IBM WebSphere Transformation Extender

Platform API

Version 8.1

Note
Before using this information, be sure to read the general information in "Notices" on page 41.

October 2006

This edition of this document applies to WebSphere Transformation Extender, 8.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, e-mail DTX_doc_feedback@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. Platform API overview

The Platform API provides tight integration of the Command Server or Launcher into user applications, including those created using development tools such as Visual Basic. The Platform API is based on the command options available for the particular development platform. These options, explained in the *Execution Commands* documentation, provide flexibility in how your maps are executed. For example, you can pass data to be mapped to the API or receive mapped data from the API without using files. The Platform API includes platform-specific database connectivity. Compile and link commands are platform and operating system specific.

Installed files

The following files are included in the WebSphere Transformation Extender, which includes the Platform API for Win32 and UNIX. For z/OS information, see ″Using the API on z/OS″.

File **Description**

librun.*
: Platform API libraries (UNIX)

runmer32.dll
mercma32.dll
mercad32.dll
mercrm32.dll
dtxpi.dll
mresname.dll
: Platform API libraries (Win32)

testapi.c
: Sample 'C' program showing use of the Platform API to execute the **sinkmap** map example

runmerc.h
: Platform API header file

readme.txt
: Readme file containing instructions for compiling and running the **testapi.c** file

testapid.c
: Sample 'C' program illustrating how to dynamically load the Platform API and how to set the exit routine and unload in the Platform API in the exit routine (UNIX only)

  This sample `C' program is for the Sun operating system. The actual implementation on other systems may be slightly different, but the concept is the same.

Platform API examples

The examples provided with the WebSphere Transformation Extender Software Development Kit are located at *install_dir*/examples/dk.

Chapter 2. Using the Platform API

You can use the Platform API functions and structures to integrate mapping processes within your application.

The Platform API is a C library.

Platform API usage guidelines

The guidelines for using the Platform API describe retaining java objects, using Platform API, and the RUN command and command options.

Retaining java objects

To retain java objects after a given map instance ends, you must enable an option in the **dtx.ini** initialization file listed under **[JVM Options]**:

```
option1=-Dmpi.map.destroy_object_pool=false
```

Since the java objects will not be released, the Platform API application is responsible for freeing any existing java objects.

For information about JVM Options, see the *Type Tree Importers* documentation.

Using Platform API

The following procedure assumes that you have a previously created map or system.

To use the Platform API

1. From within your application, if your map uses an adapter such as a database, messaging or utility, call `MercInitAPI()` function to establish adapter connections before calling any Platform API functions.
2. Initialize the `ExitParam` structure using a `memset()`call for the `ExitParam` structure, which will subsequently be used as an argument for the `RunMap` function.

   When performing a `memcpy()`, be sure that the map name is null - terminated by adding a terminating null of `\0` to the line.
3. Create a standard command line within your application.

   For example, to build a command line for use with the `RunMap` function:
   - Assign `lpDataToApp` to point to the command line.
   - Execute the `RunMap` function.
4. If you called `MercInitAPI()`to establish connections for an adapter in step 1 above, call `MercExitAPI()`after calling Platform API functions to close the adapter connections and perform necessary API clean-up.

In the simplest situation, to run **sinkmap.mmc**, your program may look like the following:

```
#include "windows.h"
#include <stdio.h>
#include "runmerc.h"
     int main(void)
```

```
    {
    EXITPARAM   ExitParam;

    MercInitAPI();

    memset(&ExitParam,0,sizeof(EXITPARAM));

    ExitParam.lpDataToApp = (unsigned char *)"sinkmap.mmc -ae -ts -p32:8\0";

    ExitParam.dwSize = sizeof(EXITPARAM);
    ExitParam.lpv = NULL;

    RunMap(&ExitParam);

    printf("Map returned %d:  %s\n", ExitParam.nReturn, ExitParam.szErrMsg);

    if (ExitParam.dwfromLen)
          free (ExitParam.lpDataFromApp);

    MercExitAPI();

    return 0;
}
```

When using the Platform API on Windows, do not call the DLL from DLLMAIN.

The Run command and command options

To execute a map from within the Platform API, specify a standard run command as an argument to the `RunMap` function.

A run command is defined as:

mapset[*mapset*](s)]|[*@command-file-name*](s)

where *mapset* is defined as:

map-name[*command-option-list*]

and *command-option-list* is defined as:

command-option[*command-option*(s)]

The content of a command-file is defined as:

mapset[*mapset*] (s)

where *map-name* specifies the compiled map file (**.mmc**) to be executed.

An example of a simple run command might be the name of the compiled map and related command options:

```
sinkmap.mmc -ae -ts -p32:8
```

A run command might consist of more than one map file to be executed, such as:

```
xlatappdata.mmc -ae -if1 mydata.txt updmaster.mmc -ts -wm
```

Alternatively, the run command might reference a command file that contains one or more maps with associated command options, such as:

```
@processorders.cmd
```

For a complete reference of command options, see the *Execution Commands* documentation.

Whose memory is It anyway?

When you call RunMap from your application, it is your responsibility to allocate and release any memory associated with lpDataToApp. lpDataFromApp is allocated by the Platform API, but you must release it. With the dynamic link library (DLL) versions of the Platform API, release memory using the macro **GlobalFreePtr**, defined in **windowsx.h** of the Microsoft header file, which is included with the 32-bit Windows APIs. With non-DLL versions of the Platform API, use the C runtime **free** call.

See ″Return Codes and Error Messages″ for a list of the error codes that result when an invalid command line is specified in lpDataToApp for the Platform API.

Data to and from the Platform API

Use the Input Source Override - Echo execution command (-IE) for a source or target to pass data as input from your application to a map or to receive data as output of a map without using files or applications.

Passing input data to a map

For example, if your application has a pointer to data you want to use as a source to the map **mymap.mmc**, the following code fragment shows one way of doing this:

```
int main(void)
{
EXITPARAM            ExitParam;
LPBYTE                  lpBuf;
DWORD                  dwLen;
char            szbuf[50];
int            nlen;
/* do user stuff to get buffer and length of buffer to map /
.
.
.
/* dwLen is length of input data */
/* lpBuf is the buffer that contains the input data */
/* -ie1 means "echo" in input for source number 1 */
/* in this example, we will use the size option with the -ie option */
/* to tell the API the size of the input */
sprintf(szbuf, "mymapp.mmc -ie1s%d ", dwLen);
nlen = strlen(szbuf);
/* allocate storage space for lpDataToApp */
ExitParam.lpDataToApp = malloc(dwLen + 100);
ExitParam.dwSize = sizeof(EXITPARAM);
ExitParam.lpv = NULL;
/* assign it to lpDataToApp */
memcpy(ExitParam.lpDataToApp, szbuf, nlen);
memcpy(&ExitParam.lpDataToApp[nlen], lpBuf, dwLen);
/* call the function to map it */
RunMap(&ExitParam);
/* lpDataFromApp contains the string representation of nReturn */
.
.
.
return 0;
}
```

Returning data from a map

When you do not override any of the map's targets with the Output Target Override - Echo execution command (-OE), lpDataFromApp contains the string representation of nReturn. For example, if nReturn is 0, lpDataFromApp is the string 0.

If you override an output by using the Output Target Override - Echo execution command (-OE), lpDataFromApp contains the data for the specified output card. Using -OE for more than one output concatenates the data for the outputs with no separator.

Using the previous example, if you append -oe1 to the lpDataToApp, then lpDataFromApp contains the data that was mapped to the first output rather than the string representation of nReturn. The length is in the dwFromLen component of the EXITPARAM structure as you can see in the following code example:

```
int main(void)
{
EXITPARAM            ExitParam;
LPBYTE                  lpBuf;
DWORD                  dwLen;
char            szbuf[50];
int            nlen;
/* do user stuff to get buffer and length of buffer to map */
.
.
.
/* dwLen is length of input data */
/* lpBuf is the buffer that contains the input data */
/* create the command line */
/* -ie1 means "echo" in input for source number 1 */
/* in this example, we will use the size option with the -ie option */
/* to tell the API the size of the input */
sprintf(szbuf, "mymapp.mmc -ie1s%d ", dwLen);
nlen = strlen(szbuf);
/* allocate storage space for lpDataToApp */
ExitParam.lpDataToApp = malloc(dwLen + 100);
ExitParam.dwSize = sizeof(EXITPARAM);
ExitParam.lpv = NULL;
/* assign it to lpDataToApp */
memcpy(ExitParam.lpDataToApp, szbuf, nlen);
memcpy(&ExitParam.lpDataToApp[nlen], lpBuf, dwLen);
/* now add in the -oe1 option to get the data that was mapped back */
memcpy(&ExitParam.lpDataToApp[nLen + dwLen], " -oe1", 5);
/* call the function to map it */
RunMap(&ExitParam);
/* lpDataFromApp now contains the output from card 1 that was mapped */
/* dwFromLen is the size of the output */
.
.
.
return 0;
}
```

Using maps in memory

The Platform API has the ability to load one or more maps into memory for subsequent execution. The advantage of using the Platform API is that the map only loads into memory once, rather than once for each execution.

The three functions Platform API that can load the map into memory are:

- InitializeRunMapInstance
- FreeRunMapInstance
- RunMapUsesInstance

An example of using maps in memory

To execute maps in memory using RunMap:

1. Initialize and set up your EXITPARAM structure.
2. Initialize each map to be executed from memory using the InitializeRunMapInstance function.
3. Associate the RunMapInstance, returned by the InitializeRunMapInstance function, with its corresponding EXITPARAM structure by calling RunMapUsesInstance with bSet equal to TRUE.
4. Set up the information in the EXITPARAM structure. Set up lpDataToApp and call RunMap to execute the map.
5. When all instances of the RunMapInstance are complete, free the memory associated with the map instance by calling FreeRunMapInstance, and clear the EXITPARAM structure from knowledge of the map instance by calling RunMapUsesInstance with bSet equal to FALSE.

Initially, if the program has an EXITPARAM variable named ep, to call RunMap the program would have set:

```
ep.lpv = NULL
ep.dwSize = sizeof(EXITPARAM);
```

The following sample code uses the RunMap function to call a map (**sinkmap.mmc**) that has been initialized into memory.

In the following code, lpv is an LPVOID type set within the user's program.

```
/* declare a new local variable: */
LPVOID        lpv;
/* to initialize, do these two steps */
/* this gets a pointer to a map instance: */
lpv = InitializeRunMapInstance("sinkmap.mmc");
/* this assigns the map instance to the ep structure: */
RunMapUsesInstance(&ep, lpv, TRUE);
/* now do the normal init stuff (setting up lpDataToApp) */
ep.lpDataToApp = "sinkmap.mmc -ae";
/* and call RunMap */
RunMap(&ep);
/* and to clear the maps in memory, these two steps: */
/* this frees the memory associated with the map instance: */
FreeRunMapInstance(lpv);
/* this clears the ep structure from knowledge of the map instance: */
RunMapUsesInstance(&ep, NULL, FALSE);
```

Loading multiple maps

You can preload more than one map. The RunMap function looks for the EXITPARAM structure to determine whether the structure has been set with RunMapUsesInstance call.

Be sure to set up the call to the RunMap function with the same map name (including path) as when InitializeRunMap was called.

DTX_DO_NOT_CHDIR for Windows

During run time, by default, the API changes to the directory where the compiled map is located. To disable this feature, set the DTX_DO_NOT_CHDIR environment variable to **TRUE**. For example:

```
DTX_DO_NOT_CHDIR=TRUE
```

Use caution when enabling the DTX_DO_NOT_CHDIR environment variable. Any references to relative paths in the mapping may not remain relative to the compiled map as required.

For detailed information, see ″Do Not Change Directory (DTX_DO_NOT_CHDIR)″.

Chapter 3. Using Platform API in a UNIX environment

Environment variables can be set for the Platform API to define the location for temporary files, to produce environmental debugging information, and to control file-locking facilities. These features are available for all UNIX systems on which the Platform API is available.

The HP Non-stop ZLE platform does not use dynamic-link libraries. Any part of this documentation that describes shared libraries and shared library path (LIBPATH) variables does not apply to this platform.

Dynamic load/unload of the API (advanced users only)

If you are using dynamic load/unload of the Platform API, the unload operation must be moved to an exit handler. See the sample program **testapid.c**, found in the default installation directory in the **Examples\Platform API** folder, for an example. The unload operation must be in the exit handler because the Platform API also uses exit handlers to free resources. If your program unloads the Platform API before issuing the exit or return) API, results can be unpredictable.

The **testapid.c** sample is for Sun Microsystems platforms only. The actual operation on other systems may be slightly different, but the concept is the same.

In the sample program **testapid.c**:

- `atexit(myexit)`sets up the exit handler in the beginning of the program.
- `dlopen` opens the Platform API shared library (**libplatapi.so**).
- `dlsym` obtains the address of the `RunMap` routine within the Platform API shared library and this routine is run.
- When the return `ExitParam.nReturn` is run, the `myexit` function issues `dlclose` to close the Platform API shared library.

Configuring the UNIX environment

The following documentation describes the guidelines for configuring the Platform API in a UNIX environment.

The HP Non-stop ZLE platform does not use dynamic link libraries. Any sections in this documentation that describe shared libraries and shared library path (LIBPATH) variables do not apply to this platform.

Environment variable setup program

Before executing a map, execute the **setup** program in the Transformation Extender installation directory. This will set the required environment variables for executing (PATH, Shared Library Path, DTX_TMP_DIR, DTX_HOME_DIR).

The following procedure assumes that your UNIX command line environment is the Korn (ksh) or Bourne (sh) shell. There must be a space between the period (`.`) and the command path. The period does not work with the C shell (csh).

Execute the command as follows:

```
. /install_directory/setup
```

Environmental debug information (DTX_DEBUG)

If a problem occurs while executing a map with the Command Server or Platform API on UNIX, environmental debug information is produced that helps determine the cause of the problem.

By default, the environmental debug facility is disabled if the DTX_DEBUG environment variable is not defined.

Enabling environmental debug

To enable this environmental debug or trace facility, set the DTX_DEBUG environment variable to TRUE. For example:

```
DTX_DEBUG=TRUE
export DTX_DEBUG
```

This environmental debug information is written to a file named **mercinfo.log** in the directory identified by the DTX_TMP_DIR environment variable. If the DTX_TMP_DIR environment variable is not set, mercinfo.log is written to /tmp. The **setup** program sets DTX_TMP_DIR to /install_directory/tmp.

The environmental debug file can be used to provide additional information for troubleshooting problems with a specific map. The following is a sample of the information contained in the environmental debug file:

```
PROCESS_ID: 27435, API_REF: 1
   Date/Time: Tue Mar 28 08:17:49.566911 2000
FILE: mercmain.c, line: 360
  info: [DTX Product Version: (561)]
PROCESS_ID: 27435, API_REF: 1
   Date/Time: Tue Mar 28 08:17:49.571748 2000
FILE: mercmain.c, line: 388
  info: [DTX RUNNING: Tue Mar 21 08:17:49 2000]
PROCESS_ID: 27435, API_REF: 1
   Date/Time: Tue Mar 21 08:17:49.583996 2000
FILE: mercrun.c, line: 1320
  New Map File - Tue Mar 28 08:17:49 2000 [sinkmap.mmc]
PROCESS_ID: 27435, API_REF: 1
   Date/Time: Tue Mar 28 08:18:05.109727 2000
FILE: mercmain.c, line: 645
  info: [Info - main(): Map Completed Successfully.]
```

Disabling environmental debug

To disable the environmental debug facility, set the DTX_DEBUG environment variable to FALSE. For example:

```
DTX_DEBUG=FALSE
export DTX_DEBUG
```

File locking (DTX_FILE_LOCKING)

The file locking facilities allow two or more mapping processes or threads to write to the same output files in a controlled manner. These files include not only output files, but also map work files, audit log files and trace files.

File locking is enabled by default. The UNIX Command Server and Platform API both use file locking. See ″Enabling File Locking″ and ″Disabling File Locking″ for more information.

During file locking, information should be written to a temporary file named **.mercShm** in the directory identified by the DTX_TMP_DIR environment variable. If the DTX_TMP_DIR environment variable is not set, this file will be written to **/tmp**.

Additional files created that are related to file locking and shared memory include:

File Name
Description

.mercPid
Contains process ID information for the mapping processes using shared memory

.mercOldShm
Contains old shared memory information

.mercPid#
A series of files containing process ID information for entries in the **.mercOldShm** file. The **#** is replaced by a number that references a corresponding line in the **.mercOldShm** file.

These files should *not* be deleted for any reason.

Do not change directory (DTX_DO_NOT_CHDIR)

The do not change directory (DTX_DO_NOT_CHDIR) facilities control the use of the change current directory (chdir) function. This means that the PAPI code will not use the chdir function to change the directory.

By default, the do not change directory facility is disabled if the DTX_DO_NOT_CHDIR environment variable is not defined.

Enabling Do Not Change Directory

To enable the do not change directory facility, set the DTX_DO_NOT_CHDIR environment variable to **TRUE**. For example:

```
DTX_DO_NOT_CHDIR=TRUE
export DTX_DO_NOT_CHDIR
```

The DTX_DO_NOT_CHDIR variable can be used only in cases of absolute path names. If you use relative path names and DTX_DO_NOT_CHDIR=TRUE, results may not be as expected.

Disabling Do Not Change Directory

To disable the environmental debug facility, set the DTX_DO_NOT_CHDIR environment variable to **FALSE**. For example:

```
DTX_DO_NOT_CHDIR=FALSE
export DTX_DO_NOT_CHDIR
```

Configuring the map execution environment

Because file locking applies to all files written to during map execution, certain map and adapter settings should be used to set up the execution environment to support this. Examples of this include:

- A map's output files can be managed using the **AdapterRetry** setting for a target. You can also specify this by using the Rc:i parameter within the -OFx execution command.

- The map's work files, audit logs and trace files can be managed using the map setting. You can also specify this by using the `-Yc:i` execution command.
- Another option for work files and audit logs is to use the **Unique FilePrefix** option for the **MapAudit** and **WorkSpace** map settings. You can also do this using the `-WU` and `-AU` execution commands.

  Because trace files do not have an option for using unique file names, they can only be managed by using the **MapRetry** map setting, or the `-Yc:i` execution command.

See the *Execution Commands* documentation and the *Map Designer* documentation for detailed information about these options.

Enabling file locking

File locking is enabled by default. Setting the environmental debug facility is optional to the file locking facility. To enable the environmental debug facility, set the `DTX_DEBUG` environment variable to `TRUE`. To enable the file locking facility, set the `DTX_FILE_LOCKING` environment variable to `TRUE`. For example:

```
DTX_FILE_LOCKING =TRUE
export DTX_FILE_LOCKING
```

To use this file locking facility, the `DTX_TMP_DIR` must be set to a common directory for all situations in which maps are using common output files.

Disabling file locking

To disable the file locking facility, set the `DTX_FILE_LOCKING` environment variable to `FALSE`. For example:

```
DTX_FILE_LOCKING=FALSE
export DTX_FILE_LOCKING
```

If you do not want file locking and the `DTX_FILE_LOCKING` environment variable is not defined, the `DTX_TMP_DIR` must be set to a unique directory path for each mapping process.

Using the Platform API with database adapters

To use the Platform API with database adapters, you must install the database adapters and set the shared object path environment variable. However, even if you are not using database adapters, this environment variable must be set to access the shared libraries.

The RS/6000 AIX platform stores in cache shared libraries. So, if you update a shared library on the disk, you will not be able to see the update. Use the slibclean command at user root to remove the old shared library from system memory.

Chapter 4. Using Platform API on z/OS

WebSphere Transformation Extender supports two execution environments on z/OS: batch and CICS. The command server execution environment for CICS provides a programming interface that uses the CICS Command Level Program Control APIs.

This interface enables COBOL programs to call the WebSphere Transformation Extender server on CICS in a fashion similar to the `RunMap` function described in this documentation. Information about the WebSphere Transformation Extender CICS programming interface is located in the *Command Server* documentation. The Platform API documentation focuses on the z/OS batch Platform API.

Overview

The Platform API for z/OS, like the other platforms, is a set of function calls that allow a high-level language program to programmatically call the WebSphere Transformation Extender execution environment. The high-level languages supported on z/OS are C/C++ and COBOL.

The Platform API is available as part of the Software Development Kit (SDK) runtime execution environment included with the WebSphere Transformation Extender for z/OS. The SDK contains the Platform API runtime and example programs for 'C' and COBOL.

Requirements

The use of the Platform API on z/OS requires the use of a Language Environment (LE) enabled C/C++ or COBOL compiler. The Platform API will work with all supported releases of the LE run time.

Calling the Platform API from COBOL

The WebSphere Transformation Extender provides a way to call platform API from COBOL by calling the Platform API functions as a DLL.

This requires an IBM Host COBOL compiler that provides COBOL DLL support. IBM COBOL for z/OS and VM 2.1 is the earliest host compiler that supports COBOL DLLs.

Calling the Platform API from a COBOL DLL

From a programming perspective, calling a DLL from COBOL is not much different from a COBOL dynamic CALL. The major difference is that the DLL name can be up to 160 characters versus eight (8) for the traditional COBOL call statement. There are some restrictions, such as the program must be reentrant.

The following COBOL compiler options were used to compile the DTXTCCOB example program, located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation:

```
CBL RENT,DLL,NOEXPORTALL,MAP,LIST,LIB,PGMNAME(M),OPT(STD)
```

For a description of these options, consult the IBM COBOL documentation for z/OS.

Example files for z/OS

A set of examples is installed with the Software Development Kit included in the WebSphere Transformation Extender installation that demonstrates how to call the Platform API. Specific to the Platform API are three example programs and an example map. There are readme files for each of the examples, which describe how to install and use them. Compiled versions of these programs are included in the DTX.SDTXLOAD load library and can be run without building them.

SINKMAP example MAP

The SINKMAP example map is the default map used by the three examples.

The map, input and JCL files are located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation.

The example programs can use a different map by adding or changing the JCL job parameter statement for the program and referencing the new map along with any command line options.

Platform API `C' example

The DTXTCCOB 'C' example is a variation of the testapid.c example described in "Dynamic Load/Unload of the API (Advanced Users only)". The source and JCL for this example is located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation.

The major difference between the DTXTCCOB z/OS version of testapid.c and the other platforms is that DTXTCCOB calls the SINKMAP example map using command line options that are unique to z/OS. To learn more about these command line options see the *Command Server* documentation.

Platform API COBOL examples

The DTXTCCOB COBOL example program demonstrates how to call all of the Platform API functions from a COBOL program compiled as a DLL. The source and JCL files for this example are located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation.

Passing function parameters

To call platform API functions as a DLL, you will set up function parameters to pass to the platform API.

The `RunMap` function takes only one parameter, the `ExitParam` structure address. When using the `RunMap` function, the pointer to the map file name and the pointer to the `DataFromApp` are contained in the `ExitParam` structure. `CALLRMAP` was designed to take the extra parameters as a way to minimize the need to manipulate and handle pointers. A major deficiency in COBOL is that the address of a working storage item cannot be assigned to a pointer. The item needs to be in the linkage section.

Setting the address of the map file name requires that either the map file name be passed through the `JCL PARM` statement (linkage section) or a sub-program be created that takes the address of the map file name as one parameter and passes the address back, in the form of a pointer, in another parameter.

Cobol copybooks

There are two copybooks included with the examples. Combined they make up the COBOL definition of Platform API structures. They are:

Name Description

DTXTCCPY
This copybook contains the COBOL definition for both the EXITPARAM and EXEOPTS structures.

DTXCICPY
This copybook contains the COBOL definition for the CARDINFO structure.

The two copybooks are required because the DTXTCCPY copybook needs to be in the COBOL program's LINKAGE-SECTION and the DTXCICPY copybook needs to reside in the WORKING-STORAGE section.

For content descriptions of these copybooks, see the Platform API Structure Definitions" in the Platform API documentation.

The variable names contained in the copybooks are the same as the component names documented in Platform API Structure Definitions", except that the COBOL names are all in uppercase. In addition, unlike C, a COBOL program cannot contain variables with the same name. Therefore, any variable name containing a "-" is already defined to the DTXTCCPY copybook.

The COBOL copybooks are located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation.

PLATAPID DLL import definition

The DTXTAEXP is a file that contains the DLL import statements for the Platform API functions. The file is required for the LE prelinker step of any COBOL or C/C++ compile job that wants to link in the support for the Platform API.

If you are using your own compile and link JCL, add the DTXTAEXP PDS member to the SYSIN DD definition of the LE prelinker step of the compile and link JCL.

The DTXTAEXP file is located in the DTX.SDTXSAMP PDS included in the WebSphere Transformation Extender installation.

Chapter 5. Platform API functions

The following list of functions make up the Platform API and provides a brief description of each function's purpose.

Function
Description

Adapter Connection and Cleanup

MercInitAPI()
Establishes connections for WebSphere Transformation Extender adapters, including database, messaging and utility.

MercExitAPI()
Closes the adapter connections established by MercInitAPI()and performs necessary API clean-up

Executing a Map

RunMap Executes the maps

Executing a Map in Memory

InitializeRunMapInstance
Initializes a specific map file into memory for one or more subsequent executions

RunMapUsesInstance
Sets or clears the EXITPARAM structure passed into the RunMap function for maps that have been initialized into memory using the InitializeRunMapInstance function

FreeRunMapInstance
Frees all memory associated with a map that has been initialized using the InitializeRunMapInstance function

Examining I/O Cards

GetCardIOs
Obtains card and adapter settings for sources and targets for a particular map file

MercInitAPI() function

If your map uses an WebSphere Transformation Extender adapter, such as a database, messaging or utility, call the MercInitAPI() function to establish adapter connections before calling any Platform API functions. MercInitAPI() is not required if you are not using adapter connections.

MercInitAPI() and MercExitAPI() are used together. Call MercInitAPI() once at the start of your application and then call a corresponding MercExitAPI() at the end. Successive calls to MercInitAPI() are unnecessary.

Syntax

Platform
Syntax

C

```
int MercInitAPI();
```

z/OS COBOL

```
CALL 'MercInitAPI' RETURNING RETCODE.
```

Returns

Returns 0 if initialization is successful.

See Also

"MercExitAPI() Function"

If `MercInitAPI()` and `MercExitAPI()` are not used, each invocation of the `RunMap` function establishes and closes adapter connections within the scope of the map. If `MercInitAPI()` and `MercExitAPI()` are used, connections can be shared across maps.

MercExitAPI() function

Use the `MercExitAPI()` function to close the adapter connections established by the `MercInitAPI()` function and perform necessary API clean-up actions.

Syntax

Platform

Syntax

C

```
void MercExitAPI();
```

z/OS COBOL

```
CALL 'MercExitAPI'.
```

Returns

There is no return value.

For more information

- "MercInitAPI() Function"

RunMap function

Use the `RunMap` function to execute the map using the map and card settings compiled into the map, as well as any overrides specified as part of the `lpDataToApp` member of the `EXITPARAM` structure.

Syntax

Platform

Syntax

C

```
void RunMap (LPEXITPARAM lpExitParam);
```

z/OS COBOL

```
CALL 'RunMap' USING BY REFERENCE EXITPARAM.
```

Parameters

RunMap function parameters:

Parameter	Parameter Description
lpExitParam	A pointer to an EXITPARAM structure. This structure contains the input and output parameters used when running the map.

CALLRMAP function parameters:

Parameter	Parameter Description
EXITPARAM	This structure contains the input and output parameters used when running the map.
MapFile	The name of the compiled map file and options to run.
DataFromApp	Data that is returned from the map execution.

Returns

There is no return value.

However, the members of the EXITPARAM structure contain the following data based on the map's execution:

EXITPARAM structure members
Returned data

nReturn
map return code

szErrMsg
map return message

lpDataFromApp
any data returned by the map as a result of echoed outputs

When calling RunMap, the API fills in the value of nReturn based on the map's execution. If multiple maps are on the command line, nReturn is based on the last map on the command line. See "Return Codes and Error Messages" for the values of nReturn and the associated messages in szErrMsg.

InitializeRunMapInstance function

Use the InitializeRunMapInstance function to initialize a specific map file into memory for one or more subsequent executions.

Syntax

Platform
Syntax

C
```
LPVOID CALLBACK InitializeRunMapInstance(LPSTR lpszMapFile);
```

z/OS COBOL

```
CALL ' InitializeRunMapInstance ' USING BY
REFERENCE lpszMapFile,
RETURNING MapInstPtr.
```

Parameters

Following are parameters for the InitializeRunMapInstance function.

Parameter	Parameter Description
lpszMapFile	The name of the compiled map file to initialize and load into memory.

Returns

If successful, InitializeRunMapInstance returns a pointer to be used for a subsequent call to RunMapInstance. A NULL return signifies a failure, most likely meaning the specified map does not exist.

For more information

- "FreeRunMapInstance Function"
- "RunMapUsesInstance Function"

RunMapUsesInstance function

Use the RunMapUsesInstance function to set or clear the EXITPARAM structure passed into the RunMap function for maps that have been initialized into memory using the function InitializeRunMapInstance.

Syntax

Platform	Syntax

C	`void CALLBACK RunMapUsesInstance( LPEXITPARAM lpep,`
	`LPVOID lpv,`
	`BOOL bSet);`

z/OS COBOL

```
CALL ' RunMapUsesInstance' USING BY REFERENCE EXITPARAM,
BY VALUE lpv,
bSet.
```

Parameters

Following are parameters for the RunMapUsesInstance function.

Parameter	Parameter Description
lpep	The address of the EXITPARAM structure to be used in a RunMap call
lpv	The return of a call to the InitializeRunMapInstance function
bSet	Indicator of whether to set (TRUE) or clear (FALSE) the EXITPARAM structure specified by lpep

If the RunMapUsesInstance function is being used to clear the EXITPARAM structure (bSet is FALSE), the lpv parameter can be NULL.

Returns

There is no return value.

For more information

- "InitializeRunMapInstance Function"
- "FreeRunMapInstance Function"

FreeRunMapInstance function

Use the FreeRunMapInstance function to free all the memory associated with a map initialized using the InitializeRunMapInstance function. The FreeRunMapInstance function is used with the RunMapUsesInstance function when maps in memory are no longer required.

Syntax

Platform
Syntax

C
```
BOOL CALLBACK FreeRunMapInstance(LPVOID lpRunMapInstance);
```

z/OS COBOL
```
CALL 'FreeRunMapInstance' USING BY VALUE lpRunMapInstance.
```

Parameters

Following are parameters for the FreeRunMapInstance function.

Parameter
Parameter Description

lpRunMapInstance
The pointer returned by the InitializeRunMapInstance function for a map loaded into memory.

Returns

The return value is always TRUE.

For more information

- "InitializeRunMapInstance Function"
- "RunMapUsesInstance Function"

GetCardIOs function

Use the GetCardIOs function to obtain card and adapter settings for sources and targets for a particular map file.

Syntax

Platform
Syntax

C

```
BOOL CALLBACK GetCardIOs( LPSTR lpszMapFile,

 LPVOID FAR * lpci,

 LPWORD lpCardCount,

 LPVOID lpExeOpts,

 LPWORD lpwRC);
```

z/OS COBOL

```
CALL 'GetCardIOs' USING BY REFERENCE lpszMapFile,
lpci,
lpCardCount,
lpExeOpts,
RETURNING BOOL.
```

Parameters

Following are parameters for the `GetCardIOs` function.

Parameter
Parameter Description

`lpszMapFile`
The map file for which to get card information. This can include an absolute path or a relative path from where the application is loaded.

`lpci` A pointer to a void that is converted to a pointer for a `CARDINFO` structure.

`lpCardCount`
A pointer to a word that will receive a number representing the total number of cards, both input and output, in the map.

`lpExeOpts`
A pointer to a void that is converted to a pointer for an `EXEOPTS` structure. This structure contains the map settings that are compiled into the map. This parameter may be NULL, in which case the map settings information will not be returned.

`lpwRC` A pointer to the return code of the map. If successful, this will be `0`. A non-zero value indicates an error.

Returns

If successful, `GetCardIOs` returns `TRUE` and `lpci` contains a pointer to an array of `CARDINFO` structures with `lpCardCount` receiving the count of cards. Otherwise, `GetCardIOs` returns `FALSE`.

When `GetCardIOs` returns `FALSE`, it might indicate an incompatible version of the map file with the release of the Platform API. This usually means the compiled map file is invalid or needs to be recompiled.

If the return value of `GetCardIOs` is `TRUE`, it is the calling function's responsibility to release the memory associated with `lpci`. The specific call to release the memory depends on the platform on which the API resides. See the **readme.txt** file that accompanies the specific API.

Chapter 6. Platform API structure definitions

EXITPARAM structure

The EXITPARAM structure is used for the input and output parameters to the RunMap function.

```
struct tagExitParamStruct
{
   DWORD         dwSize;
   DWORD         dwToLen;
   DWORD         dwFromLen;
   DWORD         dwMapInstance;
   void FAR *    lpv;
   LPSTR         lpszCmdLine;
   BYTE HUGE *   lpDataToApp;
   BYTE HUGE *   lpDataFromApp;
   UINT          uRetryCount;
   UINT          uRetryInterval;
   BOOL          bRollback;
   BOOL          bCleanup;
   int           nReturn;
   char          szErrMsg [100];
   char          szFile[260];
   void FAR *    lpMapHandle;
   void FAR *    lpInternal;
   void FAR *    lpCmdStruct;
   void FAR *    lpAdaptParms;
   void FAR *    lpContext;
   void FAR *    lpWildcard;
   void FAR *    lpfnMS;
   void FAR *    lpMS;
   DWORD         dwWildcardSize;
   LPSTR         lpszMapDirectory;
   WORD          wCardNum;
   WORD          wCleanupAction;
   WORD          wScope;
   UINT          uUnitSize;
   BOOL          bBurst;
   BOOL          bFromRule;
   BOOL          bSource;
   DWORD         dwRecords;
};
typedef struct tagExitParamStruct EXITPARAM;
typedef struct tagExitParamStruct FAR * LPEXITPARAM;
```

EXITPARAM components and usage

The EXITPARAM structure has the following components that are used by the RunMap:

Some components of EXITPARAM are not used with the Platform API because this general structure is also used elsewhere to provide a common interface method.

Component	Used As	Use
dwSize	Input	The size (in bytes) of the EXITPARAM structure to ensure correct version compatibility
dwToLen		Not used

Component	Used As	Use
dwFromLen	Output	The length (size in bytes) of lpDataFromApp
dwMapInstance[1]		Must be set to a unique number for each map that is executed
lpv		Not used and should be NULL
lpszCmdLine		Not used
lpDataToApp	Input	The command line being passed in
lpDataFromApp	Output	The result based on command options
uRetryCount		Not used
uRetryInterval		Not used
bRollback		Not used
bCleanup		Not used
nReturn	Output	Return code based on the last processed map
szErrMsg	Output	String message based on nReturn
szFile		Not used
lpInternal		Not used
lpCmdStruct		Not used
lpAdaptParms		Not used
lpContext		Not used
lpWildcard		Not used
dwWildcardSize		Not used
lpszMapDirectory		Not used
wCardNum		Not used
wCleanupAction		Not used
wScope		Not used
uUnitSize		Not used
bBurst		Not used
lpfnMS		Not used
lpMS		Not used
dwRecords		Not used

[1]The dwMapInstance component must be set to a unique number for each map that executes. Failure to do so will result in unpredictable transactional behavior.

CARDINFO structure

The CARDINFO structure is used to provide information about a map file's input and output cards.

```
struct tagCardInfo   {
    WORD wCard;
    WORD wIOType;
    BYTE byFlag;
    char szCardName[MAX_TYPENAME + 1];
    char szFileName[_MAX_PATH + 1];
```

```
BYTE byUnused[21];
BOOL bIntegralMode;
BOOL bSourceEvent;
BOOL bBackupSwitch;
BOOL bBackupAlways;
BOOL bMapDirectory;
BOOL bAppendBackup;
BOOL bRollBackOnFailure;
BOOL bRetrySwitch;
BOOL bIgnoreAdapterWarnings;
WORD wMapSuccessAction;
WORD wScope;
WORD wRetryAttempts;
WORD wRetryInterval;
WORD wFetchUnit;
WORD wUnused[9];
char szBackupFile[256];
};
```

CARDINFO components

The CARDINFO structure has the following components.

Part **Description**

wCard The value of the card number in the map. If a map has two inputs and two outputs, wCard will have the following values:

Card number
Value

first input
1

second input
2

first output
3

second output
4

wIOType
Specifies the source or target type of the card. See ″wIOType Settings″ for possible values.

byFlag
A value that is OR'd together to get attributes about a card. For example, if a card is an input card and the reuse work file attribute is set, byFlag will be CI_INPUT | CI_REUSE. See ″byFlag Settings″ for possible values.

szCardName
A NULL-terminated string that is the name of the card

szFileName
A NULL-terminated string that is the source or destination of the card

Card Setting:
SourceAdapterCommand

TargetAdapterCommand

Execution Command:
-I, -O

byUnused
Reserved for future use

bIntegralMode
Indicates whether the card mode for an input card is **Integral** (TRUE) or **Burst** (FALSE)

Card Setting:
CardMode

bSourceEvent
Indicates whether the input card is being used as a source event (TRUE).

bBackupSwitch
Indicates whether the Backup switch is **ON** (TRUE) or **OFF** (FALSE) for this card

Card Setting:
Backup → Switch

bBackupAlways
Indicates whether the When setting for the Backup option is **Always** (TRUE) or **OnError** (FALSE).

Card Setting:
Backup → When

bMapDirectory
Indicates whether to use the compiled map directory for the backup file

Card Setting:
Backup → FilePath

bAppendBackup
Indicates whether to append the backup data to an existing file, if one exists. Otherwise, a new file is created.

Card Setting:
Backup → BackupFileAction

bRollBackOnFailure
Indicates whether the **OnFailure** setting is **Rollback** (TRUE) or **Commit** (FALSE)

Card Setting:
OnFailure

Execution Command:
`-Ix#B, -Ox#B`

bRetrySwitch
Indicates whether the **AdapterRetry** switch is **ON** (TRUE) or **OFF** (FALSE)

Card Setting:
AdapterRetry

Execution Command:
`-Ix#R, -Ox#R`

bIgnoreAdapterWarnings
Indicates whether warnings returned by the adapter should be ignored (TRUE) or not (FALSE)

Card Setting:
AdapterWarnings

wMapSuccessAction
Indicates action to be taken if a map successfully executes. See ″wMapSuccessAction Settings″ for possible values.

Card Setting:
OnSuccess

Execution Command:
`-Ix#X, -Ox#X`

wScope
Scope of the transaction. See ″wScope Settings″ for possible values.

Card Setting:
AdapterScope

wRetryAttempts
Indicates the number of retry attempts to be made

Card Setting:
AdapterRetry → MaxAttempts

wRetryInterval
Indicates the interval (in seconds) at which retry attempts are to be made

Card Setting:
AdapterRetry → Interval

wFetchUnit
Indicates the number of units to retrieve for each fetch

Card Setting:
FetchUnit

wUnused
Not Used

szBackupFile
Name of the backup file that can include a full path, if specified

Card Setting:
Backup → FilePath

wIOType settings

The `wIOType` argument has the following settings.

Constant	Value	Description
`DTX_DATAFILE`	0	Source or target is a data file
`DTX_STATICFILE`	1	Not valid in this usage
`DTX_WORKFILE`	2	Not valid in this usage
`DTX_TRACEFILE`	3	Not valid in this usage
`DTX_BUFFER`	4	Not valid in this usage
`DTX_DATABASE`	5	Source or target is a database
`DTX_APPLICATION`	6 7 - 99 100+	Source or target is an application User defined Reserved

byFlag settings

The byFlag argument has the following settings.

Constant	Value	Description
CI_INPUT	1	This is an input card
CI_REUSE	2	Reuse the work file if possible
CI_APPEND	4	Append to existing output if possible
CI_UPDATE	8	Update an input card

wScope settings

The wScope argument has the following settings.

Constant	Value	Description
ADPT_SCOPE_MAP	0x0001	Scope for the card is the map
ADPT_SCOPE_BURST	0x0002	Scope for the card is the burst
ADPT_SCOPE_CARD	0x0004	Scope for the card is the card

wMapSuccessAction settings

The wMapSuccessAction argument has the following settings.

For an input card

The wMapSuccessAction argument has the following settings for an input card.

Constant	Value	Description
ADPT_KEEP_ALWAYS	0x0001	Do not remove the input data from its source.
ADPT_KEEP_NEVER	0x0002	Remove the input data from its source.
ADPT_KEEP_ONCONTENT	0x0004	Do not remove the input data from its source, unless it has no content.

For an output card

The wMapSuccessAction argument has the following settings for an output card:

Constant	Value	Description
ADPT_C_F_ALWAYS	0x0001	Always send the output data to its target.
ADPT_C_F_NEVER	0x0002	Never send the output data to its target.
ADPT_C_F_ONCONTENT	0x0004	Only send the output data to its target, if it has content.

Constant	Value	Description
ADPT_C_F_APPEND	0x0008	Append the output data to an existing data file. If the data file does not exist, it should be created.
ADPT_C_F_UPDATE	0x0010	Output data should be used by the adapter to update or replace existing data.

Additional constants

Some additional constants are available:

Constant	Value	Description
MAX_TYPENAME	32	Largest size of the card name in the CARDINFO structure
MAX_PATH	260	Largest size of the file name in the CARDINFO structure

EXEOPTS structure

The EXEOPTS structure is used to provide information about the map settings compiled into the map.

```
struct tagExecutionOptions
{
    DWORD dwTraceFlags;
    DWORD dwTraceInStart;
    DWORD dwTraceInEnd;
    DWORD dwFailFlags;
    DWORD nWorkFileType;
    BOOL  bAuditOn;
    BOOL  bUniqueLog;
    DWORD dwDataLog;
    BOOL  bAppendLog;
    DWORD dwAuditLog;
    DWORD dwMSAudit;
    DWORD dwCSAudit;
    BOOL  bBurstAudit;
    BOOL  bUseMMCforAudit;
    BOOL  bTraceOn;
    BOOL  bUseMMCforTrace;
    BOOL  bDefaultAuditFileName;
    BOOL  bTraceMemory;
    BOOL  bTraceIn;
    BOOL  bTraceOut;
    BOOL  bCardSummary;
    BOOL  bStopOnFirstError;
    BOOL  bOmitRestrictions;
    BOOL  bOmitPresentation;
    BOOL  bOmitSize;
    BOOL  bDeleteWorkFiles;
    BOOL  bUseMMCforWork;
    BOOL  bBurstRestart;
    BOOL  bMemoryLog;
    BOOL  bMemoryActionSized;
    BOOL  bMapRetry;
    BOOL  bCustomValidation;
    BOOL  bCenturySwitch;
    BOOL  bEveryMapWarning;
```

```
    BOOL  bUniqueWorkFilePrefix;
    BOOL  bUnused;
    WORD  wDateResolution;
    WORD  wTraceICardNo;
    WORD  wTraceOCardNo;
    WORD  wRetryCount;
    WORD  wRetryInterval;
    WORD  wPageSizeInK;
    WORD  wPageCount;
    WORD  wBurstErrorLimit;
    char  szAuditDir[264];
    char  szTraceDir[264];
    char  szWorkDir[264];
};
typedef struct tagExecutionOptions EXEOPTS;
typedef struct tagExecutionOptions;
```

EXEOPTS components

The EXEOPTS structure has the following components:

Part **Description**

dwTraceFlags
Specifies the type of trace information to be generated. See ″dwTraceFlags Settings″ for possible values.

dwTraceInStart
Indicates the input object at which to start producing Trace file information.

dwTraceInEnd
Indicates the input object at which to stop producing Trace file information.

dwFailFlags
Specifies the **MapSettings** → **Warnings** value. See ″dwFailFlags Settings″ for possible values.

nWorkFileType
Indicates the type of work area to be used. See ″nWorkFileType Settings″ for possible values.

bAuditOn
Indicates that the audit log is enabled.

bUniqueLog
Indicates that unique names are to be generated for the audit log files.

dwDataLog
If bAuditOn is TRUE, indicates that the data audit log is to be created.

bAppendLog
Indicates that the audit log information is to be appended to an existing file; otherwise, a new file is created.

dwAuditLog
If bAuditOn is TRUE, indicates that the summary execution audit log is to be generated.

dwMSAudit
If bAuditOn is TRUE, indicates that map settings are to be recorded in the audit log.

dwCSAudit
If bAuditOn is TRUE, indicates that card settings are to be recorded in the audit log.

bBurstAudit
If bBurstAudit is TRUE, indicates the burst execution audit log is to be generated.

bUseMMCforAudit
Indicates that the audit log file is to be created using the name and location of the compiled map file.

bTraceOn
Indicates that the trace file is enabled.

bUseMMCforTrace
Indicates that the trace file is to be created using the name and location of the compiled map file.

bDefaultAuditFileName
Indicates to use the <*mapname*>**.log** file-naming convention.

bTraceMemory
Not used.

bTraceIn
Indicates that the trace file is enabled for one or more inputs.

btraceOut
Indicates that the trace file is enabled for one or more outputs.

bCardSummary
Indicates that summary trace is enabled.

bStopOnFirstError
Indicates that map execution is to cease after the first card on which an error occurs.

bOmitRestrictions
Indicates that item data is not checked against restriction values during map execution.

bOmitPresentation
Indicates that the presentation settings of items is to be ignored during map execution.

bOmitSize
Indicates that the minimum size of items in delimited data is not to be checked during map execution.

bDeleteWorkFiles
Indicates that work files are to be deleted after map execution.

bUseMMCforWork
Indicates that work files are to be created using the name and location of the compiled map file.

bBurstRestart
Indicates that **BurstRestart** is enabled.

bMemoryLog
Indicates that the audit log is to be created in memory and returned as the contents of the **lpDataFromApp** buffer.

bMemoryActionSized
Indicates that when the audit log is returned in memory, it is returned in *<size> <sp> <data>* format.

bMapRetry
Indicates that **MapRetry** is enabled.

bCustomValidation
Indicates that **CustomValidation** is enabled.

bCenturySwitch
Indicates that **SlidingCentury** is enabled.

bEveryMapWarning
Indicates that a global **MapWarning** setting is specified.

bUniqueWorkFilePrefix
Indicates that unique work file names are to be generated.

bDefaultTraceFileName
Indicates that the default trace file name is to be used.

wDateResolution
Indicates the **CCLookup** setting for **SlidingCentury**.

wTraceICardNo
Indicates the card number for producing an input trace.

wTraceOCardNo
Indicates the card number for producing an output trace.

wRetryCount
Indicates the number of retries attempted when accessing non-data file resources.

wRetryInterval
Indicates the interval (in seconds) between retries.

wPageSizeInK
Indicates the WorkSpace page size in Kbytes.

wPageCount
Indicates the number of WorkSpace pages.

wBurstErrorLimit
Indicates the maximum number of errors that are permissible for **BurstRestart**.

szAuditDir[264]
Indicates the directory and filename for the audit log file.

szTraceDir[264]
Indicates the directory into which the trace file will be written.

szWorkDir[264]
Indicates the directory into which work files are to be written.

dwTraceFlags settings

The `dwTraceFlags` argument has the following settings:

Constant	Value	Description
`TRACE_ALL`	1	Trace all inputs or outputs

Constant	Value	Description
TRACE_CARD	2	Trace a specific input or output card
TRACE_RANGE	3	Trace a range of input objects

nWorkFileType settings

The nWorkFileType argument has the following settings:

Constant	Value	Description
WF_NORMAL	0	Create work files using the map name
WF_UNIQUE	1	Use unique work file names
WF_INMEM	2	Create work area in memory
WF_FILE	3	Create work files

dwFailFlags settings

The dwFailFlags argument has the following settings:

Constant	Value	Description
FO_OUTPUTINVALID	0x00000001	Fail on return code 14 - One or more outputs was invalid
FO_PAGEUSECTERROR	0x00000002	Fail on return code 18 - Page usage count error
FO_INPUTNOTCONSUMED	0x00000004	Fail on return code 21 - Input valid, but unknown data found
FO_OUTPUTINERRORWARN	0x00000008	Fail on return code 26 - Output type in error
FO_OUTCONTAINSERRWARN	0x00000010	Fail on return code 27 - Output type contains errors
FO_INCONTAINSERRWARN	0x00000020	Fail on return code 28 - Input type contains errors
FO_UNKNOWNINOUTWARN	0x00000040	Fail on return code 29 - Output valid, but unknown data found
FO_ALLWARNINGS[1]	0x0000007F	Fail on all warning return codes (14, 18, 21, 26, 27, 28 and 29)
IW_OUTPUTINVALID	0x00010000	Ignore return code 14 - One or more outputs was invalid
IW_PAGEUSECTERROR	0x00020000	Ignore return code 18 - Page usage count error
IW_INPUTNOTCONSUMED	0x00040000	Ignore return code 21 - Input valid, but unknown data found
IW_OUTPUTINERRORWARN	0x00080000	Ignore return code 26 - Output type in error

Constant	Value	Description
IW_OUTCONTAINSERRWARN	0x00100000	Ignore return code 27 - Output type contains errors
IW_INCONTAINSERRWARN	0x00200000	Ignore return code 28 - Input type contains errors
IW_UNKNOWNINOUTWARN	0x00400000	Ignore return code 29 - Output valid, but unknown data found
IW_ALLWARNINGS [2]	0x007F0000	Ignore all warning return codes (14, 18, 21, 26, 27, 28 and 29)

[1]Value of all FO_XXX OR'd together.

[2]Value of all IW_XXX OR'd together.

Chapter 7. Platform API return codes and error messages

Return codes and error messages are returned when the particular activity completes. Return codes and error messages may also be recorded as specified in the audit logs, trace files and execution summary files.

Messages

The Platform API return codes and messages may result when an invalid command line is specified for the Platform API.

The following table lists the return codes and messages that can result when using the Platform API.

The return codes marked with an asterisk (*) are displayed with the x replaced with the specific option associated with the error.

Return Code	Message
50	Memory allocation failure Occurs when memory fails.
51	Card override failure Occurs when memory fails.
52	I/O initialization failure Occurs when memory fails.
53	Open audit failure The audit log file is not accessible.
54	No command line There is nothing to process.
55	Recursive command files More than one command file is included in the command line.
56	Invalid command line option *-x* The option is invalid for the command.
57	Invalid `W' command line option The Work file option is invalid.
58	Invalid `B' command line option The Batch (close) file option is invalid.
59	Invalid `R' command line option The Refresh Rate option is invalid.
60	Invalid `A' command line option The Audit option is invalid.

61 Invalid `P' command line option

The Paging option is invalid

62 Invalid `Y' command line option

The General I/O Retry option is invalid.

63 Invalid `T' command line option

The Trace option is invalid.

64 Invalid `G' command line option

The Ignore option is invalid

65 Invalid `I' command line option for input *x*

The Source option is invalid for the identified input.

66 Invalid size in echo command line for input *x*

The size specified using the Size option is greater than memory allowed.

67 Invalid adapter type in command line for input *x*

The adapter is not of a known adapter type. Includes `-IMxxx` where `xxx` is an unknown adapter alias.

68 Invalid `O' command line option for output *x*

The target option is invalid for output *x*. The number of characters between the single quotes that represent the options for an adapter exceed 258 characters in the adapter override.

69 Invalid adapter type in command line for output *x*

The adapter is not of a known adapter type. Includes `-OMxxx` where `xxx` is an unknown adapter alias.

70 Command line memory failure

Occurs when memory is exceeded during echo or override card commands.

71 Invalid `D' command line option

The Date option is invalid.

72 Invalid `F' command line option

The Failure option is invalid.

73 Resource manager failure

(Launcher only) The Resource manager is not used, possibly a memory failure.

74 Invalid `Z' command line option

The Ignore option is invalid.

75 Adapter failed to get data on input

Enable the adapter trace to record the adapter activity to discover the cause of the error.

76 Adapter failed to put data on output

Enable the adapter trace to record the adapter activity to discover the cause of the error.

77 Invalid map name

This message can occur in two different cases. First, this message occurs when the map name specified on the command line is more than 32 characters long. Also, this message can occur when there is an error in the command line such that text for another execution command is erroneously being interpreted as the map name. For example, in the command line below, the number representing the size of the echoed data is missing.

```
mymap.mmc -IE1S HereIsMyDataButIForgotToSpecifyTheSize -AED
```

Because the size is missing, it is interpreted to be 0, such that there is no echoed data. The next string encountered on the command line

```
(HereIsMyData...)
```

Because it does not start with a hyphen (-), it is assumed to be the name of the next map to execute. Because the text is longer than 32 characters, the `Invalid Map Name` message is returned.

Notices

This information was developed for products and services offered in the U.S.A.

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

IBM 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:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

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

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION 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 states 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. IBM 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-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply 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:

IBM Corporation
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A.

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 IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM'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 the names and addresses used by an actual business enterprise 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 IBM, 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. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

i5/OS
IBM
the IBM logo
AIX
AIX 5L
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
HelpNow
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
Notes
OS/400
Passport Advantage
pSeries
Redbooks
SupportPac
Tivoli
WebSphere
z/OS

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

IBM WebSphere Transformation Extender, Version 8.1

Index

R

S

T

U

V

W

Printed in USA