following steps to save the new TDP:

1. Select **Save xdp As** from the **File** menu.
2. Save your new TDP as cwingccmingw.xdp.
3. Select **Save and Generate** from the **File** menu.

The TDP Editor automatically creates a directory named cwingccmingw and save the .xdp file in that location.

Chapter 4. Editing a Target Deployment Port

The TDP Editor consist of four main sections:

- **A Navigation Tree:** Use the navigation tree on the left to select customization points.
- **A Help Window:** Provides direct reference information for the selected customization point.
- **An Edit Window:** The format of the **Edit** window depends on the nature of the customization point.
- **A Comment Window:** Provides you to enter a personal comment for each customization point.

In the Navigation Tree, you can click on any customization point to obtained detailed reference information for that parameter in the **Help** window.

The Navigation Tree covers all the customization points of the TDP.

There are four main sections:

Basic Settings

This section specifies default file extensions, default compilation and link flags, environment variables and custom variables required for your target environment. This section allows you to set all the common settings and variables used by DevOps Test Embedded and the different sections of the TDP. For example, the name and location of the cross compiler for your target is stored in a Basic Settings variable, which is used throughout the compilation, preprocessing and link functions. If the compiler changes, you only need to update this variable in the Basic Settings section.

Build Settings

This section configures the functions required by the DevOps Test Embedded GUI integrated build process. It defines compilation, link and test run Perl scripts, plus any user-defined scripts when needed. This section is the core of the TDP, as it drives all the actions needed to compile and execute a piece of code on the target.

Library Settings

This section describes a set of source code files as well as a dedicated customization file (**custom.h**), which adapt the TDP to target platform requirements. This section is definitely the most complex and usually only requires customization for specialized platform TDPs (unknown RTOS, no RTOS, unknown simulator, emulator, etc.)

Parser Settings

This section modifies the behaviour of the parser in order to address non-standard compiler extensions, such as for example, non-ANSI extensions. This section allows DevOps Test Embedded to properly parse your source code, either for instrumentation or code generation purposes.

The embedded Help on the of the TDP Editor window, provides contextual reference information for the part of the TDP that is selected in the tree view pane.

Perform the following steps to edit the new TDP:

Use the TDP Editor's tree pane to navigate through the customization points of the TDP, and make the following changes:

1. Under Basic Settings: Change the **ENV_PATH** and customization points in both **For C** and **For C++** nodes. **ENV_PATH** updates the **PATH** environment variable to invoke the **gcc** compiler directly. For example:

```
ENV_PATH C:\Gcc\bin;%ENV{ 'PATH' }
```



Note: When you modify a customization point in the TDP Editor, it is generally a good idea to add a note in the **Comment** box. This makes later modifications and TDP sharing much easier.

2. In the same manner, check all the other customization points to ensure that they reflect the correct path and file names as provided with the MinGW distribution.
3. Under **Build Settings**, no changes is required however you can look at the **Compilation Function**.

Locate the corresponding Perl script and refer the Help window to understand how the **atl_cc** routine works.

Next, look at the **Link Function** to understand the **alt_link** Perl routine.



Note: All the parameters used by these Perl routines are set in the **Basic Settings** section of the TDP.

4. Under **Library Settings**, no changes are required at this point.
5. Under **Parser Settings**: The 4 sub sections are used to adapt DevOps Test Embedded parser to specific user's code and compiler behaviour. No specific changes for gcc base compiler.
6. Save the TDP.

Any changes made to the Basic Settings section of a TDP are read from the DevOps Test Embedded GUI and applied to the project. For this reason, whenever you modify the Basic Settings of a TDP that is currently used in a DevOps Test Embedded project, **you must reload the TDP into the project.**

Chapter 5. Selecting a Target Deployment Port

The Target Deployment Port (TDP) settings are read or loaded when a HCL DevOps Test Embedded (**Test Embedded**) project is opened, or when a new Configuration is used.

First, if you saved your **Test Embedded** new TDP in another location than the installation folder, you need to add the new folder location in the **Test Embedded** GUI.

In Studio you can go to **Edit > Preferences > Project > TDP directories**.

In Visual Test go to **Window > Preferences > DevOps Test Embedded > Target Deployment Port**.

For the next steps you can use the classic version of HCL DevOps Test Embedded Studio (Test Embedded Studio).

Chapter 6. Validating a new Target Deployment Port

After a TDP is created or modified, the first step is to validate that it works correctly on the target.

The first step is to change the TDP used by your project.

To make sure that your TDP is working properly, you must create a Component. Testing test node and run it with all the relevant Runtime Analysis tools enabled.

After the following steps are covered, you can consider that your TDP is fully functional:

- Create a new DevOps Test EmbeddedConfiguration
- Apply the new Configuration to a project
- Validate the compilation sequence with the new Configuration.

Creating a New Configuration

In HCL DevOps Test Embedded (**Test Embedded**), the TDP is part of a Configuration. Each Configuration is based on a TDP, plus the Configuration Settings that are specific for each node of the project.

This means that you can base several slightly different Configurations on a single TDP.

Perform the following steps to create a new Configuration in **Test Embedded**:

1. In Test Embedded Studio, open the add.rtp example project in Studio.

For this tutorial, concentrate on the add test node, which contains a simple add.c source file as well as the corresponding add.ptu test script.

2. Select **Configurations** from the **Project** menu, and then click **New**.
3. Enter a name for the new Configuration in the **New Configuration** box, and then select the TDP on which it is based.
4. Click **OK**.

For example, select your newly created MinGW TDP. Notice that two items appear in the list, one for C, another for C++ followed by the same name. Select the C version of the TDP

5. Click **Activate**, Close and save the project. Update the TDP in the project.

Applying a Configuration to a Project

Now that the new Configuration is created, based on your TDP, you need to select it for use in your project.

Although a project can use multiple Configurations, as well as multiple TDPs, there must always be at least one active Configuration.

TDP is used when selected from the Build combo-box but remember that you must be consistent between the TDP programming language selection and the source files used within your test environment.



Perform the following steps to change the current Configuration of a project:


1. Select the Configuration you want to use in the Configuration box from the Build toolbar.
2. Update any project settings if required.

Validating the Compilation Procedure

To validate the compilation sequence, the idea is to successfully compile the current project with the new Configuration.

Perform the following steps to validate the compilation procedure:

1. Select a single source file in the **Project Explorer**.
2. Click the Build Options  button from the Build tool bar and then clear all **Runtime Analysis** features such as Memory Profiling, Performance Profiling, Code Coverage and Runtime Tracing to ensure that these do not affect the build sequence.
3. Select the `add.c` source file.
4. Click **Build** from the Build toolbar .

The compilation should end with a Passed  status. If not, restart the TDP Editor and change the `atl_cc` Perl procedure accordingly.

You can repeat the same action for the following Perl procedures:

- `atl_cpp`: Preprocessing routine for Source Code Insertion
- `atl_link`: Link routine
- `atl_exec`: Execution routine
- `atl_execdbg`: Debugging routine

The compilation procedure is validated. You can now consider using the Test and Runtime Analysis features of **Test Embedded** on your project.



The next section provides help about debugging any compilation issues you may have encountered.

Debugging a Target Deployment Port

If everything does not work as it should, the following method might help you troubleshoot TDP issues with HCL DevOps Test Embedded (**Test Embedded**).

Perform the following steps to troubleshoot a TDP:

1. Set the verbose mode by using Test Embedded Studio.
2. Go to **Edit > Preferences > Project > Verbose output** Or,
3. Set the `ATTOLSTUDIO_VERBOSE` environment variable to 1. The exact procedure to do this depends on your operating system.

4. The **Test Embedded** GUI does not automatically inherit the Windows environment. Therefore, you must save the project, close and relaunch the GUI.
5. Select Configurations from the **Project** menu, and then click **New** to select the new TDP if necessary. Ensure that the correct TDP is selected.
6. Decompose the complete build process into multiple steps. To do this, click the Build Option drop-down icon , clear the **All** option and select only the first step of the compilation sequence (Source compilation). Clear any Runtime Analysis tools.
7. Select the source file under test (add.c in this example) and click **Build** icon .
8. Repeat the same operation for each other compilation step and source file until the whole node can be successfully processed.

This should provide adequate feedback to help you debug each individual step of the compilation sequence.

In the current example, any problems encountered will usually be related to an incorrect file path in the Basic Settings of the TDP.

Chapter 7. Customizing a Target Deployment Port

This section of the Tutorial demonstrate how to use the customize input-output (I/O) communication and break-point usage to address a target system without standard I/O functions.

Before starting, a short description of the Execution phase, the steps are:

- **Test Embedded** request the TDP to execute the executable test then wait for the end of its execution.
- When the TDP receive the execution request it run the perl script located in the **TDP > Build settings > execution function**

This perl script is in charge of:

1. Download the code onto the target, and then start the code execution on the target.
2. Retrieve the output data generated by the code execution from the target using the TDP data retrieval method.
3. Stop the target or get ready for the next test.

User-defined I/O Primitives (RTRT_USR) - Example 1

Perform the following steps to create a new TDP:

1. Open the cwingccmingw.xdp TDP in the TDP Editor. Select the top-level node and rename it.

For example, C gcc 6.3 (minGW) Win User Mode

2. Select **Save xdp As** from the **File** menu to save the new TDP as cwingccmingwUserIO.xdp .
3. Collapse all the nodes in the Navigation window as this section concentrates only on the Build Settings and Library Settings nodes of the TDP Editor.

Library Settings

You first need to specify the I/O user mode, which means disabling the standard I/O mode for data retrieval on the target.

By default, when executing a program compiled with **Test Embedded**, the test data is dumped to a file on the file system by using the standard fopen, fprintf and fclose functions. On some platforms, these primitives are not available hence the need to use a set of user-defined I/O functions that allow the TDP to access the File System.


Perform the following steps to change the Library settings:

1. Expand **Library Settings > Data retrieval and Error message output**, and then select **Data retrieval** to locate the RTRT_IO macro definition.

In the combo-box for RTRT_IO you can select:

- **RTRT_NONE**: Standard I/O not available
 - **RTRT_STD**: Standard I/O functions (fopen, fprintf and fclose)
 - **RTRT_USR**: User-defined I/O. This option enables the customization tabs.
2. Select **RTRT_USR**. Look at the user defined I/O primitives used to access the File System: `usr_open`, `usr_writeln` and `usr_close`.

Note that `usr_writeln()` contains the following statement `printf("$s",s)`;

3. Select **Save and Generate** from the **File** menu.
4. Update the Configuration in **Test Embedded** to use the My MinGW UserMode TDP and Build  your sample project.

The message console display the following information, when the build fails:

```
Executing gcc_step1\Histo.exe ...
gcc_step1\Histo.exe
PU "Histo"
H0 "... "
O1
NT "Initialization" 0 0
DT 0 ...
A32 OK RA=T
NT "Termination" 61 41
DT 0
FT 91e544c5DC 0b72d3c1
PT "Termination"
PS 0 0 0
PY 0 0 0
QT "Termination"
QS 91e544c5 7965f082
NO "2 (Max Calling Level reached)"
CI 0h Splitting 'gcc_step1\THisto.rio' traces file...
Traces file successfully split.
No RIO instruction found.
Errors have occurred.
```

This message shows that:

- ASCII character data was dumped from the program directly to the standard output of the executable through the `printf` directive.
- Test data output is encoded information that only the **Test Embedded** Report Generator can understand.
- The trace file is empty. Although the split is successful, no instructions are found, and an error message occurs.

Therefore, for the build to be successful, you must provide the Report Generator with a valid trace file.

Build Settings

The Execution function is a basic command that produces an output file that redirects the standard output to `$out`.

Perform the following steps to change Build settings:

1. In the TDP Editor, expand the **Build Settings** and select **Execution** function. The following code is displayed:

```
sub atl_exec($$$)
{
  my ($exe,$out,$parameters) = @_ ;
  unlink($out);
  SystemP("$exe $parameters");
}
```

2. Change the SystemP line to:

```
SystemP("$exe $parameters >$out");
```

3. Save the TDP, update the Configuration in **Test Embedded** and Build  your sample project.

Now the test should run smoothly and produce complete reports. If not, rework the above functions until the execution is successful.

User-defined I/O Primitives (RTRT_USR) - Example 2

The following section demonstrates how to define your own I/O primitives for the dump phase.

Again, create a new TDP based on the one created previously.

Perform the following steps to create a new TDP:

1. Open the **cwingccmingwUserIO.xdp** TDP in the TDP Editor.
2. Select the top-level node and rename it **My MinGW UserMode2**.
3. Save the current TDP as **cwingccmingwUserIO2.xdp**.
4. Collapse all the nodes in the Navigation window as this section concentrates only on the Build Settings and Library Settings nodes of the TDP Editor.

Perform the following steps to set up user-defined I/O primitives:

1. Expand the **Build Settings**, and then select the **Execution function**.
2. Delete the **> \$out** parameter that was added to the SystemP statement:

```
SystemP("$exe $parameters");
```

3. Expand **Library Settings > Data retrieval and Error message output**, and then select **Data retrieval** to locate the RTRT_IO macro definition.
4. Select the RTRT_USR entry.
5. On the **Settings** tab, in the RTRT_FILE_TYPE, change int to FILE*.
6. Add your own code for the usr_open function, such as:

```
printf("...Opening file...\n"); return(fopen(fileName,"w"));
```

7. Add your own code for the usr_init function:

```
return(0);
```

8. Add your own code for the usr_writeln function:

```
printf("...Dumping : %s\n",s); fprintf(f,"%s",s);
```


9. Add your own code for the `usr_close` function:

```
printf("...Closing file...\n"); fclose(f);
```

10. Save the TDP, update the Configuration in **Test Embedded** and Build  your sample project.

The examples provided may not have practical applications in real-life scenarios as they are essentially the same as the standard I/O mechanism. However, they serve to illustrate the simplicity of mapping user-defined I/O primitives to the data retrieval mechanism implemented by the TDP



Using a Debugger (RTRT_USR + debugger)

Before moving to the next step, you need to understand how **Test Embedded** uses the GDB debugger command. This function is called when the Debug build option is selected in the **Test Embedded** GUI.



Note: This is NOT a break-point strategy. The Debug option merely allows you to manually inspect application execution.

Perform the following steps to build a node in **Test Embedded** Debug mode:

1. In the **Test Embedded** Project Explorer, select the project node.
2. Click the **Build** option from the **Build** toolbar  drop-down and select **Debug** in the **Build Options** window.
3. Select the `add.c` node in the **Project Explorer**. Click the **Build**  icon from the Build toolbar.

This runs a command line window with the GDB up and running.

Debugger commands

In the GDB window, type the following commands:

```
break priv_writeln
break priv_close
display atl_buffer
run
```

If you type **c** or **cont** for continue you should see the `atl_buffer` contents changing and showing information like what you obtained in the Message Console with the `printf` command.

Debug Results

The **priv_writeln** and **priv_close** primitives are implemented within the TDP. The former is interpreted as a dump request event, whereas the latter is an end of test run event.

The `atl_buffer` symbol (default size is 1024 bytes) dynamically gathers information from the test run.

```
...
...Dumping : NO "1 (Max Calling Level reached)"
Breakpoint 1, priv_writeln (file=0x77b34660 <msvcrt!_iob+96>, str=0x40f220 <atl_buffer> "CI 0\n")
at D:/TDP base tuto/cwingccmingwUserIO_2/lib/priv.c:318
```

```

318     usr_writeln((RTRT_FILE_TYPE)file,str);

1: atl_buffer = "CI 0\n\000(Max Calling Level reached)\n\n\000ation\n\nQD 0\nQS 3052c435 3536e587\nZT
  \Termination\n" 04\nPT \Termination\n\nPS 0 0 0\nPY 0 0 0\n\000\062ef37b2 2ef37b2 2ef37b2 2ef37b2\nZT
  \add/1\n" 04\nZF 3052c435 3536e587\nZM 1 07\n"...
...
...Dumping : tdp end
Breakpoint 2, priv_close (file=0x77b34660 <msvcrt!_iob+96>)
  at D:/TDP base tuto/cwingccmingwUserIO_2/lib/priv.c:270
270     usr_close((RTRT_FILE_TYPE)file);
1: atl_buffer = "\000dp end\n\000x Calling Level reached)\n\n\000ation\n\nQD 0\nQS 3052c435 3536e587\nZT
  \Termination\n" 04\nPT \Termination\n\nPS 0 0 0\nPY 0 0 0\n\000\062ef37b2 2ef37b2 2ef37b2 2ef37b2\nZT
  \add/1\n" 04\nZF 3052c435 3536e587\nZM 1 07\n"...

```

The objective is to produce a file on the file system just as we did with the standard I/O functions or the user-defined I/O functions.

When a break point strategy is required, the manual process done (GDB commands) must be somehow automated.

Using Break Point Mode (RTRT_NONE)

On breakpoint mode systems, no I/O functions are available on the target platform. This is usually the case with small target calculators, such as those used in the automotive industry, running on a microprocessor simulator or emulator with no operating system.

If no communication functions are available on the target platform, the best alternative is to use a debugger logging mechanism, assuming one exists.

Using Break Point mode can be summed up as the following tasks:

- Compile, link and load the executable in the debugger. This is typically handled by the GUI, so no action is required.
- Dump the content of **atl_buffer** each time the break point on **priv_writeln** is met.
- Quit the debugger when the **priv_close** is reached
- Ensure sure that the file produced is ASCII

To do this, you must specify a break point for I/O. This means that you will no longer use the standard I/O or the user-defined I/O functions.

Perform the following steps to create a TDP Breakpoint mode:

1. Open the **cwingccmingwUserIO2.xdp** TDP in the TDP Editor.
2. Select the top-level node and rename it My **MinGW breakpoint Mode**.
3. Save the current TDP as **cwingccmingwBreakpoint.xdp**.

Perform the following steps to disable I/O functions:

1. Expand Library Settings, Data retrieval and error output and select Data retrieval to locate the RTRT_IO macro definition. In the combo-box for RTRT_IO you can select:
 - RTRT_NONE: No I/O available
 - RTRT_STD: Standard I/O functions
 - RTRT_SOCKET: Use socket connection
 - RTRT_USR: User-defined I/O. Only this option allows you to access the customization tabs.
2. Select RTRT_NONE. This is a common preference, especially in situations where there are constraints on target platforms, such as the absence of an operating system and a file system.

Dumping the Buffer


To capture the content of `atl_buffer` each time the breakpoint on `priv_writeln` is hit, you can achieve this without a file system by specifying the usage of the GDB debugger command line in the `atl_exec` Perl script.

The debugger documentation explains how to call **gdb** and how to automate the use of the debugger through a command script

Perform the following steps to invoke the debugger from the `atl_exec`:

1. Expand **Build Settings** and select **Execution function** to locate the `atl_exec` Perl function.
2. Comment the existing command line with a `#` character.
3. Add the following lines to invoke **gdb**:

```
my $cmd="$TARGETDIR\\cmd\\run.cmd";
SystemP("gdb -se=$exe -command=$cmd > stdout.log");
```

4. Right-click **Build Settings** and select **Ascii File**. Rename the created file to `run.cmd`.
5. Copy the contents of the `run_example.cmd` file, provided in the **example** directory, into the `run.cmd` file.
6. Save the TDP, update the TDP in the project and Build  the **add.c** node.

Converting Data to ASCII


Depending on the cross-development environment, the format of the dumped data can vary largely from one target to another. In most cases, the results must be decoded and converted to ASCII data to be processed by the **Test Embedded** Data Splitter and Report Generators.

You need to decode the dump data to ASCII with a Perl routine by using the Perl subroutine named **decode.pl**.

Perform the following steps to decode dump data to ASCII:

1. Save the TDP, update the Configuration in **Test Embedded** and Build the  the **add.c** node.

Test Embedded returns an error: the dump accomplished by the debugger does not produce a plain ASCII file as expected.

2. View the result file created by GDB. The relevant data is present but is represented in hexadecimal format and mixed with other information.
3. Open the **decode.pl** Perl script in a text editor, provided with the example.
4. In the **TDP Editor**, expand **Build Settings** and select **Execution function** to locate the **atl_exec** Perl function.
5. Copy-paste the contents of the **decode.pl** Perl script into the **atl_exec** Perl function after execution of **gdb**.
6. Save the TDP, update the TDP in the project and Build  the **add.c** node.

Notices

This document provides information about copyright, trademarks, terms and conditions for the product documentation.

© Copyright IBM Corporation 2000, 2016 / © Copyright HCL Technologies Limited 2016, 2021

This information was developed for products and services offered in the US.

HCL® may not offer the products, services, or features discussed in this document in other countries. Consult your local HCL® representative for information on the products and services currently available in your area. Any reference to an HCL® product, program, or service is not intended to state or imply that only that HCL® product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any HCL® intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-HCL® product, program, or service.

HCL® may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

For license inquiries regarding double-byte character set (DBCS) information, contact the HCL® Intellectual Property Department in your country or send inquiries, in writing, to:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

HCL TECHNOLOGIES LTD. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. HCL® may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-HCL® websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this HCL® product and use of those websites is at your own risk.

HCL® may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

HCL

330 Potrero Ave.

Sunnyvale, CA 94085

USA

Attention: Office of the General Counsel

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by HCL® under terms of the HCL® Customer Agreement, HCL® International Program License Agreement or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-HCL® products was obtained from the suppliers of those products, their published announcements or other publicly available sources. HCL® has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-HCL® products. Questions on the capabilities of non-HCL® products should be addressed to the suppliers of those products.

Statements regarding HCL®'s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to HCL®, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. HCL®, therefore, cannot guarantee or imply reliability,

serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. HCL® shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a copyright notice as follows:

© (your company name) (year).

Portions of this code are derived from HCL Ltd. Sample Programs.

© Copyright HCL Ltd. 2000, 2022.

Trademarks

HCL®, the HCL® logo, and ibm.com® are trademarks or registered trademarks of HCL Technologies Ltd., registered in many jurisdictions worldwide. Other product and service names might be trademarks of HCL® or other companies.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the HCL® website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of HCL®.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of HCL®.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

HCL® reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by HCL®, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

HCL® MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.