

# Oracle Tuxedo Application Runtime for CICS

User Guide

12c Release 2 (12.1.3)

April 2014

**ORACLE**

Copyright © 2010, 2014 Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

# Contents

## Introduction to CICS Runtime

Introduction to the CICS Runtime Environment . . . . .	1-1
Purpose . . . . .	1-1
How This Book Is Organized. . . . .	1-1

## Overview of the CICS Runtime

General Architecture . . . . .	2-1
The CICS Runtime Library . . . . .	2-2
The CICS Runtime Tuxedo Servers . . . . .	2-4
Mandatory Servers. . . . .	2-4
Optional Servers. . . . .	2-4
Server Generation . . . . .	2-5
Server Configuration . . . . .	2-5

## Initial Configuration of the CICS Runtime

CICS Runtime Configuration. . . . .	3-1
The UNIX ~/.profile File . . . . .	3-1
The Tuxedo System Files. . . . .	3-3
The CICS Runtime Resource Configuration Files . . . . .	3-10
Verifying the Initial Setting Configuration . . . . .	3-14
Using the Tuxedo tadmin psr Commands. . . . .	3-14
Using the Tuxedo tadmin psc Commands . . . . .	3-15
Using the CSGM CICS Good Morning Transaction . . . . .	3-16

# Implementing CICS Applications

Presentation of the z/OS Simple Application . . . . .	4-2
Introduction . . . . .	4-2
Description of the CICS Simple Application Components . . . . .	4-3
Configuring a Standard CICS Application With CICS Runtime . . . . .	4-4
CICS Runtime Configuration . . . . .	4-5
Verifying the CICS Application Installation . . . . .	4-13
Using the Tuxedo tadmin psr Commands . . . . .	4-13
Using the Tuxedo tadmin psc Commands . . . . .	4-14
Using the CICS Runtime Application . . . . .	4-15
Presentation of Simple Application on COBOL-IT / BDB . . . . .	4-17
Configuring ubbconfig File in CICS Runtime . . . . .	4-17
Building BDB TMS Server . . . . .	4-18
Exporting Variables Before Booting Up ART Servers . . . . .	4-18
Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances . . . . .	4-19
The Special Case of Transaction Classes With MAXACTIVE=1 . . . . .	4-19
Modification of the ubbconfig File for Sequential Transactions . . . . .	4-19
Checking the ARTSTR1 Configuration . . . . .	4-23
Implementing Asynchronous CICS Non-Delayed Transactions . . . . .	4-25
Modifying the Tuxedo ubbconfig File to Manage Asynchronous Transactions . . . . .	4-25
Using Parallel Asynchronous Transactions . . . . .	4-25
Using Non-Parallel Asynchronous Transactions . . . . .	4-27
Implementing Asynchronous CICS Delayed Transactions . . . . .	4-29
Implementing Asynchronous Transactions With ARTSRM Server . . . . .	4-29
Implementing Asynchronous Transactions With /Q . . . . .	4-30
Implementing CICS Application Using Temporary Storage (TS) Queues . . . . .	4-35

Implementing Unrecoverable TS Queues . . . . .	4-38
Implementing Recoverable TS Queues . . . . .	4-38
Managing TD Queue Intrapartitions. . . . .	4-42
Presentation of the Mechanism on Source Platform . . . . .	4-42
Automatic Transaction Initiation (ATI) . . . . .	4-43
Presentation of the Mechanism on Target Platform . . . . .	4-44
Runtime CICS Configuration of TD Queue Intrapartition . . . . .	4-45
Activating the ARTTDQ in the Tuxedo ubbconfig File . . . . .	4-49
Implementing CICS Application Using Temporary Storage (TS) Queue POOL . . . .	4-51
Implementing Distributed Program Link (DPL) . . . . .	4-55
To Detect That DPL Is Needed . . . . .	4-55
Modifying the Tuxedo ubbconfig File to Manage the DPL . . . . .	4-57
Declaring Remote Programs in CICS Runtime. . . . .	4-61
Implementing CICS Common Work Area (CWA) . . . . .	4-66
Implementing a CICS Transaction Work Area (TWA) . . . . .	4-67
Supporting TWA in ARTDPL . . . . .	4-70
Implementing Integration with WebSphere MQ . . . . .	4-72
Using ART CICS Transaction Trigger Monitor (ARTCKTI) . . . . .	4-72
Rebuilding ART for CICS Servers . . . . .	4-75
Handling CICS Runtime Preprocessor of MQOPEN/MQCLOSE Calls . . . . .	4-79
Encoding Character Set . . . . .	4-80
Changing COMP-5 back to BINARY Data Type . . . . .	4-81
Implementing Using Multiple Session Management . . . . .	4-81
Writing User Plug-In for Application List. . . . .	4-82
Configuring CICS Runtime Configuration Files. . . . .	4-82
Configuring UBBCONFIG . . . . .	4-83
Starting, Switching, and Ending Sessions . . . . .	4-84
Implementing Using ART for CICS TCP/IP Socket Interface. . . . .	4-85

ART for CICS TCP/IP Socket API . . . . .	4-86
The Client-Listener-Server Application Set . . . . .	4-92
ART for CICS TCP/IP Listener (ARTCSKL) . . . . .	4-95
Required Configurations . . . . .	4-99
Implementing Transferring CICS Regions . . . . .	4-99
Configuring ARTSRM Server . . . . .	4-99
Configuring Environment Variables . . . . .	4-100
CICS Runtime Configuration Files Declaration . . . . .	4-100
Logon ART CICS . . . . .	4-106
Implementing Intersystem Communication . . . . .	4-107
Implementing Distributed Transaction Processing (DTP) . . . . .	4-107
Implementing Asynchronous Processing . . . . .	4-111
Implementing Synchronous Processing . . . . .	4-114
Implementing Submitting JCL/KSH Online . . . . .	4-116
Submitting JCL/KSH Job Online . . . . .	4-116
Submitting JCL/KSH Job Online by SPOOL . . . . .	4-118
Implementing ART for CICS Control Utility . . . . .	4-118
Use Case 1: Implementing ART for CICS Control Utility in End-to-End Mode (IPCP Command Set) . . . . .	4-119
Use Case 2: Implementing ART for CICS Control Utility in Interactive Mode (Interactive Command Set) . . . . .	4-126
Implementing Printing CICS Runtime Applications Data . . . . .	4-126
General Configurations . . . . .	4-127
Implementing Printing with a START Command . . . . .	4-130
Implementing Printing with Transient Data . . . . .	4-131
Implementing Invoking Web Services from CICS Applications . . . . .	4-132
Converting WSDL File into MIF and Generating COPYBOOK . . . . .	4-132
Generating RECORD Definition from COPYBOOK . . . . .	4-132

Configuring SALT and Metadata Repositories . . . . .	4-132
Configuring webservice.desc . . . . .	4-133
Modifying UBBCONFIG . . . . .	4-133
Implementing ART for CICS Application Server Customized Callback Support . . .	4-133
Create Shared Library libkixcallback.so . . . . .	4-134
Include Customized C Library for Dynamically Loading. . . . .	4-136
Use Case 1: Create Shared Memory at Server Initiation. . . . .	4-136
Use Case 2: Open Database Table at Server Initiation . . . . .	4-137
Implementing Resource-Based Authorization . . . . .	4-137
Implementing COBOL Program Debugging in CICS Runtime. . . . .	4-139
CICS Runtime Logs . . . . .	4-141
Tuxedo System Log. . . . .	4-141
The CICS Runtime Server Logs. . . . .	4-141
Disabling and Enabling Programs . . . . .	4-145
Disabling Programs . . . . .	4-145
Enabling Programs . . . . .	4-146
Checking the Change in Program Status . . . . .	4-146
Removing and Adding Applications for CICS Runtime. . . . .	4-148
CICS Runtime C Program Support. . . . .	4-150
Running C Program in CICS Runtime . . . . .	4-150
C Programming Restrictions and Requirements . . . . .	4-150
Accessing EIB from C. . . . .	4-152
Accessing COMMAREA from C . . . . .	4-152
CICS Command Translator . . . . .	4-152
C Program Compilation. . . . .	4-153

## Reference

Cross Reference of .desc Configuration Files Used by CICS Runtime Servers . . . . .	5-1
---	-----

# Oracle Tuxedo Application Runtime for CICS CSD Converter

Overview .....	6-1
Resource Definition Online (RDO) Mapping .....	6-1

## ECI Client Support

Overview .....	7-1
Purpose .....	7-1
Introduction .....	7-2
Platform .....	7-2
Installation and Setup .....	7-3
Installation .....	7-3
ECI Connection to ART CICS .....	7-3
Configuration on ART CICS .....	7-3
Encoding and Decoding .....	7-3
Security .....	7-4
Failover .....	7-4
Diagnostic .....	7-5
Limitation and Compatibility .....	7-5
Limitation .....	7-5
Compatibility .....	7-5

## IMS DB Access Support

Overview .....	8-1
Configurations .....	8-1
Configure ART for CICS for Accessing IMS DB .....	8-2
Configuring ART for CICS Servers .....	8-2
Configuring Environment Variables .....	8-2
Configuring IMS .....	8-3



Supported Platforms . . . . .	8-3
Tips . . . . .	8-3

## UDB Linking

Installation Time UDB Linking . . . . .	A-1
Rebuilding Servers for UDB . . . . .	A-1

## Rebuilding ART Servers for CICS

Rebuilding the ART CICS Servers . . . . .	B-1
---	-----

## External CICS Interface (EXCI)

Overview . . . . .	C-1
EXCI in Oracle Tuxedo Application Runtime . . . . .	C-2
Supported EXCI Interface . . . . .	C-2
Precompiler Controls . . . . .	C-3
Access Authorization . . . . .	C-3
ART CICS Implementation . . . . .	C-4

## COBOL Program Debugging and Error Processing in CICS Runtime

Debugging COBOL Programs in CICS Runtime . . . . .	D-1
Debugging with Micro Focus COBOL . . . . .	D-2
Debugging with COBOL-IT COBOL . . . . .	D-2
Configuration . . . . .	D-3
Dynamically Load the Debug Configuration File . . . . .	D-3
See Also . . . . .	D-4
Error Processing in CICS Runtime . . . . .	D-4
Prerequisite . . . . .	D-4
Memory Dump . . . . .	D-4

# Integrating Client Applications Using CPI-C

Overview .....	E-1
Client Applications Impact .....	E-2
Supported CPI-C Scenarios .....	E-3
Windows Application Calling Rehosted CICS Transactions .....	E-3
WebLogic Application Calling Rehosted CICS Transactions .....	E-4
Server Side Configuration .....	E-5
ART CICS Resources Configuration .....	E-6
Oracle Tuxedo Configuration .....	E-7
Client Side Configuration .....	E-8
Configuration for Windows Client .....	E-9
Configuration for WebLogic Client .....	E-9
Oracle Tuxedo Timeout Controls .....	E-9
Settings in UBBCONFIG .....	E-10
Security .....	E-11
Scaling .....	E-11
Diagnostics .....	E-11
Packaging/Installation .....	E-11

# Introduction to CICS Runtime

## Introduction to the CICS Runtime Environment

### Purpose

This guide provides explanations and instructions for configuring and using Oracle Tuxedo Application Runtime for CICS (CICS Runtime) when developing and running On Line Transaction Processing (OLTP) applications on a UNIX/Linux platform.

This guide describes the steps required to implement and perform COBOL CICS transactions, whether they are migrated from z/OS CICS or newly written for UNIX applications.

To illustrate this purpose, the User Guide provides a detailed description of the deployment and administration of the Simple Application in a UNIX environment.

This guide helps you to:

- Configure CICS Runtime software.
- Declare components to CICS Runtime.
- Run a CICS Application.

### How This Book Is Organized

This guide is divided into seven main chapters:

- [“Overview of the CICS Runtime” on page 2-1](#): introduces the general principles of the CICS Runtime.
- [“Initial Configuration of the CICS Runtime” on page 3-1](#): describes how to set parameters to make CICS Runtime operational before implementing CICS applications.
- [“Implementing CICS Applications” on page 4-1](#): details how to configure the CICS Runtime to use CICS applications including examples moving from simple to more-and-more complex cases.

Additionally,

- [“Reference” on page 5-1](#): contains information describing the .desc files used by the different CICS Runtime servers.
- [“Oracle Tuxedo Application Runtime for CICS CSD Converter” on page 6-1](#): specifies how to set the target CSD file in argument, and the translated resource configuration files resides in current directory by default.
- [“ECI Client Support” on page 7-1](#): elaborates ECI emulator that supports customers to keep using existed program without code change when migrating mainframe applications from IBM z/OS to an open systems application grid running Oracle Tuxedo.

# Overview of the CICS Runtime

## General Architecture

In a z/OS environment, CICS is used to establish transactional communications between end-users and compiled programs via screens.

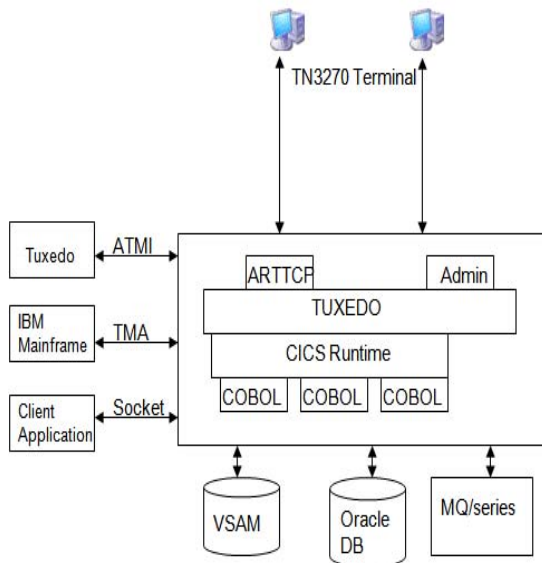
CICS is a middleware that implements the control and integrity of shared resources, providing developers with a bag of API (EXEC CICS ... END-EXEC statements) to dialog with CICS inside programs mainly developed on z/OS in COBOL, PL1 and Assembler languages.

Once all the components of z/OS CICS applications (COBOL programs and data) are migrated to a UNIX/linux platform using Oracle Tuxedo Application Runtime Workbench, CICS Runtime enables them to be run unchanged using an API emulation on top of the native Tuxedo features.

On a UNIX platform, Tuxedo performs many of the functions performed by CICS on a z/OS platform concerning the integrity of resources and data used in transactional exchanges, including those used for applications that are distributed across several machines. However, Tuxedo does not manage some specific native CICS z/OS features such as screen map handling. To provide these features on the target platform, CICS Runtime acts as a technical layer, located between Tuxedo and the converted CICS applications.

The following schema describes the global architecture of CICS Runtime.

**Figure 2-1 Oracle Tuxedo Application Runtime for CICS Architecture**



CICS Runtime is composed of two major parts:

- CICS Runtime Preprocessor and CICS Runtime library
- CICS Runtime Tuxedo Servers and their Resource Configuration Files

## The CICS Runtime Library

In z/OS CICS applications, all the interactions with the resources managed by CICS are made through the EXEC CICS API.

A CICS Preprocessor transforms these statements into calls to CICS library:

### Listing 2-1 z/OS CICS Calls

```
*EXEC CICS  
*      RECEIVE MAP      ('RTSAM10')
```

```

*           MAPSET ('RTSAM10')
*           INTO   (RTSAM10I)

*END-EXEC.

MOVE 'xxxxxxxxxxxxx00203' TO DFHEIV0
MOVE 'RTSAM10' TO DFHC0070
MOVE 'RTSAM10' TO DFHC0071
CALL 'DFHEI1' USING DFHEIV0 DFHC0070
                    RTSAM10I DFHDUMMY DFHC0071.

```

---

On UNIX, the CICS Runtime Preprocessor transforms these EXEC CICS into calls to the CICS Runtime library:

### Listing 2-2 CICS Runtime Calls

---

```

*EXEC CICS
*   RECEIVE MAP   ('RTSAM10')
*           MAPSET ('RTSAM10')
*           INTO   (RTSAM10I)
*END-EXEC.

INITIALIZE KIX--INDICS
MOVE LOW-VALUE TO KIX--ALL-ARGS
. . .
ADD 1 TO KIX--ARGS-NB
SET KIX--INDIC-MAPSET(KIX--ARGS-NB) TO TRUE
MOVE 'RTSAM10' TO KIX--MAPSET OF KIX--BMS-ARGS
ADD 1 TO KIX--ARGS-NB
SET KIX--INDIC-MAP(KIX--ARGS-NB) TO TRUE
MOVE 'RTSAM10' TO KIX--MAP OF KIX--BMS-ARGS

```

```
CALL "KIX__RECEIVE_MAP" USING KIX--INDICS KIX--ALL-ARGS
```

---

## The CICS Runtime Tuxedo Servers

The CICS Runtime Tuxedo servers are used to manage CICS features not natively present in Tuxedo.

Some of these servers are mandatory in order to make CICS Runtime available; others are optional depending on user's actual scenario and the usage of specific EXEC CICS statements in CICS Applications.

### Mandatory Servers

- The Administration server (ARTADM): required for CICS Runtime administration.

### Optional Servers

- The Terminal Connection servers (TCP servers: ARTTCPH and ARTTCPL servers): manage user connections and sessions to CICS applications thru 3270 terminals or emulators.
- The Connection server ARTCNX: manages the user session and some technical transactions relative to security (CSGM: Good Morning Screen, CESN: Sign On, CESF: Sign off).
- The Synchronous Transaction server ARTSTRN: manages standard synchronous CICS transactions that can run simultaneously.
- The Synchronous Transaction servers ARTSTR1: manages CICS synchronous transaction applications that can not run simultaneously but only sequentially (one at a time).
- The Asynchronous Transaction servers ARTATRN and ARTATR1: are similar to the ARTSTRN and ARTSTR1 but for asynchronous transactions started by EXEC CICS START TRANSID statements.
- The TS Queue servers ARTTSQ, TMQUEUE and TMQFORWARD: manage the use of CICS Temporary Storage Queues - files managed by CICS thru specific commands.
- The TD Queue servers ARTTDQ: centralizes the TD Queue operations management requested by applications.
- The Distributed Program Link server ARTDPL: runs DPL programs.



- Converse Management server `ARTCTRN` and `ARTCTR1`.
- The Web Transaction servers `ARTWTRN` and `ARTWTR1`: manage synchronous (no-conversational) non-3270s clients oriented transactions.
- The System and Resource Management (`ARTSRM`) Server centralizes the management of ART runtime information, which is generated and queried by applications.
- `ARTSHM` is used to manage shared memory for `GETMAIN SHARED`. It handles shared memory allocation and free request.
- `ARTCSKL` is the listener of ART for CICS TCP/IP socket and can perform the same functions as CICS TCP/IP listener `CSKL`.

## Server Generation

Some CICS Runtime Tuxedo servers need to be built by the tool “`buildartcics`”, such as “`ARTSTRN`”, “`ARTSTR1`”, “`ARTSTRN_UBD`”, etc.

For more information, see the [Oracle Tuxedo Application Runtime from CICS Reference Guide](#)

## Server Configuration

The CICS Runtime Tuxedo servers are configured in:

1. The `ubbconfig` file once compiled to the `tuxconfig` file, is the file read by Tuxedo at start up that defines all the servers to be launched and their parameters.
2. The CICS Runtime resource configuration files for the CICS resources managed by CICS Runtime servers are declared.

## The CICS Runtime Resource Configuration Files

### Reminder about z/OS Resource Management

On z/OS, all the technical components used by CICS applications (terminals, transactions, programs, maps, files ...) are named CICS resources and must be declared to CICS using a dedicated configuration file called `CSD`.

Each resource declared must belong to a resource Group name. This enables a set of resources bound together constituting a technical or a functional application to be managed (install, delete, copy to another `CSD`...).

Once created, one or more CICS groups can be declared in a CICS List name. All or part of these List names are given to CICS at startup to install their CICS groups, and thus make available all the resources defined in these groups.

### **CICS Runtime Resource Management**

CICS Runtime manages only a subset of the resource types previously defined in the CICS CSD file on z/OS. Each resource type definition of this subset is stored inside its own dedicated Resource Configuration file. All these files are located in the same UNIX directory.

The Group name notion is kept to preserve the same advantages as on the z/OS platform. For this purpose, each resource defined in the configuration files must belong to a CICS Group name.

CICS Runtime manages the following resources:

- Tranclasses (`transclasses.desc` file)

This file contains all the distinct Transaction classes (Tranclasses) referenced by the CICS Transactions. In CICS Runtime, a Tranclass is a feature defining whether several instances of the same transaction can be run simultaneously or sequentially.

- Transactions (`transactions.desc` file)

A transaction is a CICS feature allowing a program to be run indirectly thru a transaction code either manually from a 3270 screen or from another COBOL CICS program.

A transaction belongs to a transaction class in order to define whether this transaction must be run exclusively.

- Programs (`programs.desc` file)

This file contains a list of all COBOL or C programs invoked thru `EXEC CICS START`, `LINK` or `XCTL` statements.

- TS Queue Model (`tsqmodel.desc` File)

Contains all the TS Queue models referenced by TS Queues used in the CICS programs.

A TS Queue model defines properties that complete or replace those defined in the CICS API that manages Temporary Storage Queues. The names of these TS Queues must match a mask defined in the TS Queue model. In CICS Runtime, these models are mainly used to define whether TS Queues are recoverable or not.

- Mapsets (`mapsets.desc` file)

This file contains all the mapsets referenced by the CICS applications. A mapset is a CICS resource, but also a physical component containing one or more screens (maps) used in the exchanges between CICS applications and end-users.

These resources are used through dedicated CICS statements like `EXEC CICS SEND MAP` or `RECEIVE MAP` inside COBOL programs.

- Typeterms (typeterms.desc file)  
Contains all of the 3270 terminal types supported by the CICS Runtime TCP servers.
- Enqmodel (enqmodel.desc)  
defines named resources for which the `EXEC CICS ENQ` and `EXEC CICS DEQ` commands have a sysplex-wide scope.
- Extra TDQUEUE (tdqextra.desc)  
Defines the attributes of extra transient data queues
- Intra TDQUEUE (tdqintra.desc)  
Defines the attributes of intra transient data queues
- System (system.desc)  
Replaces the functions of system initialization table (SIT) on Mainframe.
- Terminal (terminals.desc)  
Defines terminal and its attributes.
- Connection (connections.desc)  
Defines the list of connections that can be loaded by ART CICS application servers.
- Web Service (webservice.desc)  
Defines the Web services to be invoked. This file is used for the `INVOKE WEBSERVICE` command.
- Programs List (program\_list\_table.desc)  
Define the program list to be executed during ART CICS boot or shutdown.
- TCP/IP Socket Listener (listener.desc)  
Define ART for CICS TCP/IP socket listener information.

## Overview of the CICS Runtime

**Note:** ART CICS Runtime provides the `tcxcscvrt` utility to automatically convert the CICS CSD file to CICS Runtime resource configuration files. For more information, see [Oracle Tuxedo Application Runtime for CICS CSD Converter](#).

# Initial Configuration of the CICS Runtime

## CICS Runtime Configuration

Before installing a CICS application, certain technical variables and paths must be defined in order to create the CICS Runtime environment.

These operations must be completed before configuring individual CICS applications for use with CICS Runtime.

CICS Runtime uses the following files:

- The UNIX System `~/.profile` file to centralize values and paths used by the CICS Runtime for its own needs or for Tuxedo.
- The Tuxedo `envfile` which contains parameters, variables and paths used by Tuxedo.
- The Tuxedo `ubbconfig` file to declare all the required CICS Runtime Tuxedo servers.
- The CICS Runtime resource configuration files used by the CICS Runtime Tuxedo servers.

## The UNIX `~/.profile` File

For UNIX users, most required variables are defined in the `.profile` file that centralizes all of the common variables and paths used by a user for commands and applications.

Set up in this file all of the common variables and paths that will be used later in the different configuration files required by CICS Runtime or by the other technical software or middleware invoked by it (Oracle, Tuxedo, MQ Series ...).

## Initial Configuration of the CICS Runtime

This file should then be exported.

Set the following variables in the initial settings of `~/.profile` file.

**Table 3-1 .profile Variables**

Variable	Value	Usage	Variable usage
TUXDIR	Set up at Installation time	Compulsory. Directory containing the Installed Oracle Tuxedo product.	TUXEDO
TUXCONFIG	Set up at Installation time	Compulsory. Full path name of the Tuxedo tuxconfig file	TUXEDO
KIXDIR	Set up at Installation time	Compulsory. Absolute path of the directory containing the CICS Runtime product	CICS Runtime
APPDIR	<code>\${KIXDIR}/bin</code>	Compulsory. Directory containing the CICS Runtime Servers Binaries	CICS Runtime
KIXCONFIG	Set up at Installation time	Compulsory. Directory where the Resources Configuration Files of the CICS Runtime are located	CICS Runtime
KIX_TS_DIR	Set up at Installation time	Compulsory. Directory used for the non-recoverable CICS Queue TS.	CICS Runtime

**Listing 3-1 .profile file Initial Settings Example**

```
export TUXDIR=/product/TUXEDO11GR1# Directory containing the Installed
Tuxedo product

export TUXCONFIG=${HOME}/SIMAPP/config/tux/tuxconfig# Full path name of the
Tuxedo tuxconfig file

export KIXDIR=${HOME}/KIXEDO# Absolute path of the CICS Runtime product
directory

export APPDIR=${KIXDIR}/bin # Directory containing the CICS Runtime
Servers Binaries
```

```
export KIXCONFIG=${HOME}/SIMAPP/config/resources # Directory for resources
files (*.desc)

export KIX_TS_DIR=${HOME}/SIMAPP/KIXTSDIR# Directory for TS no recovery
```

---

## The Tuxedo System Files

### The Tuxedo Envfile File

This envfile contains variables and paths used by Tuxedo and CICS Runtime. These parameters should be set in addition to those set by the Tuxedo Administrator.

Set the following variables in the initial settings of the envfile file:

**Table 3-2 envfile Variables**

Variable	Value	Usage
LC_MESSAGES	C	UNIX formats of informative and diagnostic messages
OBJECT_MODE	64	UNIX 64 bits architecture
APPDIR	\${APPDIR}	TUXEDO environment.
TUXCONFIG	\${TUXCONFIG}	TUXEDO environment
USER_TRACE	SID	TUXEDO environment. Trace Type (one per user)
KIXCONFIG	\${KIXCONFIG}	CICS Runtime directory containing its resource files
PATHTS	\${KIX_TS_DIR}	CICS Runtime directory used for the unrecoverable Temporary Storage

**Listing 3-2 envfile Initial Settings Example**

```
# <TUXDIR>
#     Refers to the location where you installed TUXEDO. The default
#     location is "/usr/tuxedo".
#
```

## Initial Configuration of the CICS Runtime

```
# <APPDIR>
#     Refers to the fully qualified directory name where your application
#     runs (i.e., the location of the libraries, mapdefs, and MIB files).
#
# <TUXCONFIG>
#     Refers to the fully qualified binary version of the TUXEDO
#     configuration file. (This is usually the "tuxconfig" in the $APPDIR
#     directory.)
#
# Copyright ©1998, BEA Systems, Inc., all rights reserved.
#-----
-
# TUXEDO environment
APPDIR=${KIXDIR}/bin
CONFDIR=${APPHOME}/config/tux
TUXCONFIG=${CONFDIR}/tuxconfig
FLDTBLDIR32=${KIXDIR}/src
FIELDTBLS32=msgflds32
OBJECT_MODE=64

#resource files directory
KIXCONFIG=${APPHOME}/config/resources

# Command executable paths
HAB_TRAN=none

# Other environment
LC_MESSAGES=C
```



```
# End
```

---

## The Tuxedo ubbconfig File

The following initial configuration of CICS Runtime is configured for typical user scenario using 3270s clients oriented transactions. Some CICS Runtime Tuxedo servers are absolutely needed while others can be optionally started and are not absolutely necessary at this time.

### The Mandatory Servers

These servers must be started to run CICS Runtime and verify that the initial settings are correct by being able to display the CICS Runtime Good Morning screen (Host Connection Welcome Screen).

- The Terminal Control Program Listener (ARTTCPL server) is needed because it establishes communication between end-users and CICS Runtime applications thru maps displayed on 3270 terminals or emulators.
- The Connection Server (ARTCNX server) is also required because it offers technical connections services during the user connection and disconnection phases. It is also used to display the CICS system transactions CICS Runtime Good Morning screen thru the System Transaction CSGM.
- The Administration Server (ARTADM server) is needed to replicate resources files for all other servers.

### The Optional Servers

These servers do not need to be launched because they are only used by CICS applications not yet installed.

To not start these servers, comment-out the corresponding line in your ubbconfig file before recompiling.

- The Synchronous Transaction Servers (ARTSTRN and ARTSTR1) that manage synchronous transaction CICS applications
- The Asynchronous Transaction Servers (ARTATR1 and ARTATR1) that manage asynchronous transaction CICS applications.

## Initial Configuration of the CICS Runtime

- The Temporary Storage Server (ARTTSQ server) that manage TS QUEUES used in COBOL CICS programs.
- The Tuxedo /Q TMQUEUE and TMQFORWARD servers that are only used for delayed CICS Transactions.

**Note:** TMQUEUE must be started before ARTCNX to support the followings:

- INQUIRE NETNAME TERMINAL () ACQSTATUS ()
- INQUIRE TERMINAL NETNAME () ACQSTATUS ()
- SET TERMINAL RELEASED/ACQUIRED/CREATE
- Static LUNAME from 3270 terminal

### Listing 3-3 ubbconfig Initial Server Configuration Example

---

```
*SERVERS

ARTTCPL SRVGRP=TCP00

        SRVID=101

        CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_tcp -e
/home2/work9/demo/Logs/TUX/sysout/stderr_tcp -- -M 4 -m 1 -L //deimos:2994
-n //deimos:2992"

ARTADM      SRVGRP=ADM00

        SRVID=3000

        SEQUENCE=1

        MIN=1 MAX=1

        CLOPT="-o /home2/work9/trf/Logs/TUX/sysout/stdout_adm -e
/home2/work9/trf/Logs/TUX/sysout/stderr_adm -r --"

ARTCNX      SRVGRP=GRP01

        SRVID=15

        CONV=Y

        MIN=1 MAX=1 RQADDR=QCX015 REPLYQ=Y
```

```
CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_cnx -e
/home2/work9/demo/Logs/TUX/sysout/stderr_cnx -r --"
```

---

Where:

**\*SERVERS**

Is the Tuxedo ubbconfig keyword indicating server definitions.

For the ARTTCPL server:

**SRVGRP**

Is the Tuxedo Group Name to which ARTTCPL belongs.

**SRVID**

Is the identifier of a ARTTCPL Tuxedo Server.

**CLOPT**

Is a quoted text string passed to the server containing its parameters.

-o

Indicates the file is used for the standard output messages of the server.

-e

Indicates the file is used for the error output messages of the server.

-M 4

Indicates the maximum number of TCPL handler processes is 4.

-m 1

Indicates that the minimum number of TCPL handler processes is 1.

-L //deimos:2994

Indicates the internal URL address used by TCPL and TCPH for their own communication.

-n //deimos:2992

Indicates the URL address where the TN3270 terminals connect to TCPL.

For the ARTADM server:

**SRVGRP**

Is the Oracle Tuxedo group name to which ARTADM belongs.

**SRVID**

Is the identifier of a Tuxedo Server of ARTADM.

**SEQUENCE=1**

This line is mandatory. It indicates this server must be started first.

**MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

**CLOPT**

Is a quoted text string passed to the server containing its parameters.

-o

Indicates the file is used for standard output messages of the server.

-e

Indicates the file is used for error output messages of the server.

-r

Is a Tuxedo parameter used to produce statistical reports.

For the ARTCNX server:

**SRVGRP**

Is the Tuxedo Group Name to which ARTCNX belongs.

**SRVID**

Is the identifier of a Tuxedo Server of ARTCNX.

**CONV=Y**

Indicates that this server operates in a conversational mode.

**MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

**REPLYQ=Y**

Indicates that this server will respond.

**RQADDR=QCNX015**

Name of the Tuxedo queue used for the responses.

**CLOPT**

Is a quoted text string passed to the server containing its parameters.

-o

Indicates the file is used for the standard output messages of the server.

- e            Indicates the file is used for the error output messages of the server.
- r            Is a Tuxedo parameter used to produce statistical reports.

## The Mandatory Server Groups

To be started, a Tuxedo Server must be defined in a Tuxedo Server Group previously defined in the ubbconfig file. As the ARTTCPL and ARTCNX servers are mandatory, verify that their groups are defined, present and not commented-out, in the ubbconfig file.

In our example, ARTTCPL belongs to the Tuxedo Server Group TCP00 (SRVGRP=TCP00) and ARTCNX belongs to the Server Group (SRVGRP=GRP01); therefore the ubbconfig file contains these two Server Group definitions in the following example:

### Listing 3-4 Server Group Definitions

---

```
*GROUPS
DEFAULT:      LMID=KIXR

# Applicative groups
TCP00          LMID=KIXR
                GRPNO=1
                TMSCOUNT=2

ADM00          GRPNO=5

GRP01
    GRPNO=11
    ENVFILE="/home2/work9/demo/config/tux/envfile"
```

---

Where:

#### **\*GROUPS**

Tuxedo ubbconfig Keyword indicating definitions of Servers Groups.

**LMID=**

Name of the CICS.

**GRPNO=**

Tuxedo Group.

**TMSCOUNT=**

Number of Tuxedo Transaction Manager Servers.

**ENVFILE**

Path of the Tuxedo envfile.

## The Optional Server Groups

These groups are used to contain the optional servers. The first group is used by the Tuxedo Server Servers Groups: ARTSTRN, ARTSTR1, ARTATRN, ARTATR1, ARTTSQ used by CICS Applications. The second one is used only for TS QUEUE management.

## The CICS Runtime Resource Configuration Files

All of the following files must exist in the `$(KIXCONFIG)` path, even when empty, for CICS Runtime to be operational.

### The Mandatory Populated Files

1. The `typeterms.desc` Configuration File

This file used by the TCP servers, describes the different kinds of terminals used with a 3270 terminal or emulator.

#### Listing 3-5 typeterm Description Example

---

```
[typeterm]
name=IBM-3279-5E
color=YES
defscreencolumn=80
defscreenrow=24
description="IBM 327x family terminal"
highlight=YES
```

```
logonmsg=YES  
outline=NO  
swastatus=ENABLED  
uctran=NO  
userarealen=0
```

---

Where

**[typeterm]**

Keyword to define a terminal type.

**name=**

Type of terminal.

**color=YES**

Indicates whether the terminal uses extended color attributes.

**defscreencolumn= 80**

Number of columns of the terminal.

**defscreenrow=24**

Number of rows of the terminal

**description="..."**

Comment about the terminal.

**hilight=YES**

Indicates that this terminal supports the highlight feature.

**logonmsg=YES**

Indicates that "Good Morning" (CSGM) transaction is automatically started on the terminal at logon time.

**outline=NO**

Indicates that this terminal does not support field outlining.

**swastatus=ENABLED**

Indicates that this terminal type is available for use by the system.

**uctran=NO**

Indicates that the lowercase alphabetic characters are not to be translated to uppercase

**userarealen=0**

The terminal control table user area (TCTUA) area size for the terminal.

2. The `mapsets.desc` Configuration File

This file must contain at least the following definition to start the CSGM transaction and see the Good Morning screen.

**Listing 3-6 mapsets.desc Example**

---

```
[mapset]
name=ABANNER
filename=<KIXDIR>/sysmap/abanner.mpdef
```

---

Where:

**name=**

Is the logical mapset name used inside the programs in the EXEC CICS SEND/RECEIVE MAP(map name) MAPSET(mapset name) ... END-EXEC statements.

**filename=**

Is the physical path containing the binary file resulting from the compilation of a mapset file source coded in a CICS z/OS BMS format.

**Note:** For the particular case of the ABANNER system mapset, the filename is located under the `${KIXDIR}` directory. The bracketed text `<KIXDIR>` must be replaced by the value of the `${KIXDIR}` variable of your UNIX `~/ .profile` system file.

In our example the result will be:

**Listing 3-7 mapsets.desc Example with `${TUXDIR}` Substitution**

---

```
[mapset]
name=ABANNER
filename=/product/art11gR1/Cics_RT/sysmap/abanner.mpdef
```



---

## The Optional Initially Populated Files

All the following files can be initially left empty:

- The `transclasses.desc` Configuration File
- The `transactions.desc` Configuration file
- The `programs.desc` Configuration File
- The `tsqmodel.desc` Configuration File
- The `mapsets.desc` Configuration File
- The `connections.desc` Configuration File
- The `program_list_table.desc` Configuration File

The contents and use of these files is described later.

**Note:** If these files are left empty, when Tuxedo launches the CICS Runtime servers, some error messages "`CMDTUX_CAT:1685: ERROR: Application initialization failure`" could be displayed after the boot message of the `ARTSTRN`, `ARTSTR1`, `ARTATRN` and `ARTATR1` servers indicating that the CICS Runtime considers this to be an anomaly.

The real number and type of servers displaying these messages depends on the servers initially launched by your `ubbconfig` file.

In this case, the servers concerned will not be mounted.

For the moment, ignore these error messages, they do not impact the Initial Setting.

- The `system.desc` Configuration File
- The `terminals.desc` Configuration File

**Note:** For more information, please see [Oracle Tuxedo Application Runtime for CICS Reference Guide](#).

## Verifying the Initial Setting Configuration

### Using the Tuxedo tadmin psr Commands

Once all the files have been modified (and compiled for the ubbconfig), stop and restart Tuxedo to take their modifications into account.

The first control is to check that they are individually correctly accepted by Tuxedo and Oracle by a visual control of the boot messages of the Tuxedo CICS Runtime Tuxedo servers.

Once this first check is made, you can enter the Tuxedo `tadmin psr` command to check that all the CICS Runtime servers are running and that their messages conform to the Tuxedo documentation and this document.

When the mandatory servers `ARTADM`, `ARTTCPL`, and `ARTCNX` *only* are started, the following messages are displayed:

#### Listing 3-8 tadmin psr Command Example

---

```
# tadmin
...

> psr

Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
-----
BBL            200933      KIXR          0      1      50 ( IDLE )
ARTTCPL 00001.00101 TCP00      101      0      0 ( IDLE )
ARTCNX        QCNX015      GRP01        15      3      150 ( IDLE )

> quit
#
```

---

**Note:** The BBL Server is a Tuxedo System Server which can be compared to a CICS server on z/OS.

## Using the Tuxedo tadmin psc Commands

You can also check that the required Tuxedo services are running using the `tadmin psc` command.

These services should include the System Transactions managed by CICS Runtime:

- CSGM: The Good Morning Screen
- CESN: Sign On transaction
- CESF: Sign Off transaction

### Listing 3-9 tadmin psc Command Example

---

```
# tadmin
...

> psc
```

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	2	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	1	AVAIL
delsess	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
gensess	cnxsvc	ARTCNX	GRP01	15	KIXR	1	AVAIL
update	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
inquire	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL

```
> quit  
#
```

---

**Note:** From a certain point of view, this Tuxedo command is equivalent to the z/OS CICS system transaction `CEMT I TRAN (...)` which allows you to display the available transactions in a given z/OS CICS environment.

## Using the CSGM CICS Good Morning Transaction

Once this first audit is made, you can access CICS Runtime with a 3270 Terminal or Emulator using the following URL address `${HOSTNAME}:${TCPNETADDR}`.

Where:

### **`${HOSTNAME}`**

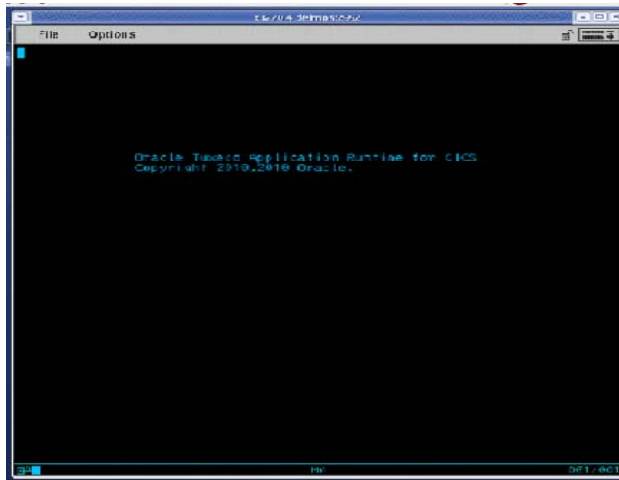
Is the System UNIX variable containing the name of the UNIX machine on which you are running CICS Runtime.

### **`${TCPNETADDR}`**

Is the port number specified by the `-n` parameter of the `ARTTCP` server in the Oracle Tuxedo `UBBCONFIG` file.

The following screen is displayed on a UNIX X11 Window after running the command `# x3270 deimos:2992:`

**Figure 3-1 Screen After Running the Command #3270 deimos:2992**



Successfully displaying this screen signifies you can continue implementing CICS applications using CICS Runtime.

## Initial Configuration of the CICS Runtime

# Implementing CICS Applications

This chapter contains the following sections:

- [Presentation of the z/OS Simple Application](#)
- [Verifying the CICS Application Installation](#)
- [Presentation of Simple Application on COBOL-IT / BDB](#)
- [Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances](#)
- [Implementing Asynchronous CICS Non-Delayed Transactions](#)
- [Implementing Asynchronous CICS Delayed Transactions](#)
- [Implementing CICS Application Using Temporary Storage \(TS\) Queues](#)
- [Implementing CICS Application Using Temporary Storage \(TS\) Queue POOL](#)
- [Implementing Distributed Program Link \(DPL\)](#)
- [Implementing CICS Common Work Area \(CWA\)](#)
- [Implementing a CICS Transaction Work Area \(TWA\)](#)
- [Implementing Integration with WebSphere MQ](#)
- [Implementing Using Multiple Session Management](#)
- [Implementing Using ART for CICS TCP/IP Socket Interface](#)

- [Implementing Transferring CICS Regions](#)
- [Implementing Intersystem Communication](#)
- [Implementing Submitting JCL/KSH Online](#)
- [Implementing ART for CICS Control Utility](#)
- [Implementing Printing CICS Runtime Applications Data](#)
- [Implementing Invoking Web Services from CICS Applications](#)
- [Implementing ART for CICS Application Server Customized Callback Support](#)
- [Implementing Resource-Based Authorization](#)
- [Implementing COBOL Program Debugging in CICS Runtime](#)
- [CICS Runtime Logs](#)
- [The CICS Runtime Server Logs](#)
- [Disabling and Enabling Programs](#)
- [CICS Runtime C Program Support](#)

# Presentation of the z/OS Simple Application

## Introduction

This application was initially developed on a z/OS platform implementing COBOL programs used in batch and CICS contexts with VSAM and QSAM files and DB2 tables.

Data was unloaded from z/OS and converted and reloaded on a UNIX platform using Oracle Tuxedo Application Rehosting Workbench.

The language components were converted or translated from z/OS to UNIX by Oracle Tuxedo Application Rehosting Workbench.

These components use two major Oracle Tuxedo Application components, Batch Runtime and CICS Runtime, to emulate the technical centralized features of their original z/OS environment. Here, we will focus on the particular case of the CICS Runtime implementing COBOL Programs using CICS statements and DB2 statements.



This Simple Application manages the customers of a company thru a set of classical functions like creation, modification and deletion.

## Description of the CICS Simple Application Components

All of the CICS components are declared with the same name, in the z/OS CICS CSD File. All of the resource declarations are made inside a z/OS CICS GROUP named `PJ01TERM`. This group is declared in the z/OS CICS LIST `PJ01LIST` used by CICS at start up to be automatically installed.

### Mapsets

**Table 4-1 Simple Application Mapsets**

<b>Name</b>	<b>Description</b>
RSSAM00	Customer maintenance entry menu
RSSAM01	Customer data inquiry screen
RSSAM02	Customer data maintenance screen (create, update and delete customer)
RSSAM03	Customer list screen

### Programs

**Table 4-2 Simple Application Programs**

<b>Name</b>	<b>Description</b>
RSSAT000	Customer maintenance entry program
RSSAT001	Customer data inquiry program
RSSAT002	Customer data maintenance program (new customer, update and delete customer)
RSSAT003	Customer list program

## Transactions Codes

**Table 4-3 Simple Application Transactions Codes**

Name	Description
SA00	Main entry transaction code (program RSSAT000)
SA01	Customer inquiry (program RSSAT001)
SA02	Customer maintenance (program RSSAT002)
SA03	Customer list (program RSSAT003)

## VSAM File

**Table 4-4 Simple Application VSAM File**

DDName	DataSetName	Description
ODCSF0	PJ01AAA.SS.VSAM.CUSTOMER	VSAM Main Customer File

## Configuring a Standard CICS Application With CICS Runtime

The first example uses the CICS Simple File-to-Oracle application which uses only a z/OS VSAM File converted into a UNIX Oracle Table.

In our example, all of the UNIX components resulting from platform migration are stored in the `trf` directory.

The COBOL programs and BMS mapsets should be compiled and available as executable modules in the respective directories `${HOME}/trf/cobexe` and `${HOME}/trf/MAP_TCP`.

## CICS Simple File-to-Oracle Application UNIX Components

### COBOL Program Files

The `${HOME}/trf/cobexe` directory contains the Simple Application CICS executable programs:

- `${HOME}/trf/cobexe/RSSAT000.gnt`
- `${HOME}/trf/cobexe/RSSAT001.gnt`

- `${HOME}/trf/cobexe/RSSAT002.gnt`
- `${HOME}/trf/cobexe/RSSAT003.gnt`

## The Mapset Files

The `${HOME}/trf/MAP_TCP` directory contains the Simple Application Data z/OS BMS mapsets compiled:

- `${HOME}/trf/MAP_TCP/RSSAM00.mpdef`
- `${HOME}/trf/MAP_TCP/RSSAM01.mpdef`
- `${HOME}/trf/MAP_TCP/RSSAM02.mpdef`
- `${HOME}/trf/MAP_TCP/RSSAM03.mpdef`

## CICS Runtime Configuration

For a standard application, in addition to the initial settings, the following CICS resources in the same Group must be implemented:

- Basic CICS transactions (synchronous and simultaneous).
- COBOL Programs without SQL statements, CICS TS queues.
- Mapsets.
- VSAM file (logical name and associated data accessors).

To configure these resources:

1. Declare these resources in their respective CICS Runtime Resource Configuration File.
2. Configure the CICS Runtime Tuxedo Servers Groups and Servers to manage these resources. See [Reference](#) for a full description of which configuration files are used with each server.

## Declaring CICS Resources to the CICS Runtime

Each resource is declared in the file corresponding to its type (program, transaction ...). Each resource defined in a resource file must belong to a group.

In the following examples using the CICS Simple File-to- Oracle Application, we will use the CICS Runtime Group name `SIMPAPP` and all our `*.desc` files will be located in the `${home}/trf/config/resources` directory.

**Note:** In these configuration files, each line beginning with a `"#"` is considered as a comment and is not processed by CICS Runtime

## Declaring CICS Transactions Codes

These declarations are made by filling the `transactions.desc` file for each transaction you have to implement.

For each transaction you have to declare in a csv format

1. The name of the transaction (mandatory).
2. The CICS Runtime Group name (mandatory).
3. A brief description of the transaction (optional, at least one blank).
4. The name of the program started by this transaction (mandatory).

In the File-to-Oracle Simple Application example, we have to declare four transactions: `SA00`, `SA01`, `SA02` and `SA03` in the `SIMPAPP` Group, starting the corresponding COBOL programs `RSSAT000`, `RSSAT001`, `RSSAT002` and `RSSAT003`.

Once filled, the `transactions.desc` file looks like this:

### Listing 4-1 Simple Application `transactions.desc` File

---

```
#Transaction Name ;Group Name ; Description ;Program Name
SA00;SIMPAPP; Home Menu Screen of the Simple Application;RSSAT000
SA01;SIMPAPP; Customer Detailed Information Screen of the Simple
Application;RSSAT001
SA02;SIMPAPP; Customer Maintenance Screen of the Simple
Application;RSSAT002
SA03;SIMPAPP; Customer List of the Simple Application;RSSAT003
```

---

## Declaring a CICS COBOL Program

All the programs used by the transactions previously declared, directly or indirectly through `EXEC CICS` statements like `LINK`, `XCTL`, `START` ... must be declared in the same Group.

These declarations are made in the `programs.desc` file for each program to implement.

For each program you have to declare in a csv format:

1. The name of the program (mandatory)
2. The CICS Runtime Group name (mandatory)
3. A brief description of the program (optional, at least one blank)
4. The language in which the program is written COBOL (default)

In our Simple Application example, the only programs needed are `RSSAT000`, `RSSAT001`, `RSSAT002` and `RSSAT003` which are all coded in the COBOL language

Once filled, the `programs.desc` file looks like this:

#### Listing 4-2 Simple Application programs.desc File

---

```
#PROGRAM;GROUP;DESCRIPTION;LANGUAGE;
RSSAT000;SIMPAPP; Home Menu Program of the Simple Application ;COBOL
RSSAT001;SIMPAPP; Customer Detailed Information Program of the Simple
Application ;COBOL
RSSAT002;SIMPAPP; Customer Maintenance Program of the Simple Application
RSSAT003;SIMPAPP; Customer List of the Simple Application ;COBOL
```

---

**Note:** Nothing is declared in the language field of `RSSAT002`, meaning that the `LANGUAGE` of this program is COBOL by default.

## Declaring CICS Mapsets

To converse with end-users thru 3270 terminals or emulators, declare to CICS Runtime all of the physical mapsets (\*.mpdef file) used in the COBOL programs previously defined thru the specific `EXEC CICS` statements described above in this document.

These declarations are made by filling the `mapsets.desc` file for each mapset you have to implement.

The input format of each of your mapset definitions must respect the following format description:

1. On the first free physical line, type the `[mapset]` keyword.

2. On the next line, enter the keyword `name=` followed by the name of your mapsets.
3. On the next line, enter the keyword `filename=` followed by the physical path of your physical mapsets (`.mpdef` file).

In our Simple Application example, the mapsets used in our COBOL programs are `RSSAM00`, `RSSAM01`, `RSSAM02` and `RSSAM03`.

Once filled, the `mapsets.desc` file looks like this:

### Listing 4-3 Simple Application `mapsets.desc` File

---

```
[mapset]
name=ABANNER
filename=<KIXDIR>/sysmap/abanner.mpdef [mapset]
name=RSSAM00
filename=<HOME>/demo/MAP_TCP/RSSAM00.mpdef
[mapset]
name=RSSAM01
filename=<HOME>/demo/MAP_TCP/RSSAM01.mpdef
[mapset]
name=RSSAM02
filename=<HOME>/demo/MAP_TCP/RSSAM02.mpdef
[mapset]
name=RSSAM03
filename=<HOME>/demo/MAP_TCP/RSSAM03.mpdef
```

---

**Note:** The `mapsets.desc` file does not accept UNIX variables, so a fully expanded path must be provided in this file.

- `<KIXDIR>`: must be replaced by the value of the `$(KIXDIR)` variable of the `~/profile`.

- <HOME>: must be replaced by the value of the  `${HOME}`  variable of the  `~/.profile` .

## Declaring ISAM Files Resulting From a z/OS VSAM File Conversion

Previously, before declaring one or more files to CICS Runtime, all the physical components, files, accessor programs, COBOL Copybooks etc. must have been generated by the Oracle Tuxedo Application Rehosting Workbench Data components.

Among all the components built or converted by the Oracle Tuxedo Application Rehosting Workbench Data components, only accessor programs on converted VSAM files are used by CICS Runtime. The reason is that, once migrated, no file can be directly accessed. The file can only be accessed indirectly through an accessor program dedicated to the management of this file (one and only one accessor program per source file).

The Simple Application uses only the CUSTOMER Oracle table, resulting from the Oracle Tuxedo Application Rehosting Workbench Data Conversion of the z/OS VSAM KSDS file  `PJ01AAA.SS.VSAM.CUSTOMER` .

So, for our File-to-Oracle application example, we have only one accessor,  `RM_ ODCSF0`  (RM for Relational Module), to declare to CICS Runtime.

**Note:**  `ODCSF0`  represents the logical name previously defined in CICS that pointed to the physical file name  `PJ01AAA.SS.VSAM.CUSTOMER` . Consequently, it is also the only file name known from the CICS COBOL program to access this file by  `EXEC CICS`  statements.

### To Declare the ISAM Migrated Files:

1. Modify the Tuxedo envfile adding a new variable, if not already present, describing all the VSAM/ISAM files used in the programs previously defined.

For our Simple Application example the following line must be entered, (for simplicity, we have located the file in the same place as the  `ubbconfig` ,  `envfile`  and  `tuxconfig`  files but this is not mandatory).

```
DD_VSAMFILE=${HOME}/trf/config/tux/desc.vsam
```

2. If the file does not exist, physically create the  `desc.vsam`  file at the indicated location.
3. Modify the  `desc.vsam`  file by adding a new line describing the different information fields used by the accessor in a "csv" format for each accessor/file used.

For our Simple Application example, the following line is entered:

#### Listing 4-4 Simple Application ISAM File Declaration

---

```
#DDname;Accessor;DSNOrganization;Format;MaxRecordLength;KeyStart;KeyLength  
ODCSF0;ASG_ ODCSF0;I;F;266;1;6
```

---

Where:

#### **ODCSF0**

Is the Data Description Name (logical name) used in the EXEC CICS Statements.

#### **RM\_ODCSF0**

Is the name of the accessor program managing the access to the Oracle table resulting from the data conversion of the former VSAM File .

#### **I**

The Data Set Name organization is indexed

#### **F**

Fixed, all the records have the same fixed length format.

#### **266**

Maximum record length.

#### **1**

Key position in the file (1 means first column or first character).

#### **6**

Key length.

## Modifying the CICS Runtime Tuxedo Servers

To manage CICS application transactions, in addition to the servers previously defined:

1. Implement the CICS Runtime Tuxedo Server `ARTSTRN`.

This server manages only basic CICS Runtime transactions, those that are the most often used: synchronous (not delayed) and simultaneous (not only one at a time).

2. Indicate to CICS Runtime to start only the transactions belonging to the `SIMPAPP` CICS Runtime Group name.



The following example of a \*SERVERS section of the Tuxedo ubbconfig file shows the configuration of a ARTSTRN server.

#### Listing 4-5 Simple Application CICS Runtime Server Tuxedo Configuration

---

```
*SERVERS
...
ARTSTRN    SRVGRP=GRP02
           SRVID=20
           CONV=Y
           MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y
           CLOPT=" -o /home2/work9/demo/Logs/TUX/sysout/stdout_strn -e
/home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -s KIXR -l SIMPAPP "
...
```

---

Where

#### **\*SERVERS**

Tuxedo ubbconfig Keyword indicating a Server Section definition.

#### **SRVGRP**

Is the Tuxedo Group Name to which ARTSTRN belongs.

#### **SRVID**

Is the identifier of a Tuxedo Server of ARTSTRN.

#### **CONV=Y**

Indicates that this server operates in a conversational mode.

#### **MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

#### **REPLYQ=Y**

Indicates that this server will respond.

**RQADDR=QCNX015**

Name of the Tuxedo queue used for the responses.

**CLOPT**

Is a quoted text string passed to the server containing its parameters.

-o

Indicates the file used for the standard output messages of the server.

-e

Indicates the file used for the error output messages of the server.

-r

Is a Tuxedo parameter used to provide statistical reports.

-s KIXR

Indicates the CICS Runtime name where the KIXR transaction is run.

-l SIMAPP

Indicates that only the transaction of the SIMAPP group are to be selected.

## Modifying the CICS Runtime Tuxedo Servers Groups

To be started, the `ARTSTRN` server must be defined in a Tuxedo Server Group previously defined (and not commented) in the `ubbconfig` file.

In our example, `ARTSTRN` belong to the Tuxedo Server Group `GRP02` (`SRVGRP=GRP02`).

### Listing 4-6 Simple Application CICS Runtime Tuxedo Servers Groups Example:

---

```
*GROUPS
...
GRP02
    GRPNO=12
    ENVFILE="/home2/work9/demo/config/tux/envfile"
    TMSNAME="TMS_ORA"
...
```

---

Where

**\*GROUPS**

Tuxedo ubbconfig Keyword indicating a Server Section Group section definition.

**GRPNO=**

Tuxedo Group.

**ENVFILE=**

Path of the Tuxedo envfile.

**TMSNAME=**

Name of the Tuxedo Transaction Manager Server executable.

## Verifying the CICS Application Installation

### Using the Tuxedo tadmin psr Commands

Enter the Tuxedo `tadmin psr` command to check that all of the CICS Runtime required servers (ARTTCPL, ARTCNX, and ARTSTRN) are running and that their messages conform to the Tuxedo documentation and this document.

#### Listing 4-7 tadmin psr Simple Application Installation Check

---

```
# tadmin
...

> psr
```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
-----	-----	-----	--	-----	-----	-----	-----
BBL	200933	KIXR	0	2	100	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	2	100	( IDLE )	
ARTSTRN	QKIX110	GRP02	20	6	300	( IDLE )	

```
> quit
#
```

---

## Using the Tuxedo tadmin psc Commands

Another possible check can be made by entering the Tuxedo `tadmin psc` command to display all the different Tuxedo Services running.

In addition to the CICS Runtime System transactions/services (CSGM, CESN, CESF ...), you can now see the transaction codes of your CICS Runtime application SA00, SA01, SA02 and SA03

### Listing 4-8 tadmin psc Simple Application Installation Check

---

```
# tadmin
...
```

```
> psc
```

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
-----	-----	-----	-----	--	-----	-----	-----
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	1	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	1	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	3	AVAIL
SA02	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	3	AVAIL

```
> quit
```

```
#
```

---

## Using the CICS Runtime Application

Before using the CICS application, you have to populate the ISAM files accessed by your application. Then, access CICS Runtime with a 3270 Terminal or Emulator, with a UNIX x3270 command. It should be:

```
# x3270 ${HOSTNAME} : ${TCPNETADDR}
```

Where:

### **\${HOSTNAME}**

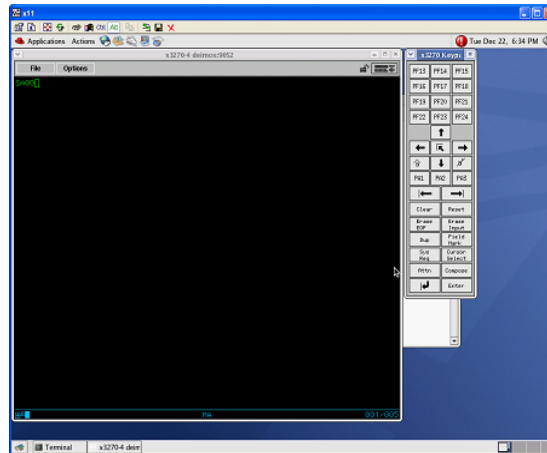
Is the System UNIX variable containing the name of the UNIX machine on which you are running CICS Runtime.

### **\${TCPNETADDR}**

Is the port number for your UNIX 3270 emulator set up by your Tuxedo Administrator at installation time in the ubbconfig file.

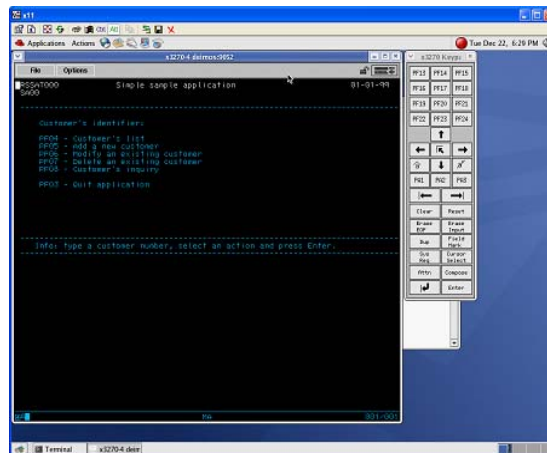
1. You will receive the Good Morning Message.
2. Clear it by pressing the `Clear` key of your 3270 emulator keypad.
3. Type the main transaction code `SA00` (of your CICS Runtime application) in the top left corner:

Figure 4-1 Simple Application transaction Code Entry



4. The main menu of the application is displayed:

Figure 4-2 Simple Application Main Menu



5. Navigate through the screens of the application to check that they are displayed without errors.

## Presentation of Simple Application on COBOL-IT / BDB

Based on BDB with XA protocol, the CICS COBOL programs compiled by COBOL-IT can access the indexed ISAM files which are converted from Mainframe VSAM files through the ART Workbench. The following sections describes the configurations should be done in ART CICS Runtime to enable this application.

### Configuring ubbconfig File in CICS Runtime

Add the MRM parameter in the group entry of \*GROUPS and \*RMS section in Tuxedo ubbconfig file. See the following example:

#### Listing 4-9 Adding MRM Parameter in ubbconfig File Example

---

```
*GROUPS

GRP02

GRPNO=12

MRM=Y

*RMS

MRMG_RM1

SRVGRP=GRP02

RMID=15

TMSNAME="TMS_BDB"

OPENINFO="BERKELEY-DB:/home2/work10/data"
```

---

Where:

#### **\*GROUPS**

Tuxedo ubbconfig keyword indicating the definitions of Servers Groups.

#### **GRPNO**

Indicates the Tuxedo Group.

**MRM= Y**

Indicates that this server group can support multiple resource managers.

**\*RMS**

Tuxedo ubbconfig keyword indicating the definitions of resource managers.

**MRMG\_RM1**

Indicates the logical name of RMS entry.

**SRVGRP**

Indicates the name of the group associated with this RM.

**RMID**

Indicates the unique ID number of this RM in the group. ID number must be between 1 and 31.

**TMSNAME**

Indicates the name of the transaction manager server associated with the group specified by SRVGRP.

**OPENINFO**

Indicates the resource manager dependent information needed when opening resource manager for the associated group.

## Building BDB TMS Server

To build the BDB TMS server, add the following lines to `$TUXDIR/udataobj/RM`:

```
BDB_HOME=/opt/cobol-it-64-bdb
```

```
BERKELEY-DB:db_xa_switch:-L/opt/cobol-it-64-bdb/lib -ldb-5
```

After updating the RM file, execute the following command to build TMS server for BDB:

```
buildtms -v -r BERKELEY-DB -o $APPDIR/TMS_BDB
```

## Exporting Variables Before Booting Up ART Servers

Export the following variables explicitly before booting up the ART servers:

- DD\_VSAMFILE

```
export DD_RBDB02=${DATA}/MTWART.ES.SFI.RCIBDB02.BDB0122
```

RBDB02 is the logical file name.

- COB\_ENABLE\_XA



```
export COB_ENABLE_XA=1
```

## Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances

In some particular cases, the number of transactions bearing the same transaction code running simultaneously has to be limited, for performance constraints for example.

On z/OS, this limit cannot be defined in the transaction resource itself but is defined in a distinct resource named `TRANCLASS` (transaction class) that contains a specific `MAXACTIVE` parameter describing the maximum number of concurrent instances of the same transaction.

To link a transaction to a transaction class, to inherit its parameters, especially the `MAXACTIVE` parameter, the z/OS CICS transaction resource has a `TRANCLASS` field containing the name of the `TRANCLASS` resource.

This instance management is performed differently on UNIX with CICS Runtime. The maximum number of transactions running concurrently is defined by the number of servers offering the same transaction. This maximum number and the minimum number are indicated respectively in the `MAX` and `MIN` parameters of the `ARTSTRN` definition in the `*SERVERS` section of the Tuxedo file `ubbconfig`.

It means that the `maxactive` parameter is not taken in account to manage these limits except in the following very particular case:

### The Special Case of Transaction Classes With `MAXACTIVE=1`

The `MAXACTIVE=1` is really an exception in this management because it indicates that no concurrent transaction belonging to these kind of transaction classes can be run simultaneously.

To manage this very particular case of sequential transactions, a Tuxedo CICS Runtime feature must be configured

### Modification of the `ubbconfig` File for Sequential Transactions

All of the transactions linked to transactions classes with a `MAXACTIVE` superior or equal to 2 are managed by the CICS Runtime Tuxedo Server `ARTSTRN` and do not required modifying anything else. For the transactions with a `MAXACTIVE` parameter set to 1, an CICS Runtime Tuxedo Server named `ARTSTR1` is dedicated to their specific management.

To activate this server, modify the ubbconfig file to add this server in the \*SERVERS section:

**Listing 4-10 Adding a ARTSTR1 Server to ubbconfig**

---

```
*SERVERS
...
ARTSTR1    SRVGRP=GRP02
           SRVID=200
           CONV=Y
           MIN=1 MAX=1
           CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_str1 -e
/home2/work9/demo/Logs/TUX/sysout/stderr_str1 -r -- -s KIXR -l SIMPAPP"
...
```

---

Where:

**\*SERVERS**

Tuxedo ubbconfig Keyword indicating a Server Section definition.

**SRVGRP**

Is the Tuxedo Group Name to which ARTSTR1 belongs.

**SRVID**

Is the identifier of a ARTSTR1 Tuxedo Server.

**CONV=Y**

Indicates that this server operates in a conversational mode.

**MIN=1 and MAX=1**

Are mandatory and indicate that only one instance of this server must run.

**CLOPT**

Is a quoted text string passed to the server containing the parameters:

-o

Indicates the file used for the standard output messages of the server.

- e  
Indicates the file used for the error output messages of the server.
- r  
Is a Tuxedo parameter used to produce statistical reports.
- s  
KIXR indicates the CICS Runtime name where the KIXR transaction is run.
- I SIMAPP  
Indicates that only the transaction of the SIMAPP group are to be selected.

**Note:** All of the CICS Runtime Transaction Servers (ARTSTRN, ARTSTR1, ARTATR1 and ARTATR1) share the same CICS Runtime Transaction Group Servers, no modifications are required to the ubbconfig Server Group Section (\*GROUPS).

## Modifying the tranclasses.desc File

For ART CICS, concurrent transactions do not really need to be bound to transactions classes with MAXACTIVE parameters superior or equal to two because parallelism is the default behavior.

For sequential transactions, it is mandatory because it is the only way to declare these transactions to CICS Runtime. Declare specific transaction classes defined with a MAXACTIVE=1 parameter. Like the other CICS Runtime resources, this one must belong to an CICS Runtime Group name. For each TRANCLASS, declare in a csv format:

1. The name of the transaction class (mandatory)
2. The CICS Runtime Group name (mandatory)
3. A brief description of the transaction class (optional, at least one blank)
4. The maximum number of the same transaction to RUN (MAXACTIVE).

**Note:** The MAXACTIVE parameter should be understood like a binary switch:

- MAXACTIVE=1 <=> Sequential transaction class (mandatory).
- MAXACTIVE>1 (all the values are at this step equivalent) <=> Concurrent transaction (optional).

Examples:

```
TRCLASS1;SIMPAPP ; Traclass with maxactive set to 1; 1
TRCLASS2;SIMPAPP ; Traclass with maxactive set to 2; 2
TRCLAS10;SIMPAPP ; Traclass with maxactive set to 10; 10
```

The first tranclass `TRCLASS1` has its `maxactive` parameter equal to 1, indicating that all the transactions belonging to this tranclass must be managed sequentially by the `ARTSTR1`.

The two last tranclasses, `TRCLASS2` and `TRCLASS10`, are in fact similar because their `maxactive` parameters are superior to 1 indicating that the transactions belonging to these tranclasses can run concurrently managed by the `ARTSTRN` server.

**Note:** These two last definitions are optional. Their absence has the same meaning.

### Modifying the `transactions.desc` File

In addition to the first four mandatory fields of this csv format file (Transaction name, Group name, Description, Program name), you must add a twelfth field: `TRANCLASS` (Transaction Class name).

The `TRANCLASS` field must be separated from the Program field by eight semicolon characters (;) with at least one blank between each of them.

In our example, let us suppose that the CICS Runtime Simple Application must have the following `MAXACTIVE` limits:

- SA00: `MAXACTIVE=0`
- SA01: `MAXACTIVE=1`
- SA02: `MAXACTIVE=2`
- SA03: `MAXACTIVE=10`

Then these transactions must be linked to the following tranclasses that we have previously defined:

- SA00: none
- SA01: `TRCLASS1`
- SA02: `TRCLASS2`
- SA03: `TRCLASS10`

Once modified, the `transactions.desc` file will look like this:

#### Listing 4-11 Example `transactions.desc` File

---

```
#Transaction Name ;Group Name ; Description ;Program Name
```

## Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances

```
SA00;SIMPAPP; Home Menu Screen of the Simple Application;RSSAT000
SA01;SIMPAPP; Customer Detailed Information Screen of the Simple ;
Application;RSSAT001; ; ; ; ; ; ;TRCLASS1
SA02;SIMPAPP; Customer Maintenance Screen of the Simple
Application;RSSAT002; ; ; ; ; ; ; TRCLASS2
SA03;SIMPAPP; Customer List of the Simple Application;RSSAT003; ; ; ; ; ;
; TRCLASS10
```

---

### Notes:

- No modification is made to SA00 meaning that no transaction class is associated with this transaction code. It means that this transaction is not associated with a MAXACTIVE=1 parameter and so is not sequential.
- SA02 and SA03 are associated to transaction classes, respectively TRCLASS2 and TRCLASS10, defined with MAXACTIVE >= 2. Knowing that these transactions are not required, the result would be the exactly the same if SA02 and SA03 were defined like SA00 without transaction classes.
- SA01, which can run sequentially, is the only one where the transaction class field is mandatory. Verify that its associated transaction class, TRCLASS1, is really defined with a MAXACTIVE=1.

## Checking the ARTSTR1 Configuration

### Using the Tuxedo tadmin psr Commands

The ARTSTR1, is shown below:

#### Listing 4-12 Checking the ARTSTR1 Server with the tadmin psr Commands

---

```
# tadmin
...

> psr
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
```

```

-----
ARTSTR1      00012.00200 GRP02      200      0      0 ( IDLE )
BBL          200933      KIXR          0      3      150 ( IDLE )
ARTTCPL 00001.00101 TCP00          101      0      0 ( IDLE )
ARTCNX       QCNX015      GRP01          15      0      0 ( IDLE )
ARTSTRN      QKIX110      GRP02          20      0      0 ( IDLE )

> quit
#

```

---

## Using the Tuxedo tadmin psc Commands

No new service or transaction should appear.

In our example where ARTSTRN was the only server running, we can see that nothing changed when ARTSTR1 is also activated.

### Listing 4-13 Checking the ARTSTRN Server with the tadmin psc Commands

---

```

# tadmin
...

> psc

Service Name Routine Name Prog Name Grp Name ID Machine # Done Status
-----
authfail      cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL
CESF          cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL
CESN          cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL
CSGM          cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL
disconnect    cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL

```

```

connect      cnxsvc      ARTCNX      GRP01      15      KIXR      0 AVAIL
SA03        kixsvc      ARTSTRN     GRP02      20      KIXR      0 AVAIL
SA02        kixsvc      ARTSTRN     GRP02      20      KIXR      0 AVAIL
SA01        kixsvc      ARTSTRN     GRP02      20      KIXR      0 AVAIL
SA00        kixsvc      ARTSTRN     GRP02      20      KIXR      0 AVAIL

> quit
#

```

---

## Implementing Asynchronous CICS Non-Delayed Transactions

These transactions are launched by specific CICS EXEC CICS START TRANSID requests coded in the CICS programs that are not using DELAY or TIME parameters to delay their execution.

If at least one of your programs contains this kind of statement, install, and activate some new features of CICS Runtime Tuxedo Servers without changing any other settings.

### Modifying the Tuxedo ubbconfig File to Manage Asynchronous Transactions

The file is modified in the same manner as for the ARTSTRN and the ARTSTR1 servers, except the "s" (synchronous) character used to prefix the name of these servers should be replaced by the "a" (asynchronous) character.

### Using Parallel Asynchronous Transactions

To use parallel asynchronous transactions, with a MAXACTIVE parameter strictly superior to one, the dedicated server is the ARTATRN. Please refer to the section describing the installation of the ARTSTRN server to install the atrn\_server.

To check your settings you can use also the tadmin psr and psc commands.

For the Simple Application example we can see that:

- The `psr` command shows that a new server is running `ARTATRN`.
- The `psc` command shows that five new services are running, one is dedicated to the asynchronous transaction while each synchronous transaction (`SA00` to `SA03`) is duplicated (`ASYNCSA00` to `ASYNCSA03`) to allow them to run in an asynchronous mode.

**Listing 4-14 tadmin Commands Showing Parallel Asynchronous Transactions**

---

```
# tadmin
...

> psr
Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
-----
ARTSTR1        00012.00200 GRP02          200    0      0 ( IDLE )
BBL            200933      KIXR           0     4     200 ( IDLE )
ARTTCPL        00001.00101 TCP00          101    0      0 ( IDLE )
ARTCNX         QCNX015     GRP01          15     0      0 ( IDLE )
ARTSTRN        QKIX110     GRP02          20     0      0 ( IDLE )
ARTATRN        QKIXATR     GRP02          30     0      0 ( IDLE )

> psc
Service Name Routine Name Prog Name  Grp Name  ID   Machine # Done Status
-----
authfail      cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
CESF          cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
CESN          cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
CSGM          cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
disconnect    cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
connect       cnxsvc     ARTCNX     GRP01     15    KIXR    0 AVAIL
```



```
SA03      kixsvc      ARTSTRN   GRP02     20      KIXR      0 AVAIL
SA02      kixsvc      ARTSTRN   GRP02     20      KIXR      0 AVAIL
SA01      kixsvc      ARTSTRN   GRP02     20      KIXR      0 AVAIL
SA00      kixsvc      ARTSTRN   GRP02     20      KIXR      0 AVAIL
ASYNC_QUEUE ASYNC_QUEUE ARTATR1   GRP02     30      KIXR      0 AVAIL
ASYNC_SA03 atrsvc      ARTATR1   GRP02     30      KIXR      0 AVAIL
ASYNC_SA02 atrsvc      ARTATR1   GRP02     30      KIXR      0 AVAIL
ASYNC_SA01 atrsvc      ARTATR1   GRP02     30      KIXR      0 AVAIL
ASYNC_SA00 atrsvc      ARTATR1   GRP02     30      KIXR      0 AVAIL
```

```
> quit
```

```
{deimos:work9}-/home2/work9/demo/config/tux#{deimos:work9}-/home2/work9/de
mo/config/tux#
```

## Using Non-Parallel Asynchronous Transactions

To use non-parallel asynchronous transactions, with a `MAXACTIVE` parameter exactly equal to one, the dedicated server is `ARTATR1`.

Please refer to the section describing the reasons and the installation of the `ARTSTR1` server to install the `ARTSTR1` server.

To check your setting, you can use also the Tuxedo `tadmin psr` and `psc` commands

For the Simple Application example we can see that:

- The `psr` command shows that a new server is running `ARTATR1`.
- The `psc` command shows that no new services are running.

### Listing 4-15 tadmin Commands Showing non-parallel Asynchronous Transactions

```
# tadmin
```

```
...
```

## Implementing CICS Applications

> psr

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
ARTATR1	00012.00300	GRP02	300	0	0	( IDLE )	
ARTSTR1	00012.00200	GRP02	200	0	0	( IDLE )	
BBL	200933	KIXR	0	4	200	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP02	20	0	0	( IDLE )	

> psc

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA02	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL

> quit

#

---

# Implementing Asynchronous CICS Delayed Transactions

ART CICS Runtime supports two methods for implementing asynchronous CICS delayed transactions launched using `EXEC CICS START TRANSID` requests:

- [Implementing Asynchronous Transactions With ARTSRM Server](#) (Recommended)
- [Implementing Asynchronous Transactions With /Q](#)

## Implementing Asynchronous Transactions With ARTSRM Server

On z/OS, there are some time-related CICS START API options can be used to start a transaction at a specified time or after a specified interval, such as `AT`, `TIME`, `AFTER`, and `INTERVAL`. ART CICS Runtime provides a server, `ARTSRM`, for implementing these options. For more information, refer to [ARTSRM Configuration](#) in [Oracle Tuxedo Application Runtime for CICS Reference Guide](#).

To activate this server, configure `ARTSRM` in the `*SERVERS` section in the `UBBCONFIG` file. You can configure a set of `ARTSRM` servers only if they are in the same group for each CICS region. Following is an example.

### Listing 4-16 Example of Configuring ARTSRM in UBBCONFIG

---

```
*SERVERS
...
ARTSRM
    SRVGRP=ARTGRP
    SRVID=500
    RESTART=Y
    MAXGEN=5
    GRACE=3600
```

```
CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_srm -e  
/home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -s KIXR -l SIMPAPP"
```

---

**Note:** Although implementing asynchronous transactions with /Q is still supported, when the `START TRANID/CANCEL` command is invoked, the request is submitted to the `TRANCTL_[SYSID]` service advertised by ARTSRM firstly. If the call gets the `TPNOENT` fail code, then use /Q to redispach the request.

## Implementing Asynchronous Transactions With /Q

Asynchronous transactions are launched when `ASYNC_QSPACE` for `EXEC START` is set with option `INTERVAL` or `PROTECT`.

In this case, the transaction request is deposited into a Oracle Tuxedo /Q Queue, and when the time is ready, the transaction will be automatically invoked.

For this feature to be available, these components must be configured:

1. An Oracle Tuxedo /Q Queue Space named `ASYNC_QSPACE`
2. An Oracle Tuxedo /Q Queue named `ASYNC_QUEUE` in `ASYNC_QSPACE`.
3. The `TMQUEUE` and `TMQFORWARD` servers dedicated to these asynchronous transactions.

## Creating the Tuxedo /Q

CICS Runtime provides a UNIX script that creates all the Tuxedo /Q components:  
`mkqmconfig.sh`.

1. Before using the script, define and export in your UNIX `~/./profile` file:
  - The `QMCONFIG` variable `QMCONFIG-` containing the full directory path that stores the Tuxedo /Q Queue Space `ASYNC_QSPACE`.
  - The `KIX_QSPACE_IPCKEY` variable - containing the IPC Key for the Queue Space.

Examples of `~/./profile` variables and values:

```
export QMCONFIG=${HOME}/trf/config/tux/kixqspace  
export KIX_QSPACE_IPCKEY=200955
```

2. Execute `mkqmconfig.sh` from the command line to create the Tuxedo /Q features.

## Modifying the Tuxedo ubbconfig File to Manage the Tuxedo /Q Queue

1. The GQUEUE Server Group must be added to the ubbconfig file in the \*GROUP section.

### Listing 4-17 Simple Application Tuxedo Queue ubbconfig Example

---

```
*GROUPS
...
# /Q
GQUEUE      GRPNO=1000
            TMSNAME=TMS_QM TMSCOUNT=2

OPENINFO="TUXEDO/QM:/home2/work9/demo/config/tux/kixqspace:ASYNC_QSPACE"
...
```

---

Where:

**\*GROUPS**

Tuxedo ubbconfig Keyword indicating definitions of Servers Groups.

**GRPNO=**

Tuxedo Group.

**TMSCOUNT=**

Number of Tuxedo Transaction Manager Servers.

**TMSNAME**

Name of the Tuxedo Transaction Manager Server executable.

**OPENINFO=**

Indicates to the Tuxedo /Q Transaction Manager QM, the QSPACE name to manage and its UNIX absolute path.

2. Then, two servers, TMQUEUE and TMQFORWARD, must be added to the ubbconfig file in the \*SERVERS section.

**Listing 4-18 Simple Application ubbconfig TMQUEUE and TMQFORWARD Example**

---

```
*SERVERS
...
# /Q
TMQUEUE      SRVGRP=GQUEUE
              SRVID=1010
              GRACE=0 RESTART=Y CONV=N MAXGEN=10
              CLOPT="-s ASYNC_QSPACE:TMQUEUE -- "
TMQFORWARD
              SRVGRP=GQUEUE
              SRVID=1020
              GRACE=0 RESTART=Y CONV=N MAXGEN=10
              CLOPT="-- -n -i 2 -q ASYNC_QUEUE"
...
```

---

Where:

**\*SERVERS**

Tuxedo ubbconfig Keyword indicating a Server Section definition.

**SRVGRP**

Is the Tuxedo Group Name which the server belongs to.

**SRVID**

Is the identifier of a Tuxedo Server.

**MAXGEN=10**

Specifies that the process can have up to 10 server restarts.

**GRACE=0**

Means there is no limit interval to contain the number of server restarts.

**CONV=N**

Indicates that this server operates in a non-conversational mode.

**CLOPT**

Is a quoted text string passed to the server containing its parameters.

Using the `tmadmin psr` and `psc` commands check that four new servers and two new services are running:

**Listing 4-19 Simple Application TMQUEUE and TMQFORWARD tmadmin Example**

---

```
# tmadmin
...

> psr
```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current Service
-----	-----	-----	---	-----	-----	-----
ARTATR1	00012.00300	GRP02	300	0	0	( IDLE )
ARTSTR1	00012.00200	GRP02	200	0	0	( IDLE )
BBL	200933	KIXR	0	4	200	( IDLE )
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )
TMS_QM	GQUEUE_TMS	GQUEUE	30001	0	0	( IDLE )
TMS_QM	GQUEUE_TMS	GQUEUE	30002	0	0	( IDLE )
TMQUEUE	01000.01010	GQUEUE	1010	0	0	( IDLE )
TMQFORWARD	01000.01020	GQUEUE	1020	0	0	( IDLE )
ARTSTRN	QKIX110	GRP02	20	0	0	( IDLE )
ARTATR1	QKIXATR	GRP02	30	0	0	( IDLE )

```
> psc
```

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
--------------	--------------	-----------	----------	----	---------	--------	--------

## Implementing CICS Applications

```
-----
TMS          TMS          TMS_QM      GQUEUE 30001      KIXR      0 AVAIL
TMS          TMS          TMS_QM      GQUEUE 30002      KIXR      0 AVAIL
ASYNC_QSPACE TMQUEUEE      TMQUEUEE      GQUEUE 1010      KIXR      0 AVAIL
authfail     cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
CESF         cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
CESN         cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
CSGM         cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
disconnect   cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
connect      cnxsvc          ARTCNX        GRP01    15        KIXR      0 AVAIL
SA03         kixsvc          ARTSTRN       GRP02    20        KIXR      0 AVAIL
SA02         kixsvc          ARTSTRN       GRP02    20        KIXR      0 AVAIL
SA01         kixsvc          ARTSTRN       GRP02    20        KIXR      0 AVAIL
SA00         kixsvc          ARTSTRN       GRP02    20        KIXR      0 AVAIL
ASYNC_QUEUE ASYNC_QUEUEE    ARTATR        GRP02    30        KIXR      0 AVAIL
ASYNC_SA03   atrsvc          ARTATR        GRP02    30        KIXR      0 AVAIL
ASYNC_SA02   atrsvc          ARTATR        GRP02    30        KIXR      0 AVAIL
ASYNC_SA01   atrsvc          ARTATR        GRP02    30        KIXR      0 AVAIL
ASYNC_SA00   atrsvc          ARTATR        GRP02    30        KIXR      0 AVAIL

> quit
#
```

---



# Implementing CICS Application Using Temporary Storage (TS) Queues

These transactions use CICS programs containing EXEC CICS requests relative to CICS Temporary Storage Queues.

The statements used are EXEC CICS WRITEQ TS ... END-EXEC, EXEC CICS READQ TS ... END-EXEC, EXEC CICS DELETEQ TS ... END-EXEC.

If at least one of your programs contains one of these statements, install and activate the new features of CICS Runtime without changing your other settings.

To manage TS Queues, activate the ARTTSQ CICS Runtime Tuxedo Server.

- To activate this server, add this server to the \*SERVERS section of the Tuxedo ubbconfig file:

## Listing 4-20 Activating the ARTTSQ in the ubbconfig File

---

```
*SERVERS
...
ARTTSQ      SRVGRP=GRP02
            SRVID=40
            MIN=1 MAX=1
            CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_tsq -e
/home2/work9/demo/Logs/TUX/sysout/stderr_tsq -r -- -s KIXR -l SIMPAPP"
...
```

---

Where:

### **\*SERVERS**

Tuxedo ubbconfig Keyword indicating a Server Section definition.

### **SRVGRP**

Is the Tuxedo Group Name to which ARTTSQ belongs.

**SRVID**

Is the identifier of a Tuxedo Server of ARTTSQ.

**MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

**CLOPT**

Is a quoted text string passed to the server containing its parameters:

-o

Indicates the following file is used for the standard output messages of the server.

-e

Indicates the following file is used for the error output messages of the servers.

-r

Is a Tuxedo parameter used to have statistical reports.

-s KIXR

Indicates the CICS Runtime name where the transaction runs is KIXR.

-l SIMAPP

Indicates that only the components of the SIMAPP group are to be selected at start up.

Use the Tuxedo `tmadmin psr` and `pssc` commands to check that the server is running and that six new services are published:

**Listing 4-21 Checking ARTTSQ Server and Services are Running**

---

```
# tmadmin  
...
```

```
> psr
```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
-----	-----	-----	--	-----	-----	-----	-----
ARTATR1	00012.00300	GRP02	300	0	0	( IDLE )	
ARTSTR1	00012.00200	GRP02	200	0	0	( IDLE )	
BBL	200933	KIXR	0	3	150	( IDLE )	

## Implementing CICS Application Using Temporary Storage (TS) Queues

```

ARTTCPL      00001.00101 TCP00          101      0          0 ( IDLE )
ARTCNX       QCNX015     GRP01           15      0          0 ( IDLE )
ARTSTRN      QKIX110     GRP02           20      0          0 ( IDLE )
ARTTSQ       00012.00040 GRP02           40      0          0 ( IDLE )
  
```

> psc

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA02	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
TSM00004_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00003_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00002_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00001_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00000_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSQUEUE	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL

> quit

```
{deimos:work9}-/home2/work9/demo/config/tux#
```

## Implementing Unrecoverable TS Queues

For unrecoverable TS Queues, no integrity is guaranteed by CICS Runtime concerning their content. For example, if an abend occurs at any point during a CICS transaction, work done on this TS is not rolled-back to the last consistency point.

TS Queues are stored in a sequential file in a dedicated directory defined in the `KIX_TS_DIR` UNIX environment variable. This variable is defined and then exported from the `~/.profile` UNIX System File:

```
KIX_TS_DIR=${HOME}/trf/KIXTSDIR
```

Modify the Tuxedo `ubbconfig` file to activate the new `ARTTSQ` server dedicated to their management.

## Implementing Recoverable TS Queues

For these TS Queues, CICS Runtime guarantees the integrity of their content. For example, if an abend occurs at any point during a CICS transaction, they are rolled-back to the last consistency point, if all is in order, their content is committed to become a new consistency point. These TS Queues are stored in Oracle Tables to benefit from the RDBMS integrity management.

Concerning the TS Queue, there is an enhanced behavior for reading a recoverable TS Queue.

On source CICS z/OS, CICS enqueueing is not invoked for `READQ` TS commands, thereby making it possible for one task to read a temporary storage queue record while another is updating the same record. To avoid this, use explicit enqueueing on the temporary storage queues so that concurrent executing tasks can read and change queues with the same temporary storage identifier.

This behavior also allows one transaction to see or read a record freshly written in a recoverable TS Queue, even before it is committed, and after its rollback.

On target we don't have this limitation, but in particular:

- A reading transaction is not able to see a record that is just added and not yet committed.
- A reading transaction is not able to see a modification to the record that is not yet committed.

## To Use Recoverable TS Queues

To use recoverable TS Queues you need to define an Oracle Table to contain the TS Queues. CICS Runtime provides a UNIX script to create all these tables: `crtstable_Oracle`.

1. Before using the script define and export from your UNIX `~/profile` file

- The `ORA_USER` variable containing the user ID used to connect to Oracle.
- The `ORA_PASSWD` variable containing the associated password.

Examples of `~/profile` variables and values:

```
export ORA_USER="Oracle_User_1"
export ORA_PASSWD="Oracle_Pswd_1"
```

2. Once the variables have been set, execute the `crtstable_Oracle` script.

3. Then, modify the Tuxedo `ubbconfig` file to modify the Server Group used by ARTTSQ to establish the connection to Oracle in the `*GROUPS` section.

### Listing 4-22 Example of the `*GROUP` Section of the Tuxedo `ubbconfig` File Concerning the Derver Group `GRP02` used by the ARTTSQ Server.

---

```
*GROUPS
...
GRP02
    GRPNO=12
    ENVFILE="/home2/work9/demo/config/tux/envfile"
    TMSNAME="TMS_ORA"
    OPENINFO="Oracle_XA:Oracle_XA+Acc=P/work9/work9+SesTm=600+LogDir=/home2/work9/demo/Logs/TUX/xa+DbgFl=0x20"
...
```

---

Where:

#### **\*GROUPS**

Tuxedo `ubbconfig` Keyword indicating definitions of Servers Groups.

**GRPNO=**

Tuxedo Group number.

**TMSNAME=**

Name of the Tuxedo Transaction Manager Server executable.

**OPENINFO=**

Parameters send to the Oracle\_XA Manager.

4. Use the Tuxedo psr and psc commands to check that Oracle is available; three new servers and three new services should be indicated:

**Listing 4-23 Simple Application Check For Recoverable TS Queues**

---

```
# tadmin
...

> psr

Prog Name      Queue Name  Grp Name      ID RqDone Load Done Current Service
-----
ARTATR1        00012.00300 GRP02          300      0      0 ( IDLE )
ARTSTR1        00012.00200 GRP02          200      0      0 ( IDLE )
BBL            200933      KIXR           0        4     200 ( IDLE )
ARTTCPL        00001.00101 TCP00          101      0      0 ( IDLE )
TMS_ORA        GRP02_TMS   GRP02         30001     0      0 ( IDLE )
TMS_ORA        GRP02_TMS   GRP02         30002     0      0 ( IDLE )
TMS_ORA        GRP02_TMS   GRP02         30003     0      0 ( IDLE )
ARTCNX         QCNX015     GRP01           15       0      0 ( IDLE )
ARTSTRN        QKIX110     GRP02           20       0      0 ( IDLE )
ARTTSQ         00012.00040 GRP02           40       0      0 ( IDLE )

> psc
```

## Implementing CICS Application Using Temporary Storage (TS) Queues

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
-----	-----	-----	-----	--	-----	-----	-----
TMS	TMS	TMS_ORA	GRP02	30001	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30002	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30003	KIXR	0	AVAIL
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA02	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
TSM00004_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00003_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00002_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00001_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00000_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSQUEUE	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL

> quit

#

---

# Managing TD Queue Intrapartitions

## Presentation of the Mechanism on Source Platform

### Transient Data Control

The CICS transient data control facility provides a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected data, specified in the application program can be routed to or from predefined symbolic transient data queues: either intrapartition or extrapartition.

Transient data queues are intrapartition if they are associated with a facility allocated to the CICS region and extrapartition if the data is directed to a destination that is external to the CICS region. Transient data queues must be defined and installed before the first reference by an application program.

You can:

- Write data to a transient data queue (`WRITEQ TD` command).
- Read data from a transient data queue (`READQ TD` command).
- Delete an intrapartition transient data queue (`DELETEQ TD` command).

**Note:** In this document we concentrate exclusively on intrapartition TD queues.

### Intrapartition Transient Data Queues

Intrapartition refers to data on direct-access storage devices for use with one or more programs running as separate tasks. Data directed to or from these internal queues is referred to as intrapartition data; it must consist of variable-length records.

When data is written to the queue by a user task, the queue can be used subsequently as input data by other tasks within the CICS region. All access is sequential, governed by read and write pointers. Once a record has been read, it cannot be read subsequently by another task. Intrapartition data may ultimately be transmitted upon request to the terminal or retrieved sequentially from the output dataset.

Typical uses of intrapartition data include:

- Message switching.
- Broadcasting.



- Database access.
- Routing of output to several terminals (for example, for order distribution).
- Queuing of data (for example, for assignment of order numbers or priority by arrival).
- Data collection (for example, for batched input from 2780 Data Transmission Terminals)

There are three types of intrapartition transient data queues:

#### **Non-recoverable**

Non-recoverable intrapartition transient data queues are recovered only on a warm start of CICS. If a unit of work (UOW) updates a non-recoverable intrapartition queue and subsequently backs out the updates, the updates made to the queue are not backed out.

#### **Physically recoverable**

Physically recoverable intrapartition transient data queues are recovered on warm and emergency restarts. If a UOW updates a physically recoverable intrapartition queue and subsequently backs out the updates, the updates made to the queue are not backed out.

#### **Logically recoverable**

Logically recoverable intrapartition transient data queues are recovered on warm and emergency restarts. If a UOW updates a logically recoverable intrapartition queue and subsequently backs out the changes it has made, the changes made to the queue are also backed out. On a warm or an emergency restart, the committed state of a logically recoverable intrapartition queue is recovered. In-flight UOWs are ignored.

## **Automatic Transaction Initiation (ATI)**

For intrapartition queues, CICS provides the option of automatic transaction initiation (ATI).

A basis for ATI is established by the system programmer by specifying a non-zero trigger level for a particular intrapartition destination. When the number of entries (created by WRITEQ TD commands issued by one or more programs) in the queue reaches the specified trigger level, a transaction specified in the definition of the queue is automatically initiated. Control is passed to a program that processes the data in the queue; the program must issue repetitive READQ TD commands to deplete the queue.

When the queue has been emptied, a new ATI cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether or not execution of the earlier task has ended. The exact point at which a new ATI cycle begins depends on whether or not the queue is defined as logically recoverable. If the queue is defined with a RECOVSTATUS of No or Physical, the new ATI cycle begins when the queue is read to QZERO. But if the queue is

defined with a recoverability attribute of Logical, the new ATI cycle begins only after the task terminates after having read the queue to QZERO.

If an automatically initiated task does not empty the queue, access to the queue is not inhibited. The task may be normally or abnormally ended before the queue is emptied (that is, before a QZERO condition occurs in response to a READQ TD command). If the contents of the queue are to be sent to a terminal, and the previous task completed normally, the fact that QZERO has not been reached means that trigger processing has not been reset and the same task is reinitiated. A subsequent WRITEQ TD command does not trigger a new task if trigger processing has not been reset.

## Presentation of the Mechanism on Target Platform

### Tuxedo /Q

Tuxedo /Q offers a robust and versatile queuing system with the same capabilities as TD queues and more.

Queues can be defined as recoverable or not, and triggering with a few different options is also available. The management of errors is much more sophisticated, and will simplify error management in case of ATI transaction failures on target.

### Architecture Design

**Table 4-5 Source to Target Mapping**

Source Element	Target Correspondence
TD Queue intrapartition	Tuxedo /Q Queue.
Associated transaction (TRANID)	Associated transaction offered by an ATR server.
Trigger level	Trigger level.
Recoverability: No Physical Logical	Similar levels available as on target, but with different configuration principles.

The CICS verbs READQ TD, WRITEQ TD and DELETEQ TD (applied to intrapartition queues), now read, write or delete from a Tuxedo /Q queue. (tpenqueue and tpdequeue) in terms of tuxedo vocabulary.

If the Queue is logically recoverable, these actions are done in the current UOW, else they are done atomically, independently of the current UOW.

For information, inside CICS Runtime, this is done by adding the TPNOTRAN flag to operations on non-logically recoverable queues.

## Triggering

In case of triggering, like in native CICS, a transaction will be automatically triggered, this transaction having to read the corresponding queue and process accordingly the messages.

In CICS Runtime these asynchronous transactions are offered and processed by a dedicated server type ARTATR, with either of its two variants ARTATR1 and ARTATR.N.

These servers process all asynchronous transactions, more precisely, transactions submitted by START TRANSID, or by automatic Transaction Invocation related to td queue intrapartition.

In this case a specific CICS Runtime client, TDI\_TRIGGER, is used to launch the corresponding asynchronous transaction, when the trigger level is reached.

# Runtime CICS Configuration of TD Queue Intrapartition

## CICS Runtime Resource Declaration

Every CICS-like resource in CICS Runtime, is declared using a dedicated configuration file stored in directory `$(KIXCONFIG)`.

TD Queue extrapartition and TD Queue intrapartition resource declaration share very few arguments, and are semantically very different objects, even if using the same API for read and write operations.

This is the reason why, in CICS Runtime, we have separated TD Queue extrapartition resource configuration and TD Queue intrapartition resource configuration into two different resource files.

Intrapartition queues are declared in the file `tdqintra.desc`, described in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

The important attributes are:

### **TDQUEUE(name)**

The queue name, exactly identical to the queue name in the source configuration, This name must be the same as the name of the queue in the Tuxedo /Q configuration.

### **RECOVSTATUS(status)**

Only the status NO or LOGICAL, are accepted, the difference between the two modes impacts the treatment of WRITEQ TD and READQ TD, more precisely LOGICAL making them part of the current UOW, while NO makes them atomic operations independent of the current UOW.

The difference between NO or PHYSICAL, is not defined in the resource configuration file but will be implemented using native tuxedo /Q configuration parameters, mapping to persistent /Q or non persistent.

### **TRANSID and TRIGGERLEVEL**

In the current release are documentary only in `tdqintra.desc`, it is their value in /Q configuration which is taken in account.

### **QSPACE\_NAME**

New argument needed for /Q: defining into which QSPACE the current /Q is stored. This argument is mandatory and must match the QSPACE\_NAME into which the actual /Q queue is physically stored.

## **/Q Configuration for TD Queue Intrapartition in CICS Runtime**

For detailed and accurate information on `qmadmin` and /Q configuration *Using the ATMI /Q Component* in the Tuxedo documentation.

The script `mk_td_qm_config.sh` distributed with CICS Runtime provides an example of qspace creation and then of queue creation and configuration into /Q, to be used for TD intrapartition queues.

This script uses three environment variables, which must be set according to your environment:

- `KIX_TD_QSPACE_DEVICE`: must contain the filename of the physical file containing the /Q database for TD queues.
- `KIX_TD_QSPACE_NAME`: contains the name of the logical QSPACE to create, which will contains the queues.
- `KIX_TD_QSPACE_IPCKEY`: a specific key which must be unique on the machine for the IPC used by the instance of /Q.

The creation of the device (`KIX_TD_QSPACE_DEVICE`) and of the QSPACE are very standard, we will not detail them.

The interesting part is related to queue configuration.

A `qopen QspaceName` command, to open the qspace which will contain the queues must be made before the creation of any queue. The `QspaceName` must match the `QSPACE` in the resource declaration of these queue(s).

Below is an example of an interactive queue creation using `qmadmin`, where the questions asked by `qmadmin` are in normal font, while the entries typed in by the user are in bold.

#### Listing 4-24 qopen Dialog

---

```
qopen TD_QSPACE
qcreate
Queue name: TEST
Queue order (priority, time, expiration, fifo, lifo): fifo
Out-of-ordering enqueueing (top, msgid, [default=none]): none
Retries [default=0]: 5
Retry delay in seconds [default=0]: 0
High limit for queue capacity warning (b for bytes used, B for blocks used,
% for percent used, m for messages [default=100%]): 5m
Reset (low) limit for queue capacity warning [default=0%]: 0m
Queue capacity command: "TDI_TRIGGER -t S049"
```

---

In a script an exact equivalent to this manual entry would be:

#### Listing 4-25 qopen Script

---

```
qopen TD_QSPACE
qcreate TEST fifo none 3 0 5m 0m "TDI_TRIGGER -t S049"
```

---

## qopen Parameters

### TD\_QSPACE

The `QspaceName` must match the `QSPACENAME` in the resource declaration of these queue(s).

### Queue name

The name of the queue must match exactly the name provided in the resource declaration.

### Queue order

The default dequeuing order when reading the queue, the setting corresponding to TD intra native behavior is: `fifo`.

### Out-of-ordering enqueueing

Not meaningful unless some application is using native /Q interface to write into these queue; for Runtime CICS only usage to set it to is: `none`

### Retries

Defines the number of times a message will be put back on the queue in case of abort of the UOW having read this queue, to avoid resubmitting again and again an ATI transaction which fails because of a bad message, set this number to a reasonable number.

When this number is reached, or at the first abort if you set it to zero, the message will be removed from this queue and put onto the error queue for further analysis.

### Retry delay in seconds

If retries is not null, defines a delay before putting a record back on its queue, in case of rollback, the recommended value with Runtime CICS is the default value 0.

### High limit for queue capacity warning

This is the much more flexible equivalent of the trigger level of TD queues. For a setting compatible with TD queues, set it to the trigger level and express it in number of messages. For example: 0m to suspend triggering, or 5m for a trigger level of 5 messages in the queue.

### Reset (low) limit for queue capacity warning

This is the down level to be reached before resetting the trigger for the upper limit, for compatibility with TD queue behavior, it should be set to 0, (`QZERO`) which is the reset value for TD queues in CICS.

**Queue capacity command:**

This is the command to be launched when the trigger level is reached, in CICS Runtime it should be set to: `TDI_TRIGGER -t TRID`. Where `TRID` is the Transaction identifier of the transaction to trigger which should match the `TRANSID` of the resource configuration.

---

**Tip:** ATR servers when processing an ATI, know whether the transaction reached `QZERO` or not, and whether it was a success or a rollback. So if `QZERO` is not reached, they resubmit the transaction in the same manner as on the source platform. But now it is the number of retries which will replace the `ATIFACILITY` parameter and will govern the fact that a rolledback TD queue record will be resubmitted or not. It is a progress compared with the source is that now the administrator can decide the number of resubmissions, and get the faulty messages on an error queue.

---

## Activating the ARTTDQ in the Tuxedo ubbconfig File

To enable TDQ motoring, ARTTDQ server should be activated.

### Listing 4-26 Activating the ARTTDQ in the ubbconfig File

---

```
*SERVERS
...
ARTTDQ SRVGRP=GRP02
SRVID=40
```

```
MIN=1 MAX=1  
CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_tdq -e  
/home2/work9/demo/Logs/TUX/sysout/stderr_tdq -r -- -s KIXR -l SIMAPP"  
...
```

---

Where:

**\*SERVERS**

Tuxedo ubbconfig Keyword indicating the definition of Server Section.

**SRVGRP**

The Tuxedo Group Name to which ARTTDQ belongs.

**SRVID**

The identifier of a Tuxedo Server of ARTTDQ.

**MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

**CLOPT**

A quoted text string passed to the server containing its parameters:

- o  
Indicates the following file is used for the standard output messages of the server.
- e  
Indicates the following file is used for the error output messages of the servers.
- r  
Is a Tuxedo parameter used to have statistical reports.
- s KIXR  
Indicates the CICS Runtime name where the transaction runs is KIXR.
- l SIMAPP  
Indicates that only the components of the SIMAPP group are to be selected at start up.



# Implementing CICS Application Using Temporary Storage (TS) Queue POOL

These transactions use CICS programs containing EXEC CICS requests relative to CICS Temporary Storage Queues.

The statements used are EXEC CICS WRITEQ TS ... END-EXEC, EXEC CICS READQ TS ... END-EXEC, EXEC CICS DELETEQ TS ... END-EXEC.

If at least one of your programs contains one of these statements and the queue is defined with POOLNAME (tsqmodel.desc), install and activate the new features of CICS Runtime without changing your other settings.

To manage TS Queues with POOL, do the following steps.

1. First, define database table to contain the TS Queue and POOL. CICS Runtime currently supports Oracle database and UDB.

CICS Runtime provides a UNIX script, `crtstable_{Oracle|UDB}`, to create all these tables. Set `MT_DB_LOGIN` environment variable and then execute `crtstable_{Oracle|UDB}` to create these tables. Set `MT_DB_LOGIN` to enter database connection information. For example: `DBUSER/DBPASSWD@DBSID`. See for [Listing 4-27](#) an example for Oracle database users.

2. Second, modify the Tuxedo `UBBCONFIG` file to modify the server group used by `ARTTSQP` to establish the connection to Oracle in the `UBBCONFIG *GROUPS` section.
3. Next, activate the `ARTTSQP` CICS Runtime Tuxedo server. To activate this server, add it to the `UBBCONFIG *SERVERS` section. See [Listing 4-28](#) for an example.
4. Last, use the Tuxedo `tadmin psr` and `psc` commands to check that the server is running and that new services are published. See [Listing 4-29](#) for an example.

When using `ARTTSQP_UDB`, you may need to do the followings to rebind the server for new DB2 server/tables.

1. Set environment variable `MT_DB_LOGIN` to enter database connection information.
2. Go to `$KIXDIR/bin`.
3. Execute:

```
../tools/bind.sh tspool_UDB.bnd
```

**Listing 4-27 Example of the UBBCONFIG Configuration Concerning the Server Group GRP02 Used by ARTTSQP Server**

---

```
*GROUPS
...
GRP02
GRPNO=12
ENVFILE="/home2/work9/demo/config/tux/envfile"
TMSNAME="TMS_ORA"
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/work9/work9+SesTm=600+LogDir=/home2/work9/demo/Logs/TUX/xa+DbgFl=0x20"
...
```

---

Where:

**\*GROUPS**

Is the Tuxedo UBBCONFIG keyword indicating definitions of server groups.

**GRPNO**

Is the Tuxedo group number.

**TMSNAME**

Is the name of the Tuxedo Transaction Manager Server executable.

**OPENINFO**

Is the parameters sent to the Oracle\_XA Manager.

**Listing 4-28 Activating the ARTTSQP in the UBBCONFIG File**

---

```
*SERVERS
...
ARTTSQP      SRVGRP=GRP02
              SRVID=40
```

```
MIN=2 MAX=2

CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_tsqp -e
/home2/work9/demo/Logs/TUX/sysout/stderr_tsqp -r -- -L list1"

...
```

---

Where:

**\*SERVERS**

Is the Tuxedo UBBCONFIG keyword indicating a server section definition.

**SRVGRP**

Is the Tuxedo group name to which ARTTSQP belongs.

**SRVID**

Is the identifier of a Tuxedo server of ARTTSQP.

**MIN=2 and MAX=2**

Indicates that you run two instances of this server (you can run greater than or equal to one instance of this server).

**CLOPT**

Is a quoted text string passed to the server containing its parameters:

- o Indicates the following file is used for the standard output messages of the server.
- e Indicates the following file is used for the error output messages of the server.
- r Is a Tuxedo parameter used to have statistical reports.
- L Indicates the list of groups to be loaded by this server.

**Listing 4-29 Checking ARTTSQP Server and Services**

---

```
# tadmin
```

## Implementing CICS Applications

...

> psr

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
BBL	42444	KIXR	0	30	1500	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30001	0	0	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30002	0	0	( IDLE )	
ARTADM	00011.00010	GRP01	10	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP02	20	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP02	21	0	0	( IDLE )	
ARTTSQP	00012.00040	GRP02	40	0	0	( IDLE )	
ARTTSQ	00012.00045	GRP02	45	0	0	( IDLE )	

> psc -I 40

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
arttsqp_mib+	tsqp_mib_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSPOOL_ADM	tsqp_adm_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00004_ADM	tsqp_adm_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00004_TS+	tsqp_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00003_ADM	tsqp_adm_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00003_TS+	tsqp_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00002_ADM	tsqp_adm_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00002_TS+	tsqp_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL
TSM00001_ADM	tsqp_adm_svc	ARTTSQP	GRP02	40	KIXR	0	AVAIL

```
TSM00001_TS+ tsqp_svc      ARTTSQP      GRP02      40      KIXR      0 AVAIL
> quit
#
```

---

## Implementing Distributed Program Link (DPL)

For several reasons, on z/OS, the Distributed Program Link function enables a local CICS program (the client program) to call another CICS program (the server program) in a remote CICS region via EXEC CICS LINK statements. CICS Runtime supports this feature used in multi-CICS architecture like MRO among migrated regions.

### To Detect That DPL Is Needed

Unless you wish to use the DPL in a UNIX written application, check the technical specificities of the z/OS application

1. Check on z/OS, using the CEDA system transaction, if at least one remote program is defined in the z/OS CICS CSD file. Such programs have some of their fields of the REMOTE ATTRIBUTES section filed:

#### Listing 4-30 Checking for Remote Programs

---

```
DEF PROGR
OVERTYPE TO MODIFY                                CICS RELEASE = 0610
CEDA DEFine PROGRam(                               )
  PROGRam      ==>
  Group        ==>
  DDescription ==>
....
REMOTE ATTRIBUTES
Dynamic      ==> No                No ! Yes
REMOTESystem ==> XXXX
```

```
REMOTEName ==> YYYYYYYY
Transid ==> ZZZZ
EXECUTIONSET ==> Dplsubset Fullapi ! Dplsubset
```

---

Where (CICS default values are underlined):

### **DYNAMIC(YES|NO)**

The following parameters cannot be overridden in the CICS LINK API. This field is only relevant for DPL use when it is set to NO and the three following fields are empty.

### **REMOTESYSTEM(name)**

Remote CICS region name. An empty field is not relevant with *DYNAMIC (YES)*

### **REMOTENAME(name)**

Remote server program name. An empty field is not relevant with *DYNAMIC (YES)* because the default is the client program name (*PROGram ==>*).

### **TRANSID(name)**

Remote mirror transaction. An empty field is not relevant with *DYNAMIC (YES)* because the default is the mirror system transaction CSMI.

### **EXECUTIONSET(FULLAPI|DPLSUBSET)**

The DPL cannot use the full CICS API but only a subset. The *DPLSUBSET* parameter indicates explicit usage of a DPL subset of the CICS API, but note that this subset may also be sufficient to execute LINK in a non-DPL context without errors. On the other hand, this field may contain *FULLAPI* in a DPL context but does not ensure that no "Invalid Request errors" will follow if non-DPL API are used.

As described above, in some cases, the Remote Attributes declaration may not exist or can be incomplete. The reason is that these fields establish only some of the default values, some of the previous parameters in bold in the example are not provided in the *EXEC CICS LINK API*.

## 2. Then check in the programs, inside the *EXEC CICS LINK API*:

- If the names of the programs called in this order match the names of programs defined in the CSD with remote attributes partially or fully informed.
- If these statement contain at least one of the optional remote parameters shown in italics in the following CICS LINK API (the others fields are not relevant for DPL).

**Listing 4-31 CICS LINK API For DPL**

---

```

EXEC CICS LINK PROGRAM(...)

    COMMAREA (...)

    LENGTH (...)

    DATALENGTH (...)

    RETCODE (...)

    SYSID(XXXX) : Remote CICS region name

    SYNCONRETURN : Used for remote CICS syncpoint or rollback

    TRANSID(XXXX) : Remote mirror transaction instead of the CSMI default

    INPUTMSG (...)

    INPUTMSGLEN (...)

END-EXEC

```

---

## Modifying the Tuxedo ubbconfig File to Manage the DPL

If at least one of your programs use the DPL, install and activate the ARTDPL server without changing your other settings.

To activate this server, modify your ubbconfig file to add this server to the \*SERVERS section of the Tuxedo ubbconfig file. This server belongs to the same Server Group as the Transactions Servers (ARTSTRN, ARTSTR1, ARTATRN, ARTATR1).

**Listing 4-32 ubbconfig File Example of a \*SERVERS Section Describing the ARTDPL Server.**

---

```

*SERVERS

...

ARTDPL      SRVGRP=GRP02

            SRVID=500

            CONV=N

            MIN=1 MAX=1 RQADDR=QKIXDPL REPLYQ=Y

```

```
CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_dpl -e  
/home2/work9/demo/Logs/TUX/sysout/stderr_dpl -r -- -s KIXD -l SIMPAPP"  
...
```

---

Where:

**\*SERVERS**

Tuxedo ubbconfig Keyword indicating a Server Section definition.

**SRVGRP**

Is the Tuxedo Group Name to which ARTDPL belongs.

**SRVID**

Is the identifier of a Tuxedo Server of ARTDPL.

**CONV=N**

Indicates that this server operates in a non-conversational mode.

**MIN=1 and MAX=1**

Indicates that only one instance of this server must be run.

**REPLYQ=Y**

Indicates that this server will respond.

**RQADDR=QKIXDPL**

Name of the Tuxedo queue used for the responses.

**CLOPT**

Is a quoted text string passed to the server containing its parameters:

-o

Indicates the following file is used for the standard output messages of the server.

-e

Indicates the following file is used for the error output messages of the server.

-r

Is a Tuxedo parameter used to provide statistical reports.

-s KIXD

Indicates the CICS Runtime name where the KIXD transaction is run.



**-I SIMAPP**

Indicates that only the components of the SIMPDPL group are to be selected at start up.

Use the Tuxedo `tmadmin psr` and `psc` commands to check that this server is running and that no new service is published:

**Listing 4-33 tmadmin Commands to Check ARTDPL Server**

---

```
# tmadmin
...

> psr
```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
-----	-----	-----	--	-----	-----	-----	-----
ARTDPL	QKIXDPL	GRP02	500	0	0	( IDLE )	
ARTATR1	00012.00300	GRP02	300	0	0	( IDLE )	
ARTSTR1	00012.00200	GRP02	200	0	0	( IDLE )	
BBL	200933	KIXR	0	5	250	( IDLE )	
TMS_QM	GQUEUE_TMS	GQUEUE	30001	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30001	0	0	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
TMS_QM	GQUEUE_TMS	GQUEUE	30002	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30002	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30003	0	0	( IDLE )	
TMQUEUE	01000.01010	GQUEUE	1010	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )	
TMQFORWARD	01000.01020	GQUEUE	1020	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP02	20	0	0	( IDLE )	
ARTATRn	QKIXATR	GRP02	30	0	0	( IDLE )	

## Implementing CICS Applications

```
ARTTSQ          00012.00040 GRP02          40          0          0 ( IDLE )
```

```
> psc
```

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
TMS	TMS	TMS_QM	GQUEUE	30001	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30001	KIXR	0	AVAIL
TMS	TMS	TMS_QM	GQUEUE	30002	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30002	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30003	KIXR	0	AVAIL
ASYNQ_QSPACE	TMQUEUE	TMQUEUE	GQUEUE	1010	KIXR	0	AVAIL
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA02	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
ASYNQ_QUEUE	ASYNQ_QUEUE	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA03	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA02	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA01	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA00	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
TSQUEUE	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL

```
> quit
#
```

---

## Declaring Remote Programs in CICS Runtime

To allow an application to use distributed programs called in EXEC CICS LINK statements, these programs must be declared to CICS Runtime.

1. To declare REMOTE programs which can only use the DPL Subset of the CICS API:
  - In the programs.desc file, set REMOTESYSTEM (the 7th field of the csv format dataset), to remote SYSID name (KIXD in sample of [Listing 4-32](#)).

The default is local (empty field), meaning that local programs are declared because they can use the FULL CICS API.

In our Simple Application example, if we suppose that RSSAT000, RSSAT001 are remote and RSSAT002 and RSSAT003 are local, then the programs.desc file is set to:

### Listing 4-34 Simple Application programs.desc Configuration of Remote Programs

---

```
#PROGRAM;GROUP;DESCRIPTION;LANGUAGE;EXECKEY;STATUS;REMOTESYSTEM;REMOTENAME
RSSAT000;SIMPAPP;Home Menu Program of Simple Application;COBOL;
;ENABLE;KIXD

RSSAT001;SIMPAPP;Customer Detailed Inf Program of Simple Application;COBOL;
;ENABLE;KIXD

RSSAT002;SIMPAPP;Customer Maintenance Program of the Simple
Application;COBOL; ;ENABLE

RSSAT003;SIMPAPP;Customer List of the Simple Application;COBOL; ;ENABLE
```

---

2. Shutdown and reboot Tuxedo.
3. Using the Tuxedo tadmin psr and psc commands, check that new services for DPL programs are published and managed by ARTDPL: KIXD\_RSSAT0001 and KIXD\_RSSAT0003.

**Note:** To avoid problems with homonyms, these distributed services have their names composed of the Tuxedo DOMAINID defined in the ubbconfig and the name of the program they manage.

**Listing 4-35 Using tadmin Commands to Check DPL Services**

---

```
{deimos:work9}~/home2/work9/demo/Logs/TUX/sysout# tadmin
...

> psr
```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
-----	-----	-----	--	-----	-----	-----	-----
ARTDPL	QKIXDPL	GRP02	500	0	0	( IDLE )	
ARTATR1	00012.00300	GRP02	300	0	0	( IDLE )	
ARTSTR1	00012.00200	GRP02	200	0	0	( IDLE )	
BBL	200933	KIXR	0	5	250	( IDLE )	
TMS_QM	GQUEUE_TMS	GQUEUE	30001	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30001	0	0	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
TMS_QM	GQUEUE_TMS	GQUEUE	30002	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30002	0	0	( IDLE )	
TMS_ORA	GRP02_TMS	GRP02	30003	0	0	( IDLE )	
TMQUEUE	01000.01010	GQUEUE	1010	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )	
TMQFORWARD	01000.01020	GQUEUE	1020	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP02	20	0	0	( IDLE )	
ARTATR1	QKIXATR	GRP02	30	0	0	( IDLE )	
ARTTSQ	00012.00040	GRP02	40	0	0	( IDLE )	

## Implementing Distributed Program Link (DPL)

> psc

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
-----	-----	-----	-----	--	-----	-----	-----
KIXD_RSSAT0+	dplsvc	ARTDPL	GRP02	500	KIXR	0	AVAIL
KIXD_RSSAT0+	dplsvc	ARTDPL	GRP02	500	KIXR	0	AVAIL
TMS	TMS	TMS_QM	GQUEUE	30001	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30001	KIXR	0	AVAIL
TMS	TMS	TMS_QM	GQUEUE	30002	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30002	KIXR	0	AVAIL
TMS	TMS	TMS_ORA	GRP02	30003	KIXR	0	AVAIL
ASYNQ_QSPACE	TMQUEUE	TMQUEUE	GQUEUE	1010	KIXR	0	AVAIL
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0	AVAIL
SA03	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA01	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
SA00	kixsvc	ARTSTRN	GRP02	20	KIXR	0	AVAIL
ASYNQ_QUEUE	ASYNQ_QUEUE	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA03	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA01	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
ASYNQ_SA00	atrsvc	ARTATR	GRP02	30	KIXR	0	AVAIL
TSM00004_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00003_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL
TSM00002_TSQ	tsqsvc	ARTTSQ	GRP02	40	KIXR	0	AVAIL

## Implementing CICS Applications

```
TSM00001_TSQ tsqsvc      ARTTSQ   GRP02    40      KIXR      0 AVAIL
TSM00000_TSQ tsqsvc      ARTTSQ   GRP02    40      KIXR      0 AVAIL
TSQUEUE      tsqsvc      ARTTSQ   GRP02    40      KIXR      0 AVAIL

> quit
# .
```

---

To see full details on the truncated values displayed, you can use the Tuxedo verbose command.

To reduce the scope of the services listed to only those managed by ARTDPL (SRVID=500), use the Tuxedo `pvc` command followed with the `-i srvid` parameter to restrict the display to a particular server id.

In our example, the `srvid` of the ARTDPL server is 500 as displayed just above.

### Listing 4-36 Using `tadmin` Commands to Check Specific DPL Service in Verbose Mode

---

```
# tadmin
...

> verbose
Verbose now on.

> pvc -i 500
    Service Name: KIXD_RSSAT003
    Service Type: USER
    Routine Name: dplsvc
        Prog Name: /home2/work9/KIXEDO/bin/ARTDPL
    Queue Name: QKIXDPL
    Process ID: 1327244, Machine ID: KIXR
```

```
      Group ID: GRP02, Server ID: 500
      Current Load: 50
      Current Priority: 50
      Current Trantime: 30
      Current Blocktime: 0
      Current BUFTYPECONV: 0
      Requests Done: 0
      Current status: AVAILABLE

      Service Name: KIXD_RSSAT001
      Service Type: USER
      Routine Name: dplsvc
      Prog Name: /home2/work9/KIXEDO/bin/ARTDPL
      Queue Name: QKIXDPL
      Process ID: 1327244, Machine ID: KIXR
      Group ID: GRP02, Server ID: 500
      Current Load: 50
      Current Priority: 50
      Current Trantime: 30
      Current Blocktime: 0
      Current BUFTYPECONV: 0
      Requests Done: 0
      Current status: AVAILABLE

> quit
#
```

---

## Implementing CICS Common Work Area (CWA)

On z/OS, the CWA is a common storage area defined in memory for a CICS region that programs can use to save and exchange data between themselves as long as this CICS region is running.

This area is addressed thru a pointer delivered by the CICS statement `EXEC CICS ADDRESS CWA`. If you find this CICS statement in your application, you have to implement this feature in CICS Runtime.

### Listing 4-37 COBOL Example of CWA Usage

---

```
LINKAGE SECTION.  
01  COMMON-WORK-AREA.  
    03  APPL-1-ID          PIC X(4) .  
    03  APPL-1-PTR        USAGE IS POINTER.  
    03  APPL-2-ID          PIC X(4) .  
    03  APPL-2-PTR        USAGE IS POINTER.  
  
PROCEDURE DIVISION.  
  
. . .  
  
    END-EXEC.  
  
* Set up addressability to the CWA  
    EXEC CICS ADDRESS  
        CWA (ADDRESS OF COMMON-WORK-AREA)  
  
    END-EXEC.
```

---

After the `CICS ADDRESS CWA`, the address of the COBOL group named `COMMON-WORK-AREA` is set to the address of the CWA allocated by CICS, meaning that `COMMON-WORK-AREA` maps and refines this memory area. The total amount of this shared memory is fixed and defined at CICS start up.



## To Replicate CICS ADDRESS CWA Functionality in CICS Runtime

1. Contact your z/OS CICS Administrator to know the size of memory implemented. (For your information this value is defined with the parameter `WRKAREA` of the `DFHSIT`. The default value is 512 bytes and the size can vary from 0 to 3584 bytes). Another way is to calculate the biggest size of the data record contained in the programs addressing the CWA.
2. Modify your `~/profile` UNIX system file to export a new CICS Runtime variable, `KIX_CWA_SIZE`, and set it to the value found in the `WRKAREA` of the `DFHSIT`. If this variable is not declared, note that the default value is 0 and the authorized interval from 0 to 32760 bytes.

Example:

```
KIX_CWA_SIZE=512
```

3. Modify your `~/profile` UNIX system file to export a new CICS Runtime variable, `KIX_CWA_IPCKEY`, and valorize it to a Unix IPC key to define the cross memory segment used as CWA.

Example:

```
KIX_CWA_IPCKEY=200944
```

4. Restart Tuxedo to take all these changes into account.

## Implementing a CICS Transaction Work Area (TWA)

On z/OS, the TWA is a common storage area defined in memory for a CICS region that programs can use to save and exchange data between themselves during the execution time of one CICS transaction. In other words, this TWA can only be accessed by the programs participating in the transaction. This area is addressed thru a pointer delivered by the CICS statement `EXEC CICS ADDRESS TWA`. If you find an `EXEC CICS ADDRESS TWA` statement in your application, you have to implement this feature in CICS Runtime.

### Listing 4-38 A COBOL Example of Use of the TWA

---

```
LINKAGE SECTION.
01  TRANSACTION-WORK-AREA.
    03  APPL-1-ID           PIC X(4) .
    03  APPL-1-PTR         USAGE IS POINTER.
    03  APPL-2-ID           PIC X(4) .
```

## Implementing CICS Applications

```
03  APPL-2-PTR          USAGE IS POINTER.

PROCEDURE DIVISION.

. . .

END-EXEC.

* Set up addressability to the TWA

EXEC CICS ADDRESS

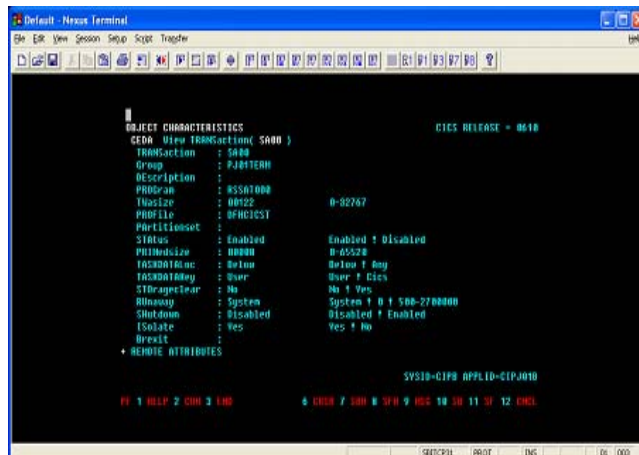
          TWA (ADDRESS OF TRANSACTION-WORK-AREA)

END-EXEC.
```

---

After the CICS ADDRESS TWA, the address of the COBOL group named TRANSACTION-WORK-AREA is set to the address of the TWA allocated by CICS, meaning that TRANSACTION -WORK-AREA maps and refines this memory area. The total amount of this shared memory is defined for each transaction in the z/OS CSD configuration file in the field TWAsize. The next screen shows the result of a z/OS CEDA system transaction where the TWAsize parameter is set to 122 for the SA00 transaction code:

**Figure 4-3 z/OS ceda System Transaction Example**



To replicate this functionality in CICS Runtime:

1. Modify the CICS Runtime `transactions.desc` file to report the needed amount of TWA memory (`TWAsize>0`).
2. For each transaction using programs with `CICS ADDRESS TWA` statements, modify the `transactions.desc` file to declare its `TWAsize` in the sixteenth field of this csv format file.

**Table 4-6 TWA Size Values Associated to Each Transaction Code of the Simple Application**

Transaction	TWA Size
SA00	0
SA01	100
SA02	200
SA03	300

**Listing 4-39 Configuration of TWA in the transactions.desc File**

```
#Transaction;Group;Description;Program; ; ; ; ; ;Status; ; ; ;Tranclass
;TWA Size
SA00;SIMPAPP;pg for simpapp;RSSAT000; ; ; ; ; ;ENABLED
SA01;SIMPAPP;pg for simpapp;RSSAT001; ; ; ; ; ;ENABLED; ; ; ;100
SA02;SIMPAPP;pg for simpapp;RSSAT002; ; ; ; ; ;ENABLED; ; ; ;200
SA03;SIMPAPP;pg for simpapp;RSSAT003; ; ; ; ; ;ENABLED; ; ; ;300
```

**Note:** Nothing is indicated for the SA00 transaction that had a TWA size equal to zero.

3. Restart the CICS Runtime Tuxedo servers, the modifications can be seen in the different `stderr` files of the servers involved in the transaction management (`ARTSTRN`, `ARTSTR1`, `ARTATRn` and `ARTATR1`)

**Listing 4-40 stderr\_strn TWA Example**

```
|-----|
| TRANSACTIONS loaded : < 4> |
```

```

|-----|-----|-----|-----|-----| | | |
|---|---|---|---|---|---|---|---|
|      |      |      |      |      |
|T|      |      |      |      |
|TRAN|  GROUP  |      |      |      |      |      |      |
|TASK |R|  TRAN  |  TWA |MAX|      |      |      |
|      |      |      |      |      |      |      |
|A|  CLASS  |  SIZ |ACT|      |      |      |      |
|      |      |      |      |      |      |      |      |
|C|      |      |  IVE|      |      |      |      |
|-----|-----|-----|-----|-----| | | | |
|---|---|---|---|---|---|---|---|---|
|SA00|SIMPAPP  |RSSAT000      |      |      |      |      |
|USER |Y|      |      |00000|999|      |      |      |
|SA01|SIMPAPP  |RSSAT001      |      |      |      |      |
|USER |Y|      |      |00100|999|      |      |      |
|SA02|SIMPAPP  |RSSAT002      |      |      |      |      |
|USER |Y|      |      |00200|999|      |      |      |
|SA03|SIMPAPP  |RSSAT003      |      |      |      |      |
|USER |Y|      |      |00300|999|      |      |      |

```

## Supporting TWA in ARTDPL

The programs within a transaction run by ARTDPL now can access TWA with following steps.

1. Modify the CICS Runtime `transactions.desc` file to configure TWASize needed for ARTDPL transaction.

### Listing 4-41 Configuration of TWA for ARTDPL in the transactions.desc File

```

#Transaction;Group;Description;Program; ; ; ; ; ;Status; ; ; ;Tranclass
;TWA Size

```

```
CPMI;SIMPAPP;pg for simpapp;DFHMIRS; ; ; ; ; ;ENABLED; ; ; ; ;100
```

---

DFHMIRS is the internal mirror program in CICS that handles inbound function shipping. In CICS RT, this mirror program should be defined under the transaction used to run the linked program in the transaction resource file if TWA is used. In the list, ARTDPL runs remote linked program under transaction named CPMI and it has TWA size equal to 100.

**Note:** Users should not name application program as DFHMIRS.

- Restart the CICS Runtime Tuxedo servers and the modifications can be seen in ARTDPL stdout file.

**Listing 4-42**      **stdout\_dpl TWA Example**

---

```
|-----|
| TRANSACTIONS loaded : < 1> |
|-----|-----|-----|-----|-----| |
|---|---|---|---|---|---|
|          |          |          |          |          |
|T|          |          |          |          |          |
|TRAN|  GROUP  |          | PROGRAM  |ALIA|M|O|PRI|E|E|  STATUS
|TASK|R|  TRAN  | TWA  |MAX| | | |
|          |          |          |          |          |
|A|  CLASS  | SIZ |ACT|          |          |          |          |
|          |          |          |          |          |          |          |
|C|          |          |IVE|          |          |          |          |
|-----|-----|-----|-----|-----|
-|-----|-|-----|-|-----|-|-----|
|CPMI|SIMPAPP  |DFHMIRS          |          |N|N|001|N|N|ENABLED
|USER |Y|          |00100|999|
|-----|
|-----|
```

---

## Implementing Integration with WebSphere MQ

- [Using ART CICS Transaction Trigger Monitor \(ARTCKTI\)](#)
- [Rebuilding ART for CICS Servers](#)
- [Handling CICS Runtime Preprocessor of MQOPEN/MQCLOSE Calls](#)
- [Encoding Character Set](#)
- [Changing COMP-5 back to BINARY Data Type](#)

### Using ART CICS Transaction Trigger Monitor (ARTCKTI)

The ART CICS Transaction Trigger Monitor (ARTCKTI) behaves the same as the CICS CKTI transaction. It listens on one or multiple WebSphere MQ initiation queues, retrieves trigger messages when a trigger event occurs, and then forwards the trigger messages to the target transaction.

#### Work Flow

ARTCKTI is a standalone Oracle Tuxedo server. The ARTCKTI server behaves as follows:

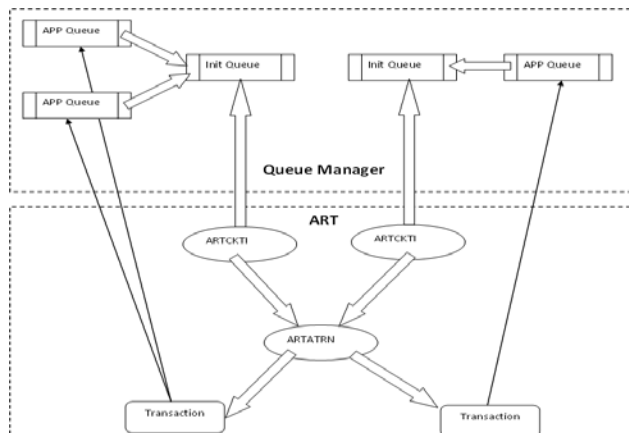
1. Monitor one or multiple WebSphere MQ initiation queues.  
One server instance can only monitor WebSphere MQ initiation queues within the same WebSphere MQ queue manager. The queues in different WebSphere MQ queue managers should be monitored by separate ARTCKTI server instances.
2. When trigger message has arrived, the ARTCKTI server retrieves the message.
3. Retrieve the transaction ID from the trigger message.
4. Transfer the trigger message from structure MQTMC to MQTMC2.  
Since MQTMC has many fields, it is always too complicated to send the structure as the parameter of EXEC CICS START call. MQTMC2 is used in CKTI to pass the structure as data to the START request for the trigger monitor.
5. Invoke the target transaction, and send the MQTMC2 data.  
Since CICS CKTI transaction starts the target transaction with asynchronous call (EXEC CICS START), the ARTCKTI server also starts the target transaction with asynchronous call (Tuxedo tpacall).

6. User transaction retrieves the trigger message by CICS `RETRIEVE`, and performs operations on the WebSphere MQ application queue.

If the user transaction does not retrieve the message or the triggered transaction is not available, WebSphere MQ no longer sends trigger message in this condition. A new trigger message is issued until the WebSphere MQ initiation queue is reopened or a new trigger condition is met.

Figure 4-4 illustrates the behavior.

**Figure 4-4 WebSphere MQ Trigger Condition**



**Note:** By default, ARTCKTI is built with client mode and a specific Websphere MQ version. You need to rebuild ARTCKTI server if your ARTCKTI accesses Websphere MQ with server mode or if your Websphere MQ runtime version is lower than the version upon which the default ARTCKTI is built. For more information, see [Rebuild ARTCKTI Server](#).

## Command Configuration

ARTCKTI accepts the following parameters for the ubbconfig file.

- **-i trigger\_interval:** specifies the maximum time (in milliseconds) that the ARTCKTI server waits for a message to arrive at the WebSphere MQ initiation queue.
- **-s retry\_interval:** specifies the retry interval for ARTCKTI to reconnect to WebSphere MQ queue manager or reopen WebSphere MQ initiation queue upon failure.
- **-m queue\_manager\_name:** specifies the name of the WebSphere MQ queue manager to be monitored.

- `-q queue1,queue2,.....`: specifies the name of the WebSphere MQ initiation queue to be monitored.

### Configuring WebSphere MQ Servers to Trigger ART for CICS Transactions

WebSphere MQ can trigger CICS transactions when one or more messages are placed on the queue. ART for CICS provides ARTCKTI server as trigger monitor (equivalent to CICS CKTI transaction).

To enable MQ Manager to trigger an ART for CICS transaction, the triggering queue and process need to be defined appropriately in the MQ Manager configuration. [Listing 4-43](#) shows a sample configuration.

#### Listing 4-43 Sample Configuration to Trigger ART for CICS Transactions

---

```
runmqsc MYMQM << EOF

DEFINE QLOCAL(INIT1) REPLACE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE)
DESCR('INITIATION QUEUE')

DEFINE QLOCAL(APP1) REPLACE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE)
DESCR('APPLICATION QUEUE') INITQ('INIT1') PROCESS('APP1.P') TRIGGER
TRIGTYPE(FIRST)

DEFINE PROCESS(APP1.P) DESCR('PROCESS DEFINITION') APPLTYPE(UNIX)
APPLICID('SA01')

DEFINE QLOCAL(APP2)

DEFINE CHANNEL(APP.C) CHLTYPE(SVRCONN)
DEFINE LISTENER(APP.L) TRPTYPE(TCP)

ALTER QMGR TRIGINT(0)

DEFINE QMODEL('SYSTEM.SAMPLE.REPLY') REPLACE DESCR('GENERAL REPLY QUEUE')
EOF
```



In the example above, `INIT1` is defined as a trigger queue associated with process `APP1.P`, and specifies that the `FIRST` message placed on the queue will be used for triggering. The process definition that follows defines `APPLTYPE` as `UNIX` and specifies `ART` for CICS transaction ID to be triggered as `APPLICID`.

Based on this definition, when the first message is queued on `INIT1` queue, `ARTCKTI` trigger monitor will `START TRANID('SA01')` in `ARTATRN` server. The application transaction will usually drain the queue and process all available messages. The next time a new message is queued on `INIT1`, it will be triggered again.

## Rebuilding ART for CICS Servers

Before rebuilding servers, you need to

- [Prepare WebSphere MQ RM Definitions](#)

You need to do the followings if ART for CICS transaction servers use different modes to access WebSphere MQ.

- [Rebuild TMS\\_MQM Server](#)
- [Rebuild ART for CICS Transaction Servers](#)
- [Rebuild ARTCKTI Server](#)

After rebuilding the above servers, you need to

- [Update Oracle Tuxedo UBBCONFIG and OPENINFO](#)

## Prepare WebSphere MQ RM Definitions

Prepare WebSphere MQ RM definitions by adding the following in `$TUXDIR/udataobj/RM` file if using local WebSphere MQ server:

```
# For building TMS_MQM server to work with local MQ server
MQSeries_XA_RMI:MQRMIXASwitchDynamic: /opt/mqm/lib64/libmqmxa64.so
/opt/mqm/lib64/libmqm.so

# For building ARTSTR*/ARTATR*/ARTDPL server
MQSeries_XA_RMI_COB:MQRMIXASwitch: -L${MQMDIR}/lib64 -lmqmx64 -lmqmc64
```

If using local WebSphere MQ client for remote connection to WebSphere MQ server, use this version (do not use duplicate entries in RM file):

```
# For building TMS_MQM server to work with local MQ client
MQSeries_XA_RMI:MQRMIXASwitchDynamic: /opt/mqm/lib64/libmqcxa64.so
/opt/mqm/lib64/libmqic.so

# For building ARTSTR*/ARTATR*/ARTDPL server
MQSeries_XA_RMI_COB:MQRMIXASwitch: -L${MQMDIR}/lib64 -lmqcx64 -lmqicb
```

**Note:** `${MQMDIR}` is the environment variable which indicates the installation path of MQM.

### Rebuild TMS\_MQM Server

Build TMS\_MQM server and put it in a directory included in the PATH set in `setenv` (for example, `$TUXDIR/bin` and `$KIXDIR/bin`) with correct execute permissions.

```
buildtms -r MQSeries_XA_RMI -o TMS_MQM
```

### Rebuild ART for CICS Transaction Servers

Build transaction server (ARTSTR\*/ARTATR\*/ARTDPL) and put them in `$KIXDIR/bin` directory or a local directory under `$APPDIR` (but then add it in the `$PATH` definition in `setenv`) with correct execute permissions. See an example for ARTATRN.

```
buildartcics -M -r Oracle_XA -r MQSeries_XA_RMI_COB -o ARTATRN_ORA_MQM
```

#### Notes:

- `-M` means "multiple RM involved".
- `-r` flags specify RMs to link with. For example, `-r Oracle_XA` points to Oracle DB RM definition, and `-r MQSeries_XA_RMI_COB` points to MQ RM for COBOL programs.

### Rebuild ARTCKTI Server

Build ARTCKTI server if MQ-initiated transaction support is required.

In general, default ARTCKTI does not need to be rebuilt unless WebSphere MQ version is changed or you need to use it in MQ server mode. (Default version is for use with WebSphere MQ client.)

To build the ARTCKTI server, execute the following command as the Oracle Tuxedo administrator with write permission for the `$KIXDIR/bin` directory:

```
buildserver -o $KIXDIR/bin/ARTCKTI -t -f "$KIXDIR/objs/ARTCKTI.o
$KIXDIR/objs/list.o" -l "-L$MQMDIR/lib64 -lmqic_r"
```

The above is to build for use with WebSphere MQ client. For use with locally installed WebSphere MQ server, use the library shown below.

```
buildserver -o $KIXDIR/bin/ARTCKTI -t -f "$KIXDIR/objs/ARTCKTI.o
$KIXDIR/objs/list.o" -l "-L$MQMDIR/lib64 -lmqm_r"
```

**Note:** \$MQMDIR is the path where WebSphere MQ has been installed.

For more information, see [ARTCKTI Configuration](#).

## Update Oracle Tuxedo UBBCONFIG and OPENINFO

ARTCKTI server does not need to be configured in the TMS group.

ARTSTR\*/ARTATR\*/ARTDPL should be configured in the TMS group, and TMS group should be configured as MQM group.

### Listing 4-44 Example of Updating Oracle Tuxedo UBBCONFIG and OPENINFO

---

```
*GROUPS

# For ARTCKTI

GRP01

GRPNO=10

ENVFILE="/xxx/envfile"

#For ARTSTR/ATR/DPL

GRP02

GRPNO=12

ENVFILE="/xxx/envfile "

    TMSNAME=TMS_ORA

    TMSCOUNT=2
```

## Implementing CICS Applications

```
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/SYSADM1/SYSADM1+SqlNet=ANA99C10+SesTm=
600+LogDir=/home/oracle/DRIVERMQ/deploy/CICS_RT/LOGS/xa+DbgFl=0x20"

MRM=Y
```

```
*RMS
```

```
RM_MQM
```

```
SRVGRP=GRP02
```

```
RMID=2
```

```
TMSNAME="TMS_MQM"
```

```
TMSCOUNT=2
```

```
# For local MQ connection: The OPENINFO only needs to configure RM name and
MQ manager name as following:
```

```
OPENINFO="MQSeries_XA_RMI_COB: MYMQM"
```

```
# For remote MQ connection: The OPENINFO needs to configure as following:
```

```
OPENINFO="MQSeries_XA_RMI_COB:qmname=MYMQM,channel=APP.C,trptype=TCP,AXLIB
=/home/bofzhu/zhubf/tuxedo/tux1213L31/lib/libtux.so,conname=10.182.73.205(
8000),tpm=Tuxedo"
```

```
AUTO=Y
```

```
*SERVERS
```

```
ARTCKTI
```

```
SRVGRP=GRP00
```

```
SRVID=1010
```

```
GRACE=0 RESTART=Y CONV=N MAXGEN=10
```

```
# -m means MQ manager name, -q means MQ queue name
```

```
# Refer to ARTCKTI Configuration in Oracle Tuxedo Application Runtime for
CICS Reference Guide
```

```
CLOPT="-A -- -m MYMQM -q INIT1"
```

#Add all required ART\*\_MQM servers (here ARTSTRN, ARTATRn, and ARTDPL are shown)

ARTATRn\_ORA\_MQM

SRVGRP=GRP02

SRVID=60

MIN=3 MAX=3

CLOPT="-o xxx -e xxx -r -- -s xxx -l xxx"

ARTSTRN\_ORA\_MQM

SRVGRP=GRP02

SRVID=65

MIN=2 MAX=2

CLOPT="-o xxx -e xxx -r -- -s xxx -l xxx"

ARTDPL\_ORA\_MQM

SRVGRP=GRP02

SRVID=70

MIN=2 MAX=2

CLOPT="-o xxx -e xxx -r -- -s xxx -l xxx"

## Handling CICS Runtime Preprocessor of MQOPEN/MQCLOSE Calls

For proper connection to WebSphere MQ, the sequence is MQCONN, MQOPEN, MQxxx (GET/PUT), MQCLOSE, and MQDISC. In mainframe CICS, MQCONN and MQDISC are often handled by CICS as resource management functions and applications only do MQOPEN/MQxxx/MQCLOSE.

To support this in ART for CICS, we use ART for CICS pre-processor to convert MQOPEN/MQCLOSE calls to KIX\_MQxxxx wrappers, which then handle MQCONN and MQDISC under

the covers. This approach also enables ART for CICS to handle connection issues and re-connect if necessary. Check the pre-processor output to verify that `KIX_MQxxx` calls are there.

In terms of the MQ wrapper, MQ wrapper can help CICS transaction to recycle or free the MQ connection if the CICS transaction does not do this itself. `prepro-cics.pl` introduces a switch `MQ_wrapper` to enable this MQ wrapper. For every application server (for example, `ARTSTR*/ARTATR*/ARTDPL`), `CLOPT -m queue_manager_name` is introduced to specify the target MQ Manager, because MQ wrapper can help to do the `MQCONNECT` before `MQOPEN` but `MQCONNECT` needs to know which MQ Manager that it should connect to. For more information, see [MQ\\_wrapper](#) and [WebSphere MQ Queue Manager Name](#).

## Encoding Character Set

When using local WebSphere MQ client for remote connection to z/OS based MQ Manager, the EBCDIC-to-ASCII conversion is not automatically enabled. It can be enabled by setting `MQGMO-CONVERT` flag in `MQGET` options as shown in example below.

### Listing 4-45 Example for MQGMO-CONVERT

---

```
COMPUTE MQGMO-OPTIONS      = MQGMO-WAIT
                             + MQGMO-SYNCPOINT
                             + MQGMO-FAIL-IF-QUIESCING
                             + MQGMO-CONVERT
END-COMPUTE.
```

---

For `MQPUT` the ASCII-to-EBCDIC conversion is done automatically if `MQMD-FORMAT` is set to `MQFMT-STRING`. For example,

```
MOVE MQFMT-STRING          TO MQMD-FORMAT.
```

When using local WebSphere MQ server, with channel defined as `SENDERS`, transcoding can be done without program change by adding `CONVERT (YES)` on the channel definition.

## Changing COMP-5 back to BINARY Data Type

Oracle Tuxedo ART Workbench adapts COBOL programs to target environment, in part by changing some mainframe numeric data types (BINARY/COMP) to compatible data type COMP-5, which is the native equivalent. This is transparent in COBOL applications.

However, on Linux, for Websphere MQ libraries this can cause an issue as they require the use of BINARY types in the parameters passed in MQ calls. Similar data type mapping is done by Pro\*COBOL pre-processor.

Therefore, change the WebSphere MQ interface definitions from COMP-5 back to BINARY before the final compile stage. See an example of the changed MQ interface definitions (BINARY data type).

### Listing 4-46 Example of the Changed MQ Interface Definitions (BINARY Data Type)

---

```

01  QM-NAME           PIC X(48) VALUE SPACES.
01  HCONN            PIC S9(9) BINARY.
01  Q-HANDLE         PIC S9(9) BINARY.
01  OPTIONS          PIC S9(9) BINARY.
01  COMPLETION-CODE  PIC S9(9) BINARY.
01  OPEN-CODE        PIC S9(9) BINARY.
01  REASON-CODE      PIC S9(9) BINARY.
01  CONN-REASON      PIC S9(9) BINARY.
01  USER-DATA-LENGTH PIC S9(9) BINARY.
01  DATA-LENGTH     PIC S9(9) BINARY.
01  TOTAL-NUM        PIC S9(9) BINARY.

```

---

## Implementing Using Multiple Session Management

ART for CICS provides multiple session management function. When connecting to ART for CICS via one 3270 terminal, you can select and run different transactions through this terminal

without terminating the previous transaction, and switch between active transactions back and forth.

### Notes:

- The multiple session management function does not support conversational user transaction.
- The default user can only run CESN/CSGM transactions when the multiple session management function is enabled.
- PA1/PA2 cannot be used in user application when this feature is enabled.

## Writing User Plug-In for Application List

ART for CICS provides a system transaction, whose program name is `DFHALST`, to get and show application list for a user when doing multiple session management. The transaction calls a user plug-in to get the list.

You should provide this plug-in, and place the library in the correct library path so that `DFHALST` can call it. For more information about this plug-in, see "[CICS Runtime Integration with Application List Transaction](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Configuring CICS Runtime Configuration Files

### Transaction Configuration File

You should define application list transaction (`ALST`) in `transactions.desc`, and set the program to `DFHALST`. `ARTSTRN` servers will load this transaction.

```
ALST;SIMPAPP;Application list transaction;DFHALST
```

For more information, see "[ALST \(Application List Transaction\)](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

### System Configuration File

You should configure `GMTRAN=CESN` in `system.desc` to let ART for CICS initiate `CESN` transaction automatically when a user connects to it.

```
[kixr]
```

```
APPLID=DBDCKixR
```



GMTRAN=CESN

## Configuring UBBCONFIG

You should enable security to do multiple session management. For more information, see "[Security Configuration](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

You should configure the following servers to UBBCONFIG. See [Table 4-47](#) for an example.

- ARTTCPL, specifying its `-t` option. For more information, see "[ARTTCPL/ARTTCPL Configuration](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.
- ARTCNX, specifying its `SYSID`.
- ARTSRM (to prevent the same user connects to ART for CICS via different terminals).
- ARTSTRN (to run application list transaction and user transactions).

### Listing 4-47 Example of Configuring UBBCONFIG

---

```
* SERVERS

ARTTCPL

    SRVGRP=TCP00

    SRVID=101

    CLOPT=" -- -M 4 -m 1 -L // hostname:34582 -n // hostname:34583 -t ALST"

ARTCNX

    SRVGRP=GRP01

    SRVID=15

    CONV=Y

    MIN=1 MAX=1 RQADDR=QCXN015 REPLYQ=Y

    CLOPT="-o /home2/work9/demo/LOGS/sysout/stdout_cnx -e
/home2/work9/demo /LOGS/sysout/stderr_cnx -r -- -s KIXR "
```

ARTSRM

## Implementing CICS Applications

```
SRVGRP=GRP02

SRVID=18

MIN=1 MAX=1

CLOPT="-o /home2/work9/demo/LOGS/sysout/stdout_srm -e
/home2/work9/demo/LOGS/sysout/stderr_srm -r -- -s KIXR -l SIMPAPP"
```

ARTSTRN

```
SRVGRP=GRP02

SRVID=20

CONV=Y

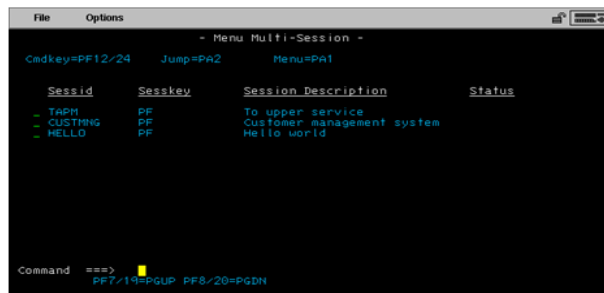
MIN=2 MAX=10 RQADDR=QKIX110 REPLYQ=Y

CLOPT="-o /home2/work9/demo/LOGS/sysout/stdout_strn -e
/home2/work9/demo/LOGS/sysout/stderr_strn -r -- -s KIXR -l SIMPAPP"
```

---

## Starting, Switching, and Ending Sessions

Boot the application, and connect to ART for CICS through a 3270 terminal. When you complete sign on, ART for CICS runs application list transaction (ALST) automatically, which shows you the application list.



```
File Options
- Menu Multi-Session -
Cmdkey=PF12/24 Jump=PA2 Menu=PA1

  Sessid      Sesskey      Session Description      Status
- TAPM       PF           To upper service
- CUSTING    PF           Customer management system
- HELLO      PF           Hello world

Command ===>
PF7/19=PGUP PF8/20=PGDN
```

### Starting Sessions

To start up a transaction (also known as session), do one of the followings.

- Move the cursor to the field at the left of the session that you want to activate, and press Enter. `DFHALST` activates the session and switches the display to the application screen.
- Type `/sessid` in the command field and press Enter.

## Switching Sessions

To switch back to application list transaction from a user transaction screen, press `PA1`.

To switch to the next active transaction, press `PA2`.

## Ending Sessions

In user transaction, issue `CICS RETURN` without `TRANSID` parameter to end the transaction, and then ART for CICS terminates the session of the transaction.

You are not able to exit application list transaction; instead, you should disconnect from ART for CICS to terminate it.

# Implementing Using ART for CICS TCP/IP Socket Interface

In z/OS, CICS TCP/IP socket interface allows remote users to access CICS client/server applications over TCP/IP Internets. CICS TCP/IP provides a variant of the Berkeley Software Distribution 4.3 sockets interface, which is widely used in TCP/IP networks and is based on the UNIX system and other operating systems. The socket interface consists of a set of calls that your CICS application programs can use to set up connections, send and receive data, and perform general communications control functions.

ART for CICS now supports CICS TCP/IP socket interface, enabling you to use it when you develop new peer-to-peer applications in which both ends of the connection are programmable.

CICS Runtime server, `ARTCSKL`, is ART for CICS TCP/IP socket listener. When client request comes, it passes the request to work task for processing, and then waits for another client request. CICS Runtime transaction server, `ARTATRN/ARTATR1`, starts up and runs user-written transactions.

You can use ART for CICS TCP/IP socket interface to write these transactions. The user-written transaction uses `EXEC CICS RETRIEVE` commands to make a socket available to `ARTATRN/ARTATR1` (it should use the output parameter to call `takesocket()`), uses `write/read()` to transfer data, and then closes the socket.

- [ART for CICS TCP/IP Socket API](#)
- [The Client-Listener-Server Application Set](#)
- [ART for CICS TCP/IP Listener \(ARTCSKL\)](#)
- [Required Configurations](#)

### Notes:

- ART for CICS TCP/IP only provides connection-oriented (TCP) services and does not support the IP (raw socket) protocol and connectionless (UDP) services. Supports both IPv4 and IPv6.
- Client applications and user transactions running in `ARTATRNL/ARTATR1` can be written in both C and COBOL languages. The `errno` returned by C socket API on all supported platforms and the `errno` returned by COBOL socket API on AIX/Solaris platforms are different from mainframe; you should use macro definition in `errno.h` in your programs.
- Only supports to start user transactions using `EXEC CICS START` with no delay interval.
- Only supports ASCII routines for those peer-to-peer applications (on both ends).
- `ARTCSKL` is the only supported socket listener; user-written listener is not supported.
- Security function is not supported.
- If a user-written program depends on platform endianness and you want to migrate it to Linux platforms, you should change your code, reconsidering its order (on mainframe it is big endian); you do not need to worry about this issue if you want to migrate it to other platforms.

## ART for CICS TCP/IP Socket API

CICS TCP/IP socket API is a collection of socket calls that enable you to perform the following primary communication functions between application programs.

- Set up and establish connections to other users on the network
- Send and receive data to and from other users
- Close down connections

In addition to these basic functions, these APIs enable you to

- Interrogate the network system to get names and status of relevant resources
- Perform system and control functions as required

ART for CICS supports the above functions as well, providing a set of C APIs and extended COBOL APIs. [Table 4-7](#) lists the supported C APIs; the last three functions are provided by ART for CICS while other functions can use OS socket library directly. [Table 4-8](#) lists the supported extended COBOL APIs.

**Table 4-7 Supported C APIs**

Call	Format	Description
<code>accept()</code>	<code>int accept(int s, struct sockaddr_in *name, int *namelen)</code>	A server issues the <code>accept()</code> call to accept a connection request from a client. The call uses a socket already created with a <code>socket()</code> call and marked by a <code>listen()</code> call.
<code>bind()</code>	<code>int bind(int s, struct sockaddr_in *name, int namelen)</code>	The <code>bind()</code> call binds a unique local port to an existing socket. Note that, on successful completion of a <code>socket()</code> call, the new socket descriptor does not have an associated port.
<code>close()</code>	<code>int close(int s)</code>	A <code>close()</code> call shuts down a socket and frees all resources allocated to the socket. If the socket refers to an open TCP connection, the connection is closed. If a stream socket is closed when input data is queued, the TCP connection is reset rather than being cleanly closed.
<code>connect()</code>	<code>int connect(int s, struct sockaddr_in *name, int namelen)</code>	A <code>connect()</code> call attempts to establish a connection between a local socket and a remote socket. For a stream socket, the call performs two tasks.
<code>fcntl()</code>	<code>signed int fcntl(int s, int cmd, int arg)</code>	The <code>fcntl()</code> call controls whether a socket is in blocking or nonblocking mode.
<code>gethostid()</code>	<code>unsigned long gethostid()</code>	<code>gethostid()</code> gets the unique 32-bit identifier for the current host in network byte order. This value is the default home IP address.

**Table 4-7 Supported C APIs**

<b>Call</b>	<b>Format</b>	<b>Description</b>
gethostname()	int gethostname(char *name, int namelen)	gethostname() returns the name of the host processor on which the program is running.
getpeername()	int getpeername(int s, struct sockaddr *name, int *namelen)	getpeername() returns the name of the peer connected to a specified socket.
getsockname()	int getsockname(int s, struct sockaddr_in *name, int *namelen)	A getsockname() call returns the current name for socket s in the sockaddr structure pointed to by the name parameter.
getsockopt()	int getsockopt(int s, int level, int optname, char *optval, int *optlen)	getsockopt() gets options associated with a socket.
ioctl()	int ioctl(int s, unsigned long cmd, char *arg)	The ioctl() call controls the operating characteristics of sockets.
listen()	int listen(int s, int backlog)	The listen() call indicates a readiness to accept client connection requests.
read()	int read(int s, char *buf, int len)	The read() call reads data on a specified connected socket.
recv()		The recv() call receives data on a specified socket.
recvfrom()	int recvfrom(int s, char *buf, int len, int flags, struct sockaddr_in *name, int *namelen)	The recvfrom() call receives data on a specified socket. The recvfrom() call applies to any datagram socket, whether connected or unconnected.
select()	int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout)	The select() call is useful in processes where multiple operations can occur, and it is necessary for the program to be able to wait on one or several of the operations to complete.

**Table 4-7 Supported C APIs**

<b>Call</b>	<b>Format</b>	<b>Description</b>
send()	int send(int s, char *msg, int len, int flags)	send() sends data on an already-connected socket.
sendto()	int sendto(int s, char *msg, int len, int flags, struct sockaddr_in *to, int tolen)	sendto() sends data to the address specified in the call.
setsockopt()	int setsockopt(int s, int level, int optname, char *optval, int *optlen)	setsockopt() sets the options.
shutdown()	int shutdown(int s, int how)	The shutdown() call shuts down all or part of a duplex connection.
socket()	int socket(int domain, int type, int protocol)	The socket() call creates an endpoint for communication and returns a socket descriptor representing the endpoint.
write()	int write(int s, char *buf, int len)	write() writes data on a connected socket.
getclientid()	int getclientid(int domain, struct clientid)	A getclientid() call returns the identifier by which the calling application is known to the TCP/IP address space.
initapi()	int initapi(int max_sock, char *subtaskid)	The initapi() call connects your application to the TCP/IP interface.
takesocket()	int takesocket(struct clientid *client_id, int hisdesc)	takesocket() acquires a socket from another program.

**Note:** takesocket() call can only be use in ARTATRN/ARTATR1. givesocket() call is not supported.

**Table 4-8 Supported Extended COBOL APIs**

<b>Call</b>	<b>Format</b>	<b>Description</b>
ACCEPT	CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.	A server issues the ACCEPT call to accept a connection request from a client.
BIND	CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.	In a typical server program, the BIND call follows a SOCKET call and completes the process of creating a new socket.
CLOSE	CALL 'EZASOKET' USING SOC-FUNCTION S ERRNO RETCODE.	The CLOSE call shuts down a socket and frees all resources allocated to it.
CONNECT	CALL 'EZASOKET' USING SOC-FUNCTION S NAME ERRNO RETCODE.	The CONNECT call is issued by a client to establish a connection between a local socket and a remote socket.
FCNTL	CALL 'EZASOKET' USING SOC-FUNCTION S COMMAND REQARG ERRNO RETCODE.	The blocking mode of a socket can either be queried or set to nonblocking using the FNDELAY flag described in the FCNTL call.
LISTEN	CALL 'EZASOKET' USING SOC-FUNCTION S BACKLOG ERRNO RETCODE.	The LISTEN call: <ul style="list-style-type: none"> <li>• Completes the bind, if BIND has not already been called for the socket.</li> <li>• Creates a connection-request queue of a specified length for incoming connection requests.</li> </ul>
READ	CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF ERRNO RETCODE.	The READ call reads the data on sockets.
RECV	CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF ERRNO RETCODE.	The RECV call, like READ, receives data on a socket with descriptor S.
RECVFROM	CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF NAME ERRNO RETCODE.	The RECVFROM call receives data on a socket with descriptor S and stores it in a buffer.



**Table 4-8 Supported Extended COBOL APIs**

<b>Call</b>	<b>Format</b>	<b>Description</b>
SELECT	CALL 'EZASOKET' USING SOC-FUNCTION MAXSOC TIMEOUT RSNDSK WSNDSK ESNDSK RRETMSK WRETMSK ERETMSK ERRNO RETCODE.	In a process where multiple I/O operations can occur, it is necessary for the program to be able to wait on one or several of the operations to complete.
SEND	CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF ERRNO RETCODE.	The SEND call sends data on a specified connected socket.
SENDTO	CALL 'EZASOKET' USING SOC-FUNCTION S FLAGS NBYTE BUF NAME ERRNO RETCODE.	SENDTO is similar to SEND, except that it includes the destination address parameter.
SHUTDOWN	CALL 'EZASOKET' USING SOC-FUNCTION S HOW ERRNO RETCODE.	The SHUTDOWN call can be used to close one-way traffic while completing data transfer in the other direction.
SOCKET	CALL 'EZASOKET' USING SOC-FUNCTION AF SOCTYPE PROTO ERRNO RETCODE.	The SOCKET call creates an endpoint for communication and returns a socket descriptor representing the endpoint.
WRITE	CALL 'EZASOKET' USING SOC-FUNCTION S NBYTE BUF ERRNO RETCODE.	The WRITE call writes data on a connected socket.
GETCLIENTID	CALL 'EZASOKET' USING SOC-FUNCTION CLIENT ERRNO RETCODE.	GETCLIENTID call returns the identifier by which the calling application is known to the TCP/IP address space in the calling program.

**Table 4-8 Supported Extended COBOL APIs**

Call	Format	Description
INITAPI		The INITAPI calls connect an application to the TCP/IP interface.
TAKESOCKET	<pre> WORKING-STORAGE SECTION. 01 SOC-FUNCTION PIC X(16) VALUE IS 'TAKESOCKET'. 01 SOCRECV PIC 9(4) BINARY. 01 CLIENT. 03 DOMAIN PIC 9(8) BINARY. 03 NAME PIC X(8). 03 TASK PIC X(8). 03 RESERVED PIC X(20). 01 ERRNO PIC 9(8) BINARY. 01 RETCODE PIC S9(8) BINARY. PROCEDURE DIVISION. CALL 'EZASOKET' USING SOC-FUNCTION SOCRECV CLIENT ERRNO RETCODE.                     </pre>	The TAKESOCKET call acquires a socket from another program and creates a new socket.

**Note:** GETSOCKOPT, IOCTL, SETSOCKOPT, and GIVESOCKET are not supported.

## The Client-Listener-Server Application Set

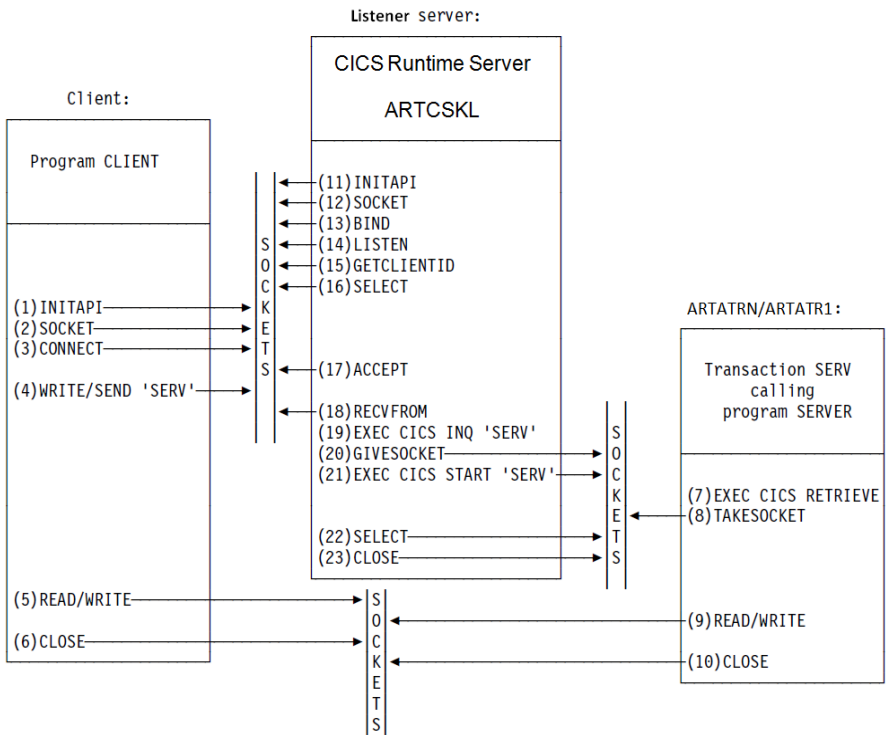
Figure 4-5 shows the sequence of CICS Runtime commands and socket calls involved in setup. CICS Runtime commands are prefixed by EXEC CICS; all other numbered items in this figure are ART for CICS TCP/IP calls.

In this case, "Client" might be running TCP/IP under the OS/2 operating system or one of the various UNIX operating systems such as AIX\*. "CICS Runtime Server ARTCSKL" and "ARTATRN/ARTATR1" processes both run under ART for CICS TCP/IP.

- [Client Call Sequence](#)

- Listener Call Sequence
- User Transaction Running in ARTATR1/ARTATR1 Call Sequence

Figure 4-5 The Client-Listener-Server Application Set



## Client Call Sequence

**Table 4-9 Client Call Sequence**

Call	Description
(1) INITAPI	Connect the CICS application to the TCP/IP interface. Use the MAX-SOCK parameter to specify the maximum number of sockets to be used by the application.
(2) SOCKET	<p>This obtains a socket. You define a socket with 3 parameters:</p> <ul style="list-style-type: none"> <li>• The domain, or addressing family</li> <li>• The type of socket</li> <li>• The protocol</li> </ul> <p>For CICS TCP/IP, the domain can only be the TCP/IP internet domain (AF_INET). The type can be stream sockets (SOCK_STREAM), or datagram sockets (SOCK_DGRAM). The protocol can be either TCP or UDP.</p> <p>Passing 0 for the protocol selects the default protocol.</p> <p>If successful, the SOCKET call returns a socket descriptor, <i>s</i>, which is always a small integer. Notice that the socket obtained is not yet attached to any local or destination address.</p>
(3) CONNECT	Client applications use this to establish a connection with a remote server. You must define the local socket <i>s</i> (obtained above) to be used in this connection and the address and port number of the remote socket. The system supplies the local address, so on successful return from CONNECT, the socket is completely defined, and is associated with a TCP connection (if stream) or UDP connection (if datagram).
(4) WRITE	This sends the first message to ARTCSKL if it runs in standard mode. The listener in standard mode requires ARTCSKL input format from the client in its first transmission. The message contains the CICS transaction code as its first 4 bytes of data. You must also specify the buffer address and length of the data to be sent.
(5) READ/WRITE	These calls continue the conversation with the server until it is complete.
(6) CLOSE	This closes a specified socket and so ends the connection. The socket resources are released for other applications.

## Listener Call Sequence

The Listener server, `ARTCSKL`, is provided as part of ART for CICS TCP/IP. [Figure 4-5](#) shows the calls issued by `ARTCSKL`. Your client and server call sequences must be prepared to work with this sequence. For more information, see [ART for CICS TCP/IP Listener \(ARTCSKL\)](#).

## User Transaction Running in ARTATRN/ARTATR1 Call Sequence

**Table 4-10 User Transaction Running in ARTATRN/ARTATR1 Call Sequence**

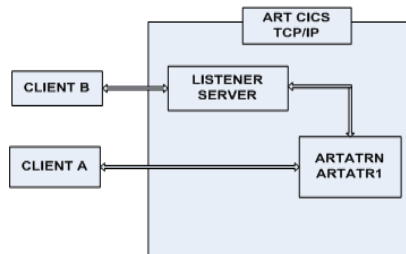
Call	Description
(7) EXEC CICS RETRIEVE	This retrieves the data passed by the EXEC CICS START command in the Listener program. This data includes the socket descriptor and the Listener client ID as well as optional additional data from the client.
(8) TAKESOCKET	This acquires the newly created socket from the Listener.  The TAKESOCKET parameters must specify the socket descriptor to be acquired and the client ID of the Listener. This information was obtained by the EXEC CICS RETRIEVE command.
(9) READ/WRITE	The conversation with the client continues until complete.
(10) CLOSE	Terminates the connection and releases the socket resources when finished.

## ART for CICS TCP/IP Listener (ARTCSKL)

### Description

`ARTCSKL` is the listener of ART for CICS TCP/IP socket and can perform the same functions as CICS TCP/IP listener `CSKL`. When client request comes, it passes the request to work task for processing, and then waits for another client request. `ARTCSKL` can run in standard or enhanced mode; you can set the mode through `FORMAT` parameter of ART for CICS TCP/IP socket listener configuration file (`listener.desc`).

**Note:** `ARTCSKL` is the only supported socket listener; user-written listener is not supported.



As shown in this figure, client A has already established a connection with the server, which has created a user transaction run in ARTATRN/ARTATR1. This allows the server to process client B's request without waiting for client A's transaction to complete. More than one user transaction can be started in this way.

ARTCSKL is written to make some of this activity go on in parallel, and it has a listening socket that has a port to receive incoming connection requests. When a connection request is received, ARTCSKL creates a new socket representing the endpoint of this connection and passes it to the applications by way of TCP/IP socket `givesocket/takesocket` calls.

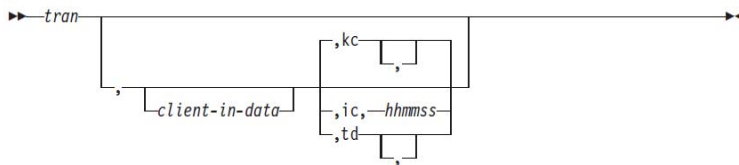
The listener performs the following functions.

- It issues appropriate TCP/IP calls to listen on the port specified in the configuration file and waits for incoming connection requests issued by clients. The port number must be configured in ART for CICS TCP/IP socket listener configuration file (`listener.desc`).
- When an incoming connection request arrives, the listener accepts it and obtains a new socket to pass to the ARTATRN/ARTATR1 server.
- The listener in standard mode starts the user transaction based on information in the first message on the new connection. The format of this information is given in following [ARTCSKL Input Format](#). For the listener in enhanced mode starts the user transaction based on information in ART for CICS TCP/IP socket listener configuration file (`listener.desc`).
- It waits for the user transaction to take the new socket and then issues the close call. When this occurs, the receiving application assumes ownership of the socket and the listener has no more interest in it.

## ARTCSKL Input Format

ARTCSKL in standard mode requires the following input format from the client in its first transmission. The client should then wait for a response before sending any subsequent transmissions. Input can be in uppercase or lowercase. The commas are required.

ARTCSKL in enhanced mode does not need this input format; ART for CICS gets transaction information from its TCP/IP Socket Listener configuration file (`listener.desc`).



### **tran**

The transaction ID (in uppercase) that the listener is going to start. This field can be one to four characters.

### **client-in-data**

Optional. Application data, the maximum length of this field is a 40-byte character (35 bytes, plus one byte filler and 4 bytes for startup type).

### **kc (only KC in uppercase is supported)**

Optional. The startup type that for ART for CICS task control. Only `KC` in uppercase is supported, indicating that the user transaction is started using `EXEC CICS START` with no delay interval. If this field is left blank, startup is immediate using the task control (`KC`).

### **hhmms (not supported)**

This is reserved for future use.

## ARTCSKL Output Format

There are two different formats for the listener output; one for user transaction started through a standard listener (see [Listing 4-48](#)) and one for user transaction started through the enhanced listener (see [Listing 4-49](#)).

A user transaction program, using the `EXEC CICS RETRIEVE` function to get the data passed to it by the listener, should expand the storage it has allocated to contain the IPv6 socket address structure. The `LENGTH` specified on the `EXEC CICS RETRIEVE` function should reflect the amount of storage allocated to contain the listener output format. The `LENGERR` flag is raised if the

LENGTH is smaller than the amount of data sent. Coding a HANDLE condition allows you to contain this.

**Note:** Output through ART for CICS Transient Data Queue (by ARTCSKL) is not supported.

### Listing 4-48 C Structure of the Listener Output Format - Standard Listener

---

```
struct sock_standard_tim {          /* declaration of structure */
    unsigned long   give_take_socket; /* Socket being given */
    char           listen_name[8];    /* Listener name */
    char           listen_taskid[8];  /* Listener task id */
    char           client_in_data[35]; /* Client data */
    char           ote[1];            /* Threadsafe ind. @W1C */
    union { /* Clients socket address */
        struct sockaddr_in   sin;
        struct sockaddr_in6  sin6;
    } sockaddr_in_parm;
    char           reserved2[68];     /* reserved */
};
```

---

### Listing 4-49 C Structure of the Listener Output Format - Enhanced Listener

---

```
struct sock_enhanced_tim {          /* declaration of structure */
    unsigned long   give_take_socket; /* Socket being given
*/
    char           listen_name[8];    /* Listener name */
    char           listen_taskid[8];  /* Listener task id */
    char           client_in_data[35]; /* Client-in-data */
    char           ote[1];            /* Threadsafe ind. @W1C */
};
```



```

union { /* Clients socket address */
    struct    sockaddr_in    sin;
    struct    sockaddr_in6   sin6;
} sockaddr_in_parm;

char    reserved2[68];          /* reserved */
short   client_in_data_length; /* Length of data recved */
char    client_in_data_2;      /* data from Client */
};

```

---

## Required Configurations

To use ART for CICS TCP/IP Socket Interface functions, you should

- Declare resources on ART for CICS TCP/IP socket listener configuration file (`listener.desc`). For more information, see "[TCP/IP Socket Listener Configuration File](#)" (`listener.desc`) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.
- Configure CICS Runtime server `ARTCSKL` and `ARTATRN/ARTATR1`. CICS Runtime server `ARTCSKL` and `ARTATRN/ARTATR1` should be configured on the same machine. For more information about `ARTCSKL` and `ARTATRN/ARTATR1` servers, see *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Implementing Transferring CICS Regions

In z/OS, `ISSUE PASS` command is used to transfer CICS regions without terminal reconnection; users can also implement data transference using `LOGONMSG`. When `ISSUE PASS` is invoked, the `GMTRAN` of destination region will be invoked by force.

ART CICS also supports the above scenario. Following configurations are required.

### Configuring ARTSRM Server

It is required to configure ARTSRM. For more information, please refer to [ARTSRM Configuration](#).

## Configuring Environment Variables

It is required to set environment variable `ISC_ENABLE` to `YES`. For more information, please refer to [ISC\\_ENABLE](#).

## CICS Runtime Configuration Files Declaration

### `system.desc`

`system.desc` defines system initialization parameters of CICS regions.

#### Listing 4-50 Example for `system.desc` Configurations

---

```
[kixr]
APPLID=DBDCKiXR
INITPARM=(ASINTP='Hello world')

[kixl]
APPLID=DBDCKiXL
INITPARM=(ASINTP='Hello world')
GMTRAN=ISSS
LGNMSG=YES
```

---

In this example, two CICS regions are defined. `SYSID` are specified as `kixr` and `kixl` respectively. On one hand, `kixl` specifies `GMTRAN=ISSS`; when users log in `DBDCKiXL`, transaction are invoked automatically. On the other hand, `kixr` doesn't specify `GMTRAN`; default `CSGM` is used. `LGNMSG` specified in `kixl` enables data transference function using `ISSUE PASS` in `EXTRACT LOGONMSG`. For more information about `system.desc`, please refer to [System Configuration File](#).

### `transactions.desc` and `programs.desc`

If `GMTRAN` (not other system transactions, such as `CSGM`, `CESN`, or `CESF`) is defined, `transactions/programs` should be configured in `transactions.desc/programs.desc`, and then loaded by `ARTSTRN/ARTSTR1`. For more information about

transactions.desc/programs.desc, please refer to [Transaction Configuration File](#) and [Programs Configuration File](#).

---

#### Listing 4-51 Example for transactions.desc and programs.desc Configurations

---

```
transactions.desc:
ISSS;SIMPAPPB;pg for simpapp;ISSPASSS
programs.desc:
ISSPASSS;SIMPAPPB;pg for simpapp;COBOL; ;ENABLED
```

---

### terminals.desc (Optional)

This configuration file defines `terminal` available to ART CICS; it is mandatory for using static LUNAME to logon ART CICS. For more information about `terminals.desc`, please refer to [Terminal Configuration File](#).

---

#### Listing 4-52 Example for terminals.desc Configurations

---

```
[terminal]
name=0001
netname=CICS0001
group=SIMPAPP

[terminal]
name=0002
netname=CICS0002
group=SIMPAPP
```

---

### UBB Declaration

To implement transferring CICS regions, the following requirements should be met.

- TMQUEUE should be configured for each CICS region.
- ARTLOGN should be configured.
- At least one ARTCNX should be configured for each CICS region.
- DDR published by ARTCNX should be configured (an example is provided as below).

### Listing 4-53 Example for DDR Configurations

---

```
GRP00
    GRPNO=10
    ENVFILE="/home2/work9/demo/config/tux/envfile"

GRP01
    GRPNO=11
    ENVFILE="/home2/work9/demo/config/tux/envfile"

GRP02
    GRPNO=12
    ENVFILE="/home2/work9/demo/config/tux/envfile"

GQUEKIXR
    GRPNO=1010
    TMSNAME=TMS_QM TMSCOUNT=2
    OPENINFO="TUXEDO/QM: /home2/work9/demo/sysfile/kixrqspace:DBDCkixR"

GQUEKIXL
    GRPNO=1020
    TMSNAME=TMS_QM TMSCOUNT=2
    OPENINFO="TUXEDO/QM: /home2/work9/demo/sysfile/kixlqspace:DBDCkixL"

...

TMQUEUE
```

```

SRVGRP=GQUEKIXR
SRVID=1110
RESTART=Y GRACE=0 CONV=N MAXGEN=10
CLOPT="-s DBDCkixR:TMQUEUE -- "
TMQUEUE
SRVGRP=GQUEKIXL
SRVID=1210
RESTART=Y GRACE=0 CONV=N MAXGEN=10
CLOPT="-s DBDCkixL:TMQUEUE -- "

ARTCNX
SRVGRP=GRP01
SRVID=15
CONV=Y
MIN=1 MAX=1 RQADDR=QCNX015 REPLYQ=Y
CLOPT="-o /home2/work9/demo /LOGS/sysout/stdout_cnx_15 -e
/home2/work9/demo /LOGS/sysout/stderr_cnx_15 -r -- -s KIXR -l SIMPAPP"
ARTCNX
SRVGRP=GRP02
SRVID=16
CONV=Y
MIN=1 MAX=1 RQADDR=QCNX016 REPLYQ=Y
CLOPT="-o /home2/work9/demo /sysout/stdout_cnx_16 -e
/home2/work9/demo /LOGS/sysout/stderr_cnx_16 -r -- -s KIXL -l SIMPAPP"
ARTLOGN
SRVGRP=GRP00
SRVID=18
CONV=Y

```

## Implementing CICS Applications

```
MIN=1 MAX=1 RQADDR=QLGN018 REPLYQ=Y

CLOPT="-o /home2/work9/demo /LOGS/sysout/stdout_logn -e
/home2/work9/demo /LOGS/sysout/stderr_logn -r --"

...

*SERVICES

DEFAULT:  SVCTIMEOUT=0 TRANTIME=80

connect  ROUTING=CICSISC

disconnect      ROUTING=CICSISC

inquire  ROUTING=CICSISC

update   ROUTING=CICSISC

CSGM     ROUTING=CICSISC

CESN     ROUTING=CICSISC

CESF     ROUTING=CICSISC

authfail      ROUTING=CICSISC

*ROUTING

CICSISC  FIELD=CX_APPLID

RANGES=" 'DBDCKIXR':GRP01, 'DBDCKIXL':GRP02, *:GRP01 " BUFTYPE="FML32 "
```

---

**Notes:** DDR configuration in UBB is mandatory. DDR routes login request to ARTCNX in different CICS regions by FML field CX\_APPLID.

The APPLID configured in ROUTING RANGES should be in upper case.

ART reserved FML FIELD ID from 8100 to 8191 for DDR.

## Environment Variable Declaration

Set environment variable ISC\_ENABLE=YES to transfer CICS regions.

Use Tuxedo `tmadmin psr` and `tmadmin psc` to check whether ARTLOGN starts successfully. ARTCNX and TMQUEUE are included in each region.

**Listing 4-54 Example for Environment Variable Declaration**

```

/home2/work9/demo> tmadmin> tmadmin
...

> psr

```

Prog Name	Queue Name	Grp Name	ID	RqDone	Load Done	Current	Service
-----	-----	-----	---	-----	-----	-----	-----
BBL	34790	KIXR	0	55	2750	( IDLE )	
TMS_QM	GQUEKIXL_T+	GQUEKIXL	30001	0	0	( IDLE )	
TMS_QM	GQUEKIXR_T+	GQUEKIXR	30001	0	0	( IDLE )	
ARTTCPL	00001.00101	TCP00	101	0	0	( IDLE )	
TMS_QM	GQUEKIXL_T+	GQUEKIXL	30002	0	0	( IDLE )	
TMS_QM	GQUEKIXR_T+	GQUEKIXR	30002	0	0	( IDLE )	
TMQUEUE	01020.01210	GQUEKIXL	1210	1	50	( IDLE )	
TMQUEUE	01010.01110	GQUEKIXR	1110	2	100	( IDLE )	
ARTADM	00011.00010	GRP01	10	0	0	( IDLE )	
ARTCNX	QCNX015	GRP01	15	0	0	( IDLE )	
ARTCNX	QCNX016	GRP02	16	0	0	( IDLE )	
ARTLOGN	QLGN018	GRP00	18	0	0	( IDLE )	
ARTSTRN	QKIX110	GRP12	20	0	0	( IDLE )	
...							

```

> psc

```

Service Name	Routine Name	Prog Name	Grp Name	ID	Machine	# Done	Status
-----	-----	-----	-----	---	-----	-----	-----

## Implementing CICS Applications

DBDckixL	TMQUEUE	TMQUEUE	GQUEK+	1210	KIXR	1 AVAIL
DBDckixR	TMQUEUE	TMQUEUE	GQUEK+	1110	KIXR	2 AVAIL
disconnect	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
connect	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
update	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
inquire	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
authfail	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
CESF	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
CESN	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
CSGM	cnxsvc	ARTCNX	GRP01	15	KIXR	0 AVAIL
disconnect	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
connect	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
update	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
inquire	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
authfail	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
CESF	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
CESN	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
CSGM	cnxsvc	ARTCNX	GRP02	16	KIXR	0 AVAIL
delsess	lognsvc	ARTLOGN	GRP00	18	KIXR	0 AVAIL
gensess	lognsvc	ARTLOGN	GRP00	18	KIXR	0 AVAIL
ART_LOGON	lognsvc	ARTLOGN	GRP00	18	KIXR	0 AVAIL
ISSS	strsvc	ARTSTRN	GRP12	25	KIXR	0 AVAIL
...						

---

## Logon ART CICS

After a successful boot up, users can connect to ART CICS, and then logon screen prompts out for users to specify the CICS region (APPLID) to logon.



Figure 4-6 Logon Screen

```

Session C - [24 x 80]
File Edit View Communication Actions Window Help
***** Welcome To *****
***** Oracle Tuxedo Application Runtime *****
*****
===> Enter CICS APPLID followed "LOGON APPL=". Example "LOGON APPL = CICSA"
===> LOGON APPL = DDBCKIXR
M A C A 23/020
Connected to remote server/host bej301358 using port 17001

```

## Implementing Intersystem Communication

ART CICS Runtime supports implementing two z/OS intercommunication features:

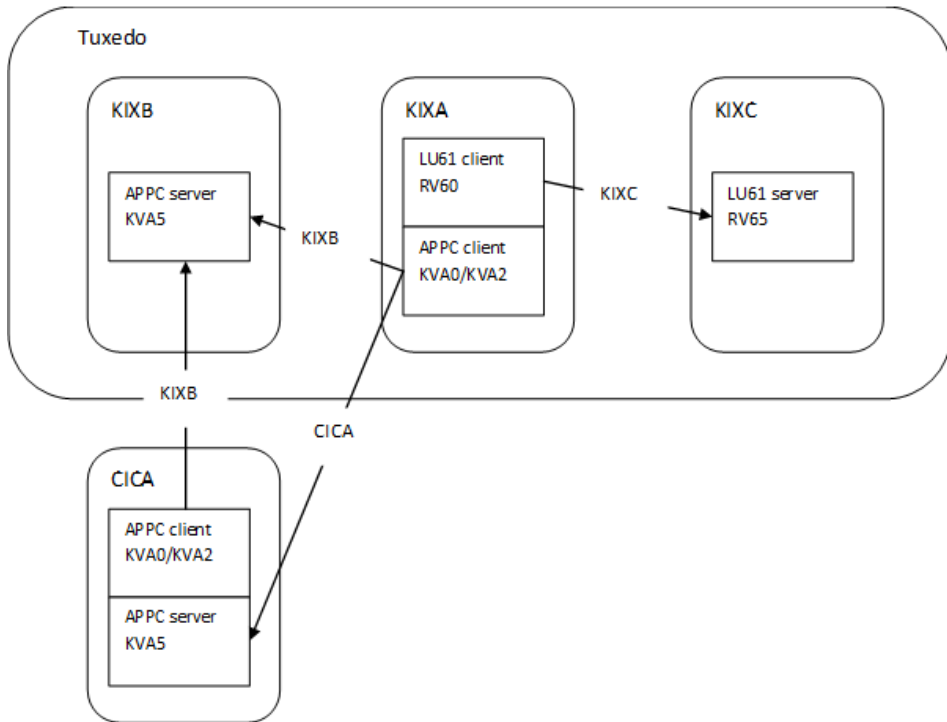
- [Implementing Distributed Transaction Processing \(DTP\)](#)
- [Implementing Asynchronous Processing](#)
- [Implementing Synchronous Processing](#)

### Implementing Distributed Transaction Processing (DTP)

ART CICS supports DTP connections in multiple ART CICS regions through APPC mapped and LUTYPE6.1 protocol. On this view, COBOL applications using DTP (APPC/LUTYPE6.1) verbs can be deployed to ART CICS directly after being translated by Oracle Tuxedo Application Rehosting Workbench.

ART CICS also supports integration with Oracle TMA to enable DTP connections between ART CICS region and Mainframe CICS region through APPC. Following is a typical end-to-end user case.

**Figure 4-7 Typical End-to-End User Case**



In this scenario, there are three ART CICS regions,  $KIXA$ ,  $KIXB$ , and  $KIXC$ , among which  $KIXA$  and  $KIXB$  communicates with each other via APPC protocol, and  $KIXA$  and  $KIXC$  communicates with each other via LU61 protocol.

Besides, there is another CICS region called CICA on Mainframe, which communicates with either  $KIXA$  or  $KIXB$  via APPC protocol.

As shown in [Figure 4-7](#), the conversations occur in these regions are:

- Conversations among ART CICS regions
  - APPC conversation, issued from the terminal in  $KIXA$ , to  $KIXB$ 
    - $KVA0$   $KIXB$   $KVA5$
    - $KVA2$   $KIXB$   $KVA5$
  - LU61 conversation, issued from the terminal in  $KIXA$ , to  $KIXC$

- RV60 KIXC RV65
- Conversations between ART CICS region and Mainframe CICS region through TMA
  - APPC outbound conversation issued from the terminal in KIXA to CICA
    - KVA0 CICA KVA5
    - KVA2 CICA KVA5
  - APPC inbound conversation issued from the terminal in CICA to KIXB
    - KVA0 CRM1 KVA5
    - KVA2 CRM1 KVA5

**Note:** CRM1 is a connection from CICA to TMA LU.

## Configurations

Following are configurations required for DTP connections to work in this scenario.

### CICS Region Definitions in system.desc

Following CICS regions are defined in the `system.desc` configuration file:

- Three ART CICS regions on open system:
  - KIXA: APPC/LU61 front end
  - KIXB: APPC back end
  - KIXC: LU61 back end
- One CICS region on Mainframe:
  - CICA: APPC front / back end

For more information about `system.desc`, refer to [System Configuration File](#).

### Connections Definitions in connections.desc

Following connections are defined in the `connection.desc` configuration file:

- Three connections are defined in KIXA:
  - KIXB: connects to KIXB, protocol is APPC
  - KIXC: connects to KIXC, protocol is LU61
  - CICA: connects to CICA, protocol is APPC

## Implementing CICS Applications

- Two connections are defined in `KIXB`:
  - `KIXA`: connects to `KIXA`, protocol is `APPC`
  - `CICA`: connects to `CICA`, protocol is `APPC`
- One connection is defined in `KIXC`:
  - `KIXA`: connects to `KIXA`, protocol is `LU61`

**Note:** Connection definition for `CICA` is not presented on local because `CICA` is an external region.

For more information about `connections.desc`, refer to [Connection Configuration File](#).

### Programs Definitions in `programs.desc`

Following programs are defined in the `programs.desc` configuration file:

- Two programs in `KIXA`:
  - `COVSATMC`: `APPC` client with `view32`
  - `RVS61C`: `LU61` client
- One program in `KIXB`:
  - `COVSATMS`: `APPC` server with `view32`
- One program in `KIXC`:
  - `RVS61S`: `LU61` server

For more information about `programs.desc`, refer to [Programs Configuration File](#).

### Transactions Definitions in `transactions.desc`

Following transactions are defined in the `transactions.desc` configuration file:

- Three transactions in `KIXA`:
  - `KVA0`: `APPC` client on `COVSATMC`, sync level 0
  - `KVA2`: `APPC` client on `COVSATMC`, sync level 2
  - `RV60`: `LU61` client on `RVS61C`
- One transaction in `KIXB`:
  - `KVA5`: `APPC` server on `COVSATMS`

- One transaction in KIXC:
  - RV65: LU61 server on RVS61S

For more information about `transactions.desc`, refer to [Transaction Configuration File](#).

## UBBCONFIG Configuration

Following are configured in the `UBBCONFIG` file:

- One `ARTSTRN` server for KIXA, with three services defined in `transactions.desc`: `KVA0`, `KVA2`, and `RV60`
- One `ARTCTRN` server for KIXB, with one service defined in `transactions.desc`: `KIXB_KVA5`
- One `ARTCTRN` server for KIXC, with one service defined in `transactions.desc`: `KIXC_RV65`
- `GWSNAX` gateway for TMA integration (for `CICA`)

For more information about `ARTSTRN` and `ARTCTRN`, refer to [CICS Runtime Servers](#).

**Note:** For `ARTCTRN` configurations in `UBBCONFIG`, it is required to specify `CONV=Y`.

## DMCONFIG Configuration

Following are configured in the `DMCONFIG` file:

- Remote domain definition for TMA integration (for `CICA`)
- Import the `CICA_KVA5` service from external
- Export the `KIXB_KVA5` service to external

## Implementing Asynchronous Processing

On z/OS, asynchronous processing refers to a `START` command that starts a transaction on a remote system. ART CICS Runtime supports implementing this feature using the `START` command with a `SYSID` option.

The following sections describe the configuration tasks you need to perform.

## Defining Regions in system.desc

Define your CICS regions in the `system.desc` configuration file. Following is an example that defines two regions, `KIXR` and `KIXX`, with respective application definitions, `DBDCKixR` and `DBDCKIXX`.

### Listing 4-55 Example of Defining Regions in system.desc

---

```
[KIXR]
    APPLID=DBDCKixR
    INITPARM=(ASINTP='Hello world')
[KIXX]
    APPLID=DBDCKIXX
    INITPARM=(ASINTP='Hello worldL')
```

---

## Configuring ARTSRM Server

It is required to configure ARTSRM server. For more information, please refer to [ARTSRM Configuration](#).

## Modifying the UBBCONFIG File

It is required to configure ARTATRN servers in the UBBCONFIG file for each CICS region.

Suppose you have defined two regions, `KIXR` and `KIXX`, as shown in [Listing 4-55](#). Following is a configuration example.

### Listing 4-56 Example of Modifying UBBCONFIG

---

```
...
*SERVERS
...
ARTATRN
    SRVGRP=GRP02
```

```

SRVID=30

CONV=N

MIN=1 MAX=1 RQADDR=QKIXATR REPLYQ=Y

CLOPT="-o
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stdout_atr
n -e /u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS
/sysout/stderr_atrn -r -- -s KIXR -l SIMPAPP"

ARTSRM SRVGRP= GRPX SRVID=36 MIN=1 MAX=1 RQADDR= QKIXATR REPLYQ=Y CLOPT="-o
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stdout_srm
-e
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stderr_srm
-r -- -s KIXR -l SIMPAPP "

ARTATRN

SRVGRP=GRPX

SRVID=35

CONV=N

MIN=1 MAX=1 RQADDR=QKIXATRX REPLYQ=Y

CLOPT="-o
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stdout_atr
n -e /u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS
/sysout/stderr_atrn -r -- -s KIXX -l SIMPAPP"

ARTSRM SRVGRP= GRPX SRVID=36 MIN=1 MAX=1 RQADDR= QKIXATRX REPLYQ=Y CLOPT="-o
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stdout_srm
-e
/u01/common/patches/yfli/KIX12110/test/CIT_ORA/strt/LOGS/sysout/stderr_srm
-r -- -s KIXX -l SIMPAPP "

...

*SERVICES

```

---

## Implementing Synchronous Processing

ART CICS Runtime supports invoking a synchronous transaction, which resides on a remote CICS system. To implement this feature, the following configurations are required.

### Configuring Environment Variables

Set the `ISC_ENABLE` environment variable to `YES` to enable the synchronous processing feature.

### Defining Regions in `system.desc`

Define your CICS regions in `system.desc` configuration file. Following is an example that defines two regions, `KIXR` and `KIXX`, with respective application definitions, `DBDCKIXR` and `DBDCKIXX`.

#### Listing 4-57 Example of Defining Regions in `system.desc`

---

```
[KIXR]
    APPLID=DBDCKIXR
[KIXX]
    APPLID=DBDCKIXX
```

---

### Modifying the `UBBCONFIG` File

Make the following configurations in the `UBBCONFIG` file:

- Configure the `ARTSTRN` servers for each CICS region
- Configure the `DDR` routing definition appropriately

Suppose you have defined two regions, `KIXR` and `KIXX`, as shown in [Listing 4-57](#). Following is a configuration example.

#### Listing 4-58 Example of Modifying `UBBCONFIG`

---

```
*GROUPS
GRPKIXR
```



```
GRPNO=11
TMSNAME="TMS_ORA"
TMSCOUNT=2
```

```
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/yfli/yfli+SqlNet=artkix+SesTm=600+LogD
ir=/LOGS/xa+DbgFl=0x20"
```

```
GRPKIXX
```

```
GRPNO=12
TMSNAME="TMS_ORA"
TMSCOUNT=2
```

```
OPENINFO="Oracle_XA:Oracle_XA+Acc=P/yfli/yfli+SqlNet=artkix+SesTm=600+LogD
ir=/LOGS/xa+DbgFl=0x20"
```

```
*SERVERS
```

```
ARTSTRN
```

```
SRVGRP=GRPKIXR
SRVID=1101
CONV=Y
MIN=1 MAX=1 RQADDR=QKIXSTRR REPLYQ=Y
CLOPT="-- -s KIXR -1 SIMPAPP"
```

```
ARTSTRN
```

```
SRVGRP=GRPKIXX
SRVID=1201
CONV=Y
MIN=1 MAX=1 RQADDR=QKIXSTRX REPLYQ=Y
CLOPT="-- -s KIXX -1 SIMPAPP"
```

```
*SERVICES  
  
DEFAULT:  SVCTIMEOUT=0  TRANTIME=80  
  
SB00      ROUTING=APPLID  
  
SB01      ROUTING=APPLID  
  
SB02      ROUTING=APPLID  
  
SB03      ROUTING=APPLID  
  
  
*ROUTING  
  
APPLID FIELD=CX_APPLID RANGES=" 'DBDCKIXX' :GRPKIXX, *:GRPKIXR"  
BUFTYPE="FML32"
```

---

In this example, requests from KIXX region are routed to ARTSTRN in GRPKIXX, and all other requests are routed to ARTSTRN in GRPKIXR.

## Implementing Submitting JCL/KSH Online

### Submitting JCL/KSH Job Online

On z/OS, CICS programs can submit JCL/KSH job by the `WRITEQ TD` command and pass the JCL/KSH job statements to JES internal reader by TDQ. ART CICS Runtime supports this function by using the special TDQ definition and the internal service advertised by TuxJES system.

Before using this feature, make sure ART Batch Runtime and TuxJES environment is set up. For more information, refer to [Using Tuxedo Job Enqueueing Service \(TuxJES\)](#).

### Configuring the UBBCONFIG File

The submitted JCL/KSH job statements are transferred to TuxJES by the ARTTDQ server. To activate this server, configure `ARTTDQ` in the `*SERVERS` section in the `UBBCONFIG` file. Following is an example.

**Listing 4-59 Example of Configuring ARTTDQ in UBBCONFIG**

---

```
*SERVERS
...
ARTTDQ
    SRVGRP=GRP02
    SRVID=50
    MIN=1 MAX=1
    CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_strn -e
/home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -s KIXR -L LIST1"
```

---

**Configuring tdqextra.desc**

To implement the submitting JCL/KSH function, you need to specify the following fields in the `tdqextra.desc` configuration file:

- **BLOCKFORMAT**: An unblocked or blocked record format for the extrapartition queues that are used as the interface to the TuxJES internal reader.
- **INTRDR**: Set the TDQ definition as the internal reader.

For example:

```
IRDR;SIMPAPP;ON LINE SUBMIT JOB;EXTRAQJ; ; ; ;V; ;32767;OUTPUT;DSN; ; ;Y;U;
```

Where:

**Y**

Indicates this is an internal reader TDQ.

**U**

Indicates the block format is UNBLOCKED.

**Note:**

- For JCL job file: To submit a JCL job to TuxJES, one JCL end flag `/*EOF` needs to be written to TDQ INTRDR. If `BLOCKFORMAT=B` in `tdqextra.desc` is set for TDQ, two end flags `/*EOF` need to be written to TDQ.

- For KSH job file: To submit a KSH job to TuxJES, one KSH end flag "#EOF" needs to be written to TDQ.

For more information, refer to [TD Queue Extra Partition Configuration File](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Submitting JCL/KSH Job Online by SPOOL

On z/OS, CICS programs can submit JCL/KSH job by `SPOOLWRITE` command and pass the JCL/KSH job statements to JES internal reader by `SPOOL`. ART for CICS supports this function by using the internal service advertised by TuxJES system.

Before using this feature, make sure ART Batch Runtime and TuxJES environment is set up. For more information, refer to [Using Tuxedo Job Enqueueing Service \(TuxJES\)](#).

**Note:** You should write end flag to `SPOOL` file to submit JCL/KSH job; otherwise, the job file written to `SPOOL` will be submitted automatically to TuxJES as a JCL job file when `EXEC CICS SPOOLCLOSE` is used.

- For JCL job file: To submit a JCL job to TuxJES, one JCL end flag `"/ *EOF"` needs to be written to `SPOOL` file.
- For KSH job file: To submit a KSH job to TuxJES, one KSH end flag `"#EOF"` needs to be written to `SPOOL` file.

## Configuring SPOOL Related Environment Variables

Configure the following environment variables.

- `KIX_SPOOL_JOB_AUTO_SUBMIT=YES`
- `KIX_SPOOL_OUTPUT_DIR=${APPHOME}/spool`

For more information, see [KIX\\_SPOOL\\_JOB\\_AUTO\\_SUBMIT](#) and [KIX\\_SPOOL\\_OUTPUT\\_DIR](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Implementing ART for CICS Control Utility

ART for CICS provides `artcicsutil` utility to track and dominate the CICS related resources from ART for Batch. It's always triggered by ART for Batch jobs. With the single command, ART for Batch jobs can open/close files, enable/disable CICS transactions, initiate CICS transactions, and etc.

`artcicsutil` provides two modes of command set to control ART for CICS. They are end-to-end mode (IPCP command set and CAFC command set) and interactive mode (interactive

commend set). For all supported subcommands for each commend set, see [artcicsutil](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

There are some typical use cases for implementing ART for CICS Control Utility.

- [Use Case 1: Implementing ART for CICS Control Utility in End-to-End Mode \(IPCP Command Set\)](#)
- [Use Case 2: Implementing ART for CICS Control Utility in Interactive Mode \(Interactive Command Set\)](#)

#### Notes:

- `artcicsutil` must work together with the ART for CICS server `ARTSRM`. `artcicsutil` is the trigger to centralize the CICS resources related operations; `ARTSRM` is the portal of each CICS region in ART for CICS, and `ARTSRM` takes charge of all operation executions.
- Before using this feature, make sure ART Batch Runtime and TuxJES environment is set up. For more information, see [Using Tuxedo Job Enqueueing Service \(TuxJES\)](#).

## Use Case 1: Implementing ART for CICS Control Utility in End-to-End Mode (IPCP Command Set)

Follow this work flow to implement `artcicsutil` utility in End-to-End mode (use IPCP commend set here for example).

- [Using ART for Workbench to convert JCL to KSH](#)
- [Configuring UBBCONFIG in CICS Runtime Domain](#)
- [Configuring Resource Files](#)
- [Configuring DMCONFIG in ART for CICS Domain and ART for Batch Domain](#)

### Using ART for Workbench to convert JCL to KSH

Suppose you have a JCL file containing IPCP commend set (see [Listing 4-60](#) for a sample).

#### Listing 4-60 Sample JCL Containing IPCP Subcommands

---

```
//IPCPEXP4 JOB 0001, 'IPCP', CLASS=A
```

## Implementing CICS Applications

```
//IPCP01 EXEC PGM=IPCPBTCH,PARM='CICS CC ONLY=CICS3'  
//STEPLIB DD DSN=IPCPvn.LOADLIB,DISP=SHR  
//IPCPCDS DD DSN=IPCPvn.COMMAND.DATASET,DISP=SHR  
//AUDIT DD SYSOUT=X  
//SYSIN DD *  
INIT KC HEL1
```

---

In this JCL, IPCP program IPCPBTCH is issued inside (EXEC PGM=IPCPBTCH), the target CICS region is set as CICS3 (EXEC PARM='CICS CC ONLY=CICS3'), and CICS transaction HEL1 is started in the target CICS region (INIT KC HEL1). In this case, the IPCP subcommands can be issued by both EXEC PARM and JCL SYSIN DD.

You should use ART for Workbench to convert this JCL file to the following KSH file.

### Listing 4-61 KSH Converted by ART for Workbench

---

```
#!/usr/bin/ksh  
  
m_JobBegin -j IPCPEXP4 -s START -v 2.0 -c A  
while true ;  
do  
    m_PhaseBegin  
    case ${CURRENT_LABEL} in  
        (START)  
# *****//  
        JUMP_LABEL= IPCPEXP4  
        ;;  
        (IPCPEXP4)  
        m_FileAssign -d SHR IPCPCDS ${DATA}/XXXX.IPCP.CICS.IPCPCNTL
```

```

m_OutputAssign -c "*" AUDIT
m_FileAssign -i SYSIN
artcicsutil -t IPCPBATCH "CICS CC ONLY=CICS3"
JUMP_LABEL=END_JOB
;;
(END_JOB)
break
;;
(*)
m_RcSet ${MT_RC_ABORT:-S999} "Unknown label : ${CURRENT_LABEL}"
break
;;
esac
m_PhaseEnd
done
m_JobEnd
#@ (#) -----

```

---

In this KSH (note `artcicsutil -t IPCPBATCH "CICS CC ONLY=CICS3"` subcommand), the program `IPCPBTCH` is translated to `artcicsutil`, `EXEC PARM` is passed to `artcicsutil` as its positional parameter, and all other subcommands (included in `SYSIN DD`) are stored by ART for Batch as a local file that `artcicsutil` can directly access. `-t` option is used to specify the type of command set; its value is set to `IPCPBTCH` (IPCP command set).

## Configuring UBBCONFIG in CICS Runtime Domain

In `UBBCONFIG`, set the following servers.

- ARTSRM (mandatory)

ARTSRM is the proxy of `artcicsutil` utility and thus must be configured in the `UBBCONFIG`. The corresponding System Configuration File (`system.desc`) should be configured as well; see [Configuring Resource Files](#) for more information.

ARTSRM must be configured in every ART for CICS region.

- ARTATRN (optional)

If command `INIT KC` or `ENAB/DISA KC` in `IPCP` commend set or `STRT` in `CAFC` commend set is used in your JCL file, you should set `ARTATRN` in `UBBCONFIG`. You should also define the target transaction in the transaction configuration file (`transactions.desc`) and programs configuration file (`programs.desc`); see [Configuring Resource Files](#) for more information.

If you use `ARTATRN`, `ARTATRN` must be configured in every ART for CICS region.

See [Listing 4-62](#) for an example. `ARTSRM` server is mandatory, because all requests that `artcicsutil` issues are received by `ARTSRM` at first, and then `ARTSRM` acts as the proxy to ask the target CICS server to handle these requests. `ARTATRN` server is also specified because `INIT KC` subcommand is used in JCL (see [Listing 4-60](#)); the target CICS transaction of `INIT KC` subcommand is the one that is just deployed at `ARTATRN` server.

### Listing 4-62 Example of Configuring `UBBCONFIG` in CICS Runtime Domain (Server `ARTSRM` and `ARTATRN`)

---

```
*SERVERS
...
ARTATRN
    SRVGRP=GRP02
    SRVID=30
    MIN=1 MAX=1
CLOPT="-o /home2/work9/demo/cics/LOGS/sysout/stdout_atrn -e
/home2/work9/demo/cics/LOGS/sysout/stderr_atrn -r -- -s KIXR -l SIMPAPP"

ARTSRM
    SRVGRP=GRP02
    SRVID=25
```



```
CLOPT="-o /home2/work9/demo/cics/LOGS/sysout/stdout_srm -e
/home2/work9/demo/cics/LOGS/sysout/stderr_srm -r -- -s KIXR -l SIMPAPP"
...
```

---

## Configuring Resource Files

Configure the following resource files.

- System Configuration File (`system.desc`)

This is mandatory. See [Listing 4-63](#) for an example. CICS Runtime region CICS3 is configured, for this CICS3 is specified in your JCL file (see EXEC

PGM=IPCPBTCH, PARM='CICS CC ONLY=CICS3' subcommand in [Listing 4-60](#)).

- Transaction Configuration File (`transactions.desc`) and Programs Configuration File (`programs.desc`)

They are optional. If command `INIT KC` in `IPCP` command set or `STRT` in `CAFC` command set is used in your JCL file, `ARTATRN` is required; you should define the target transaction in transaction configuration file (`transactions.desc`) and programs configuration file (`programs.desc`).

- VSAM Configuration File (`desc.vsam`)

This is optional. If command `OPEN/CLOS DB` in `IPCP` command set is used in your JCL file, you should configure this VSAM configuration file (`desc.vsam`).

### Listing 4-63 Example of Configuring System Configuration File

---

```
[KIXA]
APPLID=CICS1

[KIXB]
APPLID=CICS2

[KIXR]
APPLID=CICS3
```

---

## Configuring DMCONFIG in ART for CICS Domain and ART for Batch Domain

For end-to-end mode, you should configure DMCONFIG. The key service entry, `$(APPLID)_CICS_CTRL`, should be exported/imported crossing ART for CICS domain (Listing 4-64, note the `CICS3_CICS_CTRL` in bold) and ART for Batch domain (Listing 4-65, note the `CICS3_CICS_CTRL` in bold). This service is advertised by each configured ARTSRM (ARTSRM is configured in every ART for CICS region); the prefix `CICS3` is the `APPLID` of target CICS region.

Note that the `$(APPLID)` here should be in uppercase.

### Listing 4-64 Example of Configuring DMCONFIG in ART for CICS Domain

---

```
*DM_RESOURCES
VERSION=U22

*DM_LOCAL
CICS1    GWGRP= "GWGRP1 "
        TYPE=TDOMAIN
        ACCESSPOINTID= "CICSAP1 "
        DMTLOGDEV= "/home/work9/demo/cics/config/tux/DMTLOG1 "
        DMTLOGNAME= "DMTLOG1 "
        CONNECTION_POLICY= "ON_STARTUP "

*DM_REMOTE
#
BATCH1
        TYPE=TDOMAIN
        ACCESSPOINTID= "BATCHAP1 "

*DM_TDOMAIN
CICS1    NWADDR= "//optiplex:8301 "
BATCH1   NWADDR= "//optiplex:8401 "
```

```
*DM_EXPORT
CICS3_CICS_CTRL  RACCESSPOINT=BATCH1
*DM_IMPORT
```

---

#### Listing 4-65 Example of Configuring DMCONFIG in ART for Batch Domain

---

```
*DM_RESOURCES
VERSION=U22

*DM_LOCAL
BATCH1  GWGRP="GWGRP1 "
        TYPE=TDOMAIN
        ACCESSPOINTID=" BATCHAP1 "
        DMTLOGDEV="/home/work9/demo/jes/config/tux/DMTLOG1 "
        DMTLOGNAME=" DMTLOG1 "
        CONNECTION_POLICY="ON_STARTUP"

*DM_REMOTE
#
CICS1  TYPE=TDOMAIN
        ACCESSPOINTID=" CICSAP1 "

*DM_TDOMAIN
CICS1  NWADDR="//optiplex:8301 "
BATCH1  NWADDR="//optiplex:8401 "

*DM_EXPORT
```

```
*DM_IMPORT  
CICS3_CICS_CTRL  
*DM_ROUTING
```

---

## Use Case 2: Implementing ART for CICS Control Utility in Interactive Mode (Interactive Command Set)

This use case describes the way to invoke `artcicsutil` utility in interactive mode. In this example, transaction `EQDQ` is started.

### Listing 4-66 Example of Invoking `artcicsutil` Utility in Interactive Mode

---

```
> artcicsutil -t native  
> start EQDQ  
start trans:  
transaction ' EQDQ'  
trmid      ''  
from      ''  
start trans done.
```

---

## Implementing Printing CICS Runtime Applications Data

ART CICS Runtime supports printing applications data to a CICS 3270 printer using two methods:

- Printing with a `START` command
- Printing with transient data

## General Configurations

Before using either of the methods to implement printing, you need to perform the following configuration tasks:

1. Configure the printer terminal definition in `typeterms.desc`. Following is an example:

---

### Listing 4-67 Example of Configuring Printer Terminal Definition in `typeterms.desc`

---

```
[typeterm]
name=IBM-3287-1
color=YES
defscreencolumn=80
defscreenrow=24
description="IBM 327x family printer"
highlight=YES
logonmsg=NO
outline=NO
swastatus=ENABLED
uctran=NO
userarealen=100
```

---

**Note:** ART CICS does not support the alternate size for printer, so the following attributes must not be defined in `typeterms.desc` for IBM-3287-1: `scrnsize=alternate`, `altscreenrow`, and `altscreencolumn`.

2. Configure the printer transactions to the class that will never be executed concurrently:

For example:

```
TRCLASS1;UNIGRP; A tranclass bidon for UNIGRP; 1
```

3. Configure the printer program definition to the ARTSTR1 program group in `transactions.desc`.

For example:

```
PRNT;UNIGRP;pg for ARTSTR1; PRNTPROG; ; ; ; ; ;ENABLED; ; ; ;TRCLASS1
```

4. Configure the printer program to the ARTSTR1 program group in `programs.desc`.

For example:

```
PRNTPROG;UNIGRP;pg for UNIGRP;COBOL; ;ENABLED
```

5. Define a printer terminal in `terminals.desc` to indicate the printer LUNAME and TERMID.

For example:

```
[terminal]
name=PRT1
netname=CICSPRT1
group=UNIGRP
```

6. Configure the printer program group to the ARTSTR1 program group using `CLOPT -l` option in `UBBCONFIG`.

For example:

```
CLOPT="-o stdout_str1 -e stderr_str1 -r -- -s KIXR -l UNIGRP"
```

7. Do one of the following to specify a printer device:

- If you use a `wc3270` client as the printer terminal, configure the terminal type to `IBM-3287-1`.

For example:

```
-tn IBM-3287-1
```

- If you use a `PCOM` client as the printer terminal, configure a LUNAME in the `Telnet3270` interface and set the **Session Type** to **Printer** in the `Session Parameters` interface, as shown in following figures:

**Figure 4-8 Configure a LUNAME**

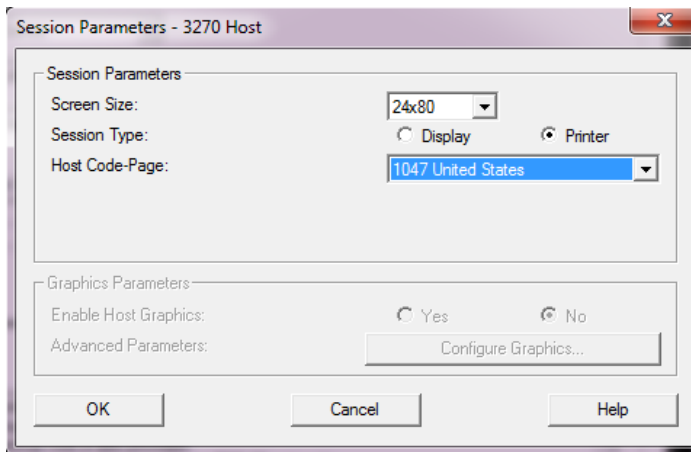
The screenshot shows the 'Telnet3270' dialog box with the 'Host Definition' tab selected. The dialog is divided into several sections:

- Host Definition:** This section contains a table for configuring host connections.
 

	Host Name or IP Address	LU or Pool Name	Port Number
Primary	xxx.xxx.xxx.xxx	CICSPRT1	12345
Backup 1			23
Backup 2			23
- Connection Options:** This section contains:
  - Connection Timeout: 6 Seconds (with a spin button)
  - Auto-reconnect
  - Try connecting to last configured host infinitely

At the bottom of the dialog are four buttons: OK, Cancel, Apply, and Help.

Figure 4-9 Setting the PCOM Session Type



## Implementing Printing with a START Command

Figure 4-10 Printing with a START Command

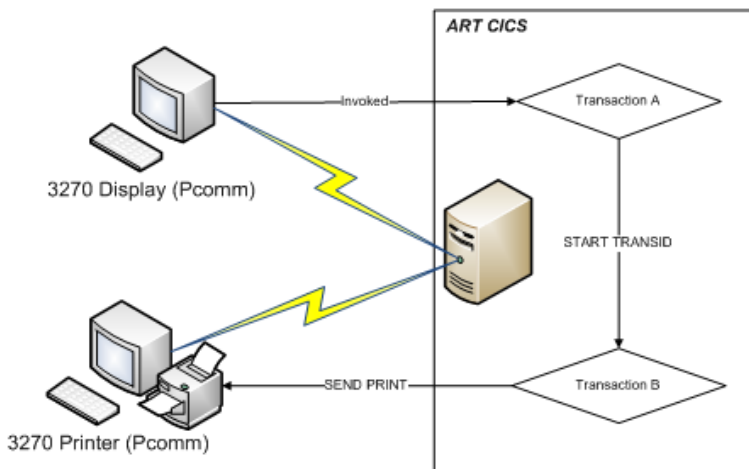


Figure 4-10 shows a typical user case. The procedure to implement printing with a `START` command in such scenario are as follows:

1. Configure the Pcomm terminal to connect to an external printer device.



2. Establish the connection from the Printer terminal to ART TCP with the `LUNAME` you specified previously.
3. Establish the connection from the Display terminal to ART CICS, and start a transaction A from the Display terminal.
4. Transaction A invokes an asynchronous transaction B with the `START` command, and specifies the `TERMID` as one of the `LUNAME` defined previously.
5. Transaction B invokes a BMS or terminal control command, then sends the command application data to the printer terminal, and finally the Printer terminal sends the print job to the connected external printer device for printing.

## Implementing Printing with Transient Data

To implement printing with transient data, do the following:

1. Configure `ATIFACILITY` and `FACILITYID` in `tdqintra.desc`, as follows:

### Listing 4-68 Example of Configuring `ATIFACILITY` and `FACILITYID` in `tdqintra.desc`

---

```
#
TDQUEUE;GROUP;DESCRIPTION;RECOVSTATUS;TRANSID;TRIGGERLEVEL;USERID;WAIT;WAI
TACTION;QSPACENAME;TRT;ATIFACILITY;FACILITYID
MTI1;SIMPAPP;TDQ FOR PRINTER;;BBBB;1;;;;;T;PRT2
```

---

2. Define `PRT2` in `terminals.desc`.

For example:

```
[terminal]
name=PRT2
netname=CICSPRT2
group=UNIGRP
```

3. After ART CICS is started, establish the connection between `PCOMM` (printer with the specified `LUNAME= CICS PRT2`) and ART CICS.
4. For ATI users, configure a `/Q` as follows:

```
qcreate MTI1 fifo none 2 30 1m 0m "TDI_TRIGGER -q queue_name -d
space_name"
```

When the ATI trigger level is reached, TDI\_TRIGGER client will be invoked, and the `-q` and `-d` options will notify ART CICS to start a transaction associated `queue_name` and `space_name` on the terminal PRT2.

# Implementing Invoking Web Services from CICS Applications

ART CICS provides support for invoking web services from CICS applications using the `INVOKE WEBSERVICE` command. To implement this feature, you need to perform the configuration tasks described in the following sections.

- [Converting WSDL File into MIF and Generating COPYBOOK](#)
- [Generating RECORD Definition from COPYBOOK](#)
- [Configuring SALT and Metadata Repositories](#)
- [Configuring webservice.desc](#)
- [Modifying UBBCONFIG](#)

## Converting WSDL File into MIF and Generating COPYBOOK

Convert your WSDL file into the Oracle Tuxedo metadata repository input file (MIF) using the Oracle SALT command utility, `wsdlcvt`. Specify its `-c` option to generate COPYBOOK. For more information, see [Configuring an Oracle SALT Application](#).

## Generating RECORD Definition from COPYBOOK

Use `cpy2record` tool to generate RECORD definition from the COPYBOOK that the previous step generates, and export environment variable for RECORD. For more information, see [Using a RECORD Typed Buffer](#).

## Configuring SALT and Metadata Repositories

Build SALT configuration file (using Oracle Tuxedo SALT utility `wsloadcf`), and loads service information into an Oracle Tuxedo service metadata repository (using `tmloadrepos`). For more information, see [Configuring an Oracle SALT Application](#).

## Configuring webservice.desc

Add a service section in `webservice.desc` configuration file, specifying `REQUEST` and `RESPONSE` in `webservice.desc`. [Listing 4-69](#) shows an example.

### Listing 4-69 Example of webservice.desc Configuration

---

```
[DFH0XCMNOperation]
    REQUEST=DFH0XCMNOperation
    RESPONSE=DFH0XCMNOperationResponse
    TRANSACTION=N
```

---

## Modifying UBBCONFIG

Configure the `TMMETADATA` and `GWWS` servers in the `UBBCONFIG` file. [Listing 4-70](#) shows an example.

### Listing 4-70 Example of Adding TMMETADATA and GWWS in UBBCONFIG

---

```
*SERVERS
...
TMMETADATA SRVGRP=GROUP2 SRVID=2 CLOPT="-A -- -f pmu.repos"
GWWS SRVGRP=GROUP2 SRVID=3 CLOPT="-A -r -- -iGWWS1"
```

---

# Implementing ART for CICS Application Server Customized Callback Support

Oracle Tuxedo Application Runtime for CICS allows you to extend one or more of your ART for CICS application servers by loading your customized functions at server initiation and unloading those functions at server shutdown. To implement this feature, do the followings.

- [Create Shared Library libkixcallback.so](#)
- [Include Customized C Library for Dynamically Loading](#)

There are some typical use cases for implementing this feature.

- [Use Case 1: Create Shared Memory at Server Initiation](#)
- [Use Case 2: Open Database Table at Server Initiation](#)

**Note:** Only the following ART for CICS application servers support this feature: ARTSTR1/N, ARTATR1/N, ARTCTR1/N, ARTWTR1/N, ARTDPL and ARTDPL\_XN.

## Create Shared Library libkixcallback.so

You need to create one and only one shared library called `libkixcallback.so`, and declare the following two callback functions in this shared library.

**Note:** Although you can extend many application servers, you should put all callback functions in the same shared library.

- [int ARTKIX\\_\\_svrinit\\_callback\(ARTKIX\\_SRVINIT\\_PARA\\*\) \(at Server Initiation\)](#)
- [void ARTKIX\\_\\_svrdone\\_callback\(\) \(at Server Shutdown\)](#)

We recommend you put this shared library under the directory where the CICS Runtime product is installed. (Environment variable `$KIXDIR` is used to declare this directory. Usually it is `$KIXDIR/lib` directory.) When you do it, do not remove this shared library from this directory when your Oracle Tuxedo Application Runtime for CICS is updating or migrating.

Oracle Tuxedo Application Runtime for CICS provides you a user header file called `artkixcallback.h` (published in directory `$KIXDIR/include`) to help you develop your shared library. It defines the following enumeration inside the header file.

### Listing 4-71 User Header File `artkixcallback.h`

---

```
typedef enum artkix_svrtype
{
    e_ARTKIX_STR1, // for Synchronous Transaction server type
    e_ARTKIX_STRN,
    e_ARTKIX_ATR1, // this is for Asynchronous Transaction servers type
}
```

```

e_ARTKIX_ATRN,
e_ARTKIX_CTRL, // for Converse Management server type
e_ARTKIX_CTRN,
e_ARTKIX_WTR1, // for Non-3270s Terminal server type
e_ARTKIX_WTRN,
e_ARTKIX_DPL // for Distributed Program Link server type
} ARTKIX_SVRTYPE;

```

---

## int ARTKIX\_\_svrinit\_callback(ARTKIX\_SRVINIT\_PARA\*) (at Server Initiation)

ART for CICS application servers call this function when initiated.

Input: The input argument is a pointer to the following structure.

### Listing 4-72 Structure

---

```

typedef struct artkix_svrinfo_t
{
    long    tux_svr_grpno;
           /* this is the server's group id. */
    long    tux_svr_id;
           /* this is the server's id. */
    char    kix_applid[8];
           /* this is the server's applid. */
    char    kix_sysid[4];
           /* this is the server's sysid. */
    ARTKIX_SVRTYPE kix_svrtype;
           /* this is the server's type. */
    int     tux_svr_argc;
}

```

```
const char **tux_svr_argv;

        /* these are command line options to be passed to server
when it is activated. */
} ARTKIX_SRVINIT_PARA;
```

---

Output: None

Return code: Returns zero if initiation succeeds. Returns nonzero if it fails (and these servers cannot start up).

### **void ARTKIX\_\_svrdone\_callback() (at Server Shutdown)**

ART for CICS application servers call this function when shutting down.

Input: None

Output: None

Return code: None

## **Include Customized C Library for Dynamically Loading**

Include your customized C library in `$LD_LIBRARY_PATH` for Linux/Solaris (or `$LIBPATH` for AIX). This library will be loaded dynamically in the runtime.

If ART for CICS application servers cannot find any customized shared library in the above paths, this feature is disabled but these servers can still successfully start up.

## **Use Case 1: Create Shared Memory at Server Initiation**

If you want to create a shared memory used when ARTDPL server starts up, you need to create a shared library named `libkixcallback.so`, which exports two callback functions, and place the extended shared library under `$KIXDIR/lib/` directory. For more information, see [Create Shared Library libkixcallback.so](#).

When the ARTDPL server starts up, it searches this extended shared library under `$KIXDIR/lib` at first. If it cannot find it, it continues to search `$LD_LIBRARY_PATH` for Linux/Solaris (or `$LIBPATH` for AIX). After finding this shared library, the ARTDPL server loads it.

At this server initiation, you can create your shared memory. If the function fails, it returns nonzero and then the ARTDPL server cannot start up; if the function succeeds, it returns zero and then the ARTDPL server starts up as usual.

## Use Case 2: Open Database Table at Server Initiation

If you want to open some database tables on your Linux platform when an ARTDPL server starts up, you need to create a named `libkixcallback.so` shared library, which exports two callback functions, and make sure the pathname of this C shared library is included in `$LD_LIBRARY_PATH` environment variable. For more information, see [Create Shared Library libkixcallback.so](#).

When the ARTDPL server starts up, it searches this customized shared library under `$(KIXDIR)/lib`. If it cannot find it, it continues to search under `$LD_LIBRARY_PATH`. After finding this shared library, the ARTDPL server loads it.

When the shared library is initiated, you can open your database tables. If the function fails, it returns nonzero and then the ARTDPL server cannot start up; if the function succeeds, it returns zero and then the ARTDPL server starts up as usual.

## Implementing Resource-Based Authorization

ART for CICS offers a security framework which allows a customer to choose integration with an external security manager. Using this framework, ART for CICS can perform authorization checking when you access a resource in a transaction. For example, if you issue CICS WRITEQ TS command, ART for CICS first consults external security manager to confirm whether you have the right to write the TS queue.

To enable resource-based authorization, do the followings.

- Set environment variable `KIX_RESSEC` to A or Y.
  - `KIX_RESSEC=A`: performs resource-based authorization when you access a resource in a transaction. This applies to all transactions.
  - `KIX_RESSEC=Y`: performs resource-based authorization when you access a resource in a transaction. This only applies to the transactions whose `RESSEC=Y` is specified in `transactions.desc`.

For more information, see [KIX\\_RESSEC](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

- If `KIX_RESSEC=Y` is set, in `transactions.desc`, configure `RESSEC=Y` for the transactions that you want to check resource-based authorization.

For more information, see [Transaction Configuration File](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

- Replace the default authorization function with your customized function.

ART for CICS has a default authorization function called `CheckResourceAuth.gnt` (under ART for CICS installation directory). To integrate with external ESM, you need to replace the default `CheckResourceAuth.gnt` with your customized function.

The function interface is listed in [Integration with the External Security Manager](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

- Enable Tuxedo Security in `UBBCONFIG`

Enable Tuxedo security. For example, configure `XAUTHSVR` in `UBBCONFIG` to enable LDAP based authentication and authorization.

For more information, see [XAUTHSVR \(5\)](#) in *Oracle Tuxedo File Formats, Data Descriptions, MIBs, and System Processes Reference*.

### Listing 4-73 Enable Tuxedo Security in `UBBCONFIG`

---

```
*RESOURCES

SECURITY      ACL

AUTHSVC       ". .AUTHSVC "

OPTIONS       EXT_AA, EXT_MON

*GROUPS

AUTHGRP

                GRPNO=25

*SERVERS

XAUTHSVR      SRVGRP=AUTHGRP

                SRVID=5

                MIN=1 MAX=1

                CLOPT="-A -- -n /opt/oracle/CICS_RT/atnldap -z
/opt/oracle/CICS_RT/atzldap"
```



---

# Implementing COBOL Program Debugging in CICS Runtime

There are some typical use cases for implementing COBOL program debugging in CICS runtime.

- Use Case 1: Two users want to debug two COBOL programs respectively.
- Use Case 2: One user wants to debug two COBOL programs in one transaction.

You can also issue `XCTL` or `LINK` (local) inside one transaction.

- Use Case 3: One user wants to debug two programs with `START TRANSID`.
- Use Case 4: One user wants to debug two programs with `LINK` (remote).

For more information, see [COBOL Program Debugging and Error Processing in CICS Runtime](#) in *Oracle Tuxedo Application Runtime for CICS User Guide* and "[Debug Configuration File](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Use Case 1: Two users want to debug two COBOL programs respectively.

If ART for CICS user A wants to debug COBOL `Program1` and ART for CICS user B wants to debug COBOL `Program2`, the two users should do the followings.

1. Add COBOL debug information into `kix_cobol_dbg.cfg` configuration file.

```
myAnimSrvID1;;;; Program1
```

```
myAnimSrvID2;;;; Program2
```

2. Input `anim` command lines.

- User A inputs the following command line from his/her terminal.

```
anim %XmyAnimSrvID1
```

- User B inputs the following command line from his/her terminal.

```
anim %XmyAnimSrvID2
```

3. User A and B either start up their own ART for CICS application servers with the same Linux account which starts up the `anim` utility, or dynamically change the debug configuration resource file without restarting the ART for CICS servers.

**Note:** The Linux user account that starts up the ART for CICS server must be the same as the Linux user account that runs the `anim` command line. Only the `ANIMSRVID` which the `anim` utility specifies will be debugged.

### Use Case 2: One user wants to debug two COBOL programs in one transaction.

If one ART for CICS user wants to debug two different COBOL programs in one transaction (for example, if ART for CICS user wants to debug COBOL `Program1` and COBOL `Program2`, and both programs are in the same transaction), do the followings.

1. Add COBOL debug information into `kix_cobol_dbg.cfg` configuration file.  

```
myAnimSrvID1;;;transaction1;
```
2. Input the following command line in one terminal, and then both programs can be debugged.  

```
anim %XmyAnimSrvID1
```

### Use Case 3: One user wants to debug two programs with START TRANSID.

If one ART for CICS user wants to debug two different COBOL programs with `START TRANSID` command, do the followings.

1. Add COBOL debug information into `kix_cobol_dbg.cfg` configuration file.  

```
myAnimSrvID1;;; program1  
myAnimSrvID2;;; program2
```
2. Input the following command lines in separate terminals respectively:  

```
anim %XmyAnimSrvID1  
anim %XmyAnimSrvID2
```

When these two programs are launched by the ART for CICS application server, the user can debug both programs in separate terminals respectively.

### Use Case 4: One user wants to debug two programs with LINK (remote).

If one ART for CICS user wants to debug two different COBOL programs with `LINK` command, do the followings.

1. Add the following COBOL debug information into `kix_cobol_dbg.cfg` configuration file.

```
myAnimSrvID1;;;; program1
myAnimSrvID2;;;; program2
```

2. Input the following command lines in separate terminals respectively:

```
anim %XmyAnimSrvID1
anim %XmyAnimSrvID2
```

When two programs are launched by the ART for CICS application server, the user can debug both programs in separate terminals respectively.

## CICS Runtime Logs

### Tuxedo System Log

Like other Tuxedo applications, CICS Runtime is managed by Tuxedo that records certain events and problems in a dedicated system log.

This log is the standard Tuxedo User Log (ULOG) whose name is contained in the system variable `ULOGPFX` of the Tuxedo `ubbconfig` file.

Example:

```
ULOGPFX="/home2/work9/demo/Logs/TUX/log/ULOG"
```

## The CICS Runtime Server Logs

When declaring a service in the Tuxedo `ubbconfig` file, each server has `CLOPT` options defined including two files:

- `-o` option for `stdout` (normal messages)

The name of this file is `stdout_<server name>` without the `ART` prefix.

For example: the `ARTSTRN` server has a standard output named `stdout_strn`.

- `-e` option for `stderr` (error messages)

The name of this file is `stderr_<server name>` without the `ART` prefix.

For example: the `ARTSTRN` server has an error output named `stderr_strn`.

The different `stdout` and `stderr` message files for each CICS Runtime server are:

**Table 4-11 Message Files by Server**

Server name	-o standard output file	-e standard error file
ARTTCPL	stdout_tcp	stderr_tcp
ARTCNX	stdout_cnx	stderr_cnx
ARTSTRN	stdout_strn	stderr_strn
ARTSTR1	stdout_str1	stderr_str1
ARTATRn	stdout_atrn	stderr_atrn
ARTATR1	stdout_atr1	stderr_atr1
ARTTSQ	stdout_tsq	stderr_tsq
ARTDPL	stdout_dpl	stderr_dpl

**Note:** In the stderr file of a server all the configuration files mounted are described. The stderr file contains not only the error messages concerning problems encountered when the servers are booted but also information about the different resources loaded. Specifically you will find:

- The groups of resources installed depending on the -l list parameter of each CICS Runtime server.
- The resources successfully installed and available for use (remember that an installed resource may be disabled for use) depending on the valorization of each .desc configuration file.

**Listing 4-74 Example of the stdout\_strn Just After Start Up for a ARTSTRN Server**

```
Groups loaded: <0001>
```

```
|-----|
|  GROUP  |
|-----|
|SIMPAPP  |
|-----|
```

ARTSTRN: Read config done

```

|-----|
| TRANCLASS loaded : < 2> |
|-----|
|          TRANCLASS          |  GROUP  |MAXACTIVE|
|-----|-----|-----|
|TRCLASS1                    |SIMPAPP |    001 |
|TRCLASS2                    |SIMPAPP |    002 |
|-----|
|-----|
| PROGRAMS loaded : < 4> |
|-----|
|          PROGRAM          |  GROUP  |LANGUAGE|EXEC|  STATUS |
|          |                |          |    |    |
|-----|-----|-----|----|-----|
|RSSAT000                    |SIMPAPP |COBOL  |USER|ENABLED |
|RSSAT001                    |SIMPAPP |COBOL  |USER|ENABLED |
|RSSAT002                    |SIMPAPP |COBOL  |USER|ENABLED |
|RSSAT003                    |SIMPAPP |COBOL  |USER|ENABLED |
|-----|
|-----|
| TRANSACTIONS loaded : < 4> |
|-----|-----|----|---|---|---|-----|
|-----|---|---|---|
|          |          |          |          |          |
|T|          |          |          |          |
|TRAN|  GROUP  |          PROGRAM          |ALIA|M|O|PRI|E|E|  STATUS
|TASK |R|  TRAN  | TWA |MAX|

```

## Implementing CICS Applications

```
|      |      |      |      |      |      |      |      |      |      | |
|A| CLASS | SIZ |ACT|      |      |      |      |      |      |      |
|      |      |      |      |      |      |      |      |      |      |
|C|      |      | IVE|      |      |      |      |      |      |
|-----|-----|-----|-----|-----|-----|-----|-----| | |
|---|---|---|---|---|---|---|---|---|---|
|SA00|SIMPAPP  |RSSAT000      |      |      |      |      |      |      |
|USER |Y|      |00000|999|      |      |      |      |      |
|SA01|SIMPAPP  |RSSAT001      |      |      |      |      |      |      |
|USER |Y|      |00000|999|      |      |      |      |      |
|SA02|SIMPAPP  |RSSAT002      |      |      |      |      |      |      |
|USER |Y|      |00000|999|      |      |      |      |      |
|SA03|SIMPAPP  |RSSAT003      |      |      |      |      |      |      |
|USER |Y|      |00000|999|      |      |      |      |      |
|-----|-----|-----|-----|-----|-----|-----|
-----|
```

Warning: zero TSQMODEL loaded!!

FILES<FILE> lineNo(1) skipping Record: Group not to load

FILES<FIC3> lineNo(4) skipping Record: Group not to load

---

We can note in this example that

- One group (SIMPAPP) is selected with the `-1` option
- Four configurations files are used: transactions, tranclasses, programs and tsqmodels.
- Information on the successful loading of these resources (Warning: zero TSQMODEL loaded).
- The detail of the resources loaded and their explicit characteristics (name, group, description ...) even default/implicit values were used in the `.desc` file leaving the fields filed with space(s).

## Disabling and Enabling Programs

Sometimes, problems are encountered in a program that significantly impacts your system and the program must be eliminated urgently by prohibiting end-users from running it. In the immediate, this helps temporarily to stabilize the system giving time to analyze and solve the dysfunction.

As on z/OS, CICS Runtime allows to disable a program. A program is disabled by modifying the CICS Runtime configuration file `programs.desc`. This file contains a dedicated field, the `STATUS` field, to indicate if a program is `DISABLED` or `ENABLED` (status by default).

See also dynamic administration of CICS resources information in the [Oracle Tuxedo Application Runtime for CICS Reference Guide](#).

### Disabling Programs

To switch your transaction from enabled to disabled, you have to modify the seventh field of this csv file, to change the previous value from an implicit (" " space(s)) or an explicit `ENABLED` status to the explicit `DISABLED` status.

After shutting down and booting the CICS Runtime Tuxedo servers, your modifications of one or more programs will be taken in account.

If you disable a program, when somebody wants to use it, the error messages displayed depend on the way that the application handles CICS errors.

---

#### Listing 4-75 Example Simple Application SA02 COBOL Program Set to `DISABLED` in `programs.desc`

---

```
#PROGRAM;GROUP;DESCRIPTION;LANGUAGE; ; ;STATUS
RSSAT000;SIMPAPP; Home Menu Program of the Simple Application ;COBOL
RSSAT001;SIMPAPP; Customer Detailed Information Program of the Simple
Application ;COBOL; ; ;ENABLED;
RSSAT002;SIMPAPP; Customer Maintenance Program of the Simple
Application;COBOL; ; ;DISABLED;
RSSAT003;SIMPAPP; Customer List of the Simple Application ;COBOL
```

---

## Enabling Programs

To enable a program, you have only to do the opposite, changing the STATUS field from DISABLED to ENABLED or " " (at least one space).

After shutting down and booting the CICS Runtime Tuxedo servers, your modifications of one or more programs take effect.

## Checking the Change in Program Status

If you consult the logs of the different transactions servers or the CICS Runtime you will note the modification of the modified status in the `stderr_*` logs.

Just after the start up of this server, the logs shows (in italics) that this program is disabled.

### Listing 4-76 Log Report Showing Program Status

---

```
Groups loaded: <0001>
|-----|
|  GROUP  |
|-----|
|SIMPAPP  |
|-----|
ARTSTRN: Read config done
|-----|
| TRANCLASS loaded : < 2> |
|-----|
|          TRANCLASS          |  GROUP  |MAXACTIVE|
|-----|-----|-----|
|TRCLASS1          |SIMPAPP |    001 |
|TRCLASS2          |SIMPAPP |    002 |
|-----|-----|-----|
|-----|
```



## Disabling and Enabling Programs

```

| PROGRAMS loaded : < 4>
|-----|
|          PROGRAM          |  GROUP  | LANGUAGE|EXEC|  STATUS  |
|                            |          |         |KEY |         |
|-----|-----|-----|----|-----|
|RSSAT000                    |SIMPAPP  | COBOL   |USER|ENABLED  |
|RSSAT001                    |SIMPAPP  | COBOL   |USER|ENABLED  |
|RSSAT002                    |SIMPAPP  | COBOL   |USER|DISABLED |
|RSSAT003                    |SIMPAPP  | COBOL   |USER|ENABLED  |
|-----|
|-----|
| TRANSACTIONS loaded : < 4> |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|
|          |          |          |          |          |
|T|          |          |          |          |
|TRAN|  GROUP  |          PROGRAM          |ALIA|M|O|PRI|E|E|  STATUS
|TASK |R|  TRAN  | TWA |MAX| | | |
|          |          |          |          |          |
|A|  CLASS  | SIZ |ACT|          |          |          |          |
|          |          |          |          |          |          |
|C|          |          |IVE|          |          |          |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|
|SA00|SIMPAPP  |RSSAT000          |          |N|N|001|N|N|ENABLED
|USER |Y|          |00000|999|
|SA01|SIMPAPP  |RSSAT001          |          |N|N|001|N|N|ENABLED
|USER |Y|          |00000|999|
|SA02|SIMPAPP  |RSSAT002          |          |N|N|001|N|N|ENABLED
|USER |Y|          |00000|999|

```

```
|SA03|SIMPAPP   |RSSAT003           |      |N|N|001|N|N|ENABLED  
|USER |Y|         |00000|999|
```

```
|-----  
-----|
```

Warning: zero TSQMODEL loaded!!

---

## Removing and Adding Applications for CICS Runtime

Sometimes, you want to delete an application from a given machine either to definitely delete all its components or to move them to another machine. If all the resources used by your application were defined in one or more resource groups dedicated to your application, you have only to suppress these groups from CICS Runtime and eventually install them elsewhere.

Each CICS Runtime Tuxedo Server reads a list of groups to be selected and installed at start up, contained in its CLOPT options after the `-l` parameter. To remove or add group(s) from an application, you have only to remove or add these groups from this list for each CICS Runtime Tuxedo server.

Your modifications on one or more programs take effect after shutting down and booting up the CICS Runtime Tuxedo servers.

### Listing 4-77 Example of Application in ARTSTRN Server

---

```
ARTSTRN   SRVGRP=GRP02  
  
          SRVID=20  
  
          CONV=Y  
  
          MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y  
  
          CLOPT=" -o /home2/work9/demo/Logs/TUX/sysout/stdout_strn  
-e /home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -  
s KIXR -l SIMPAPP"
```

---

If you want to add one or more groups, you have to concatenate these new groups to those previously defined, separating them with a ":" character.

---

**Listing 4-78 Example of Adding group1 and group2 in ARTSTRN Server**

---

```
ARTSTRN    SRVGRP=GRP02
           SRVID=20
           CONV=Y
           MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y
           CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_strn
           -e /home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -
           s KIXR -l SIMPAPP:GROUP1:GROUP2 "
```

---

If you want to remove groups, you remove them from the -l lists when they are present, leaving only one : character between the remaining groups.

---

**Listing 4-79 Example of Removing group1 in ARTSTRN Server**

---

```
ARTSTRN    SRVGRP=GRP02
           SRVID=20
           CONV=Y
           MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y
           CLOPT="-o /home2/work9/demo/Logs/TUX/sysout/stdout_strn
           -e /home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -
           s KIXR -l SIMPAPP:GROUP2 "
```

---

**Notes:**

- When the groups are removed, all the resources of these groups are only logically suppressed. If you want also to suppress them physically, you have to delete all the lines of the CICS Runtime resource configuration files containing the group names.
- When the groups are added, all the resources of these groups must be present in the different CICS Runtime resource configuration files under the group names. To avoid future problems, do not omit to declare resources in a group because they are already declared in groups from other applications.
- When groups are added or removed, be careful to indicate the same list of groups in each CICS Runtime server.

## CICS Runtime C Program Support

ART CICS allows users to implement and run CICS applications in C language.

### Running C Program in CICS Runtime

Each C program is loaded as a COBOL program and executed in COBOL runtime, so CICS C program support is COBOL production depended.

### C Programming Restrictions and Requirements

Restrictions and requirements for CICS/C support on ART CICS are listed as below.

- The EXEC CICS commands related to nonstructured exception handling are not supported, including:
  - HANDLE ABEND
  - HANDLE AID
  - HANDLE CONDITION
  - IGNORE CONDITION
  - PUSH HANDLE
  - POP HANDLE
- In a C application, every EXEC CICS command is treated as if it had NOHANDLE or RESP option specified.

- On Linux, file names and words are case-sensitive. If a header file name is in lowercase in C source file while such name in file system is in uppercase, errors will occur in compiling.
- Does not support packed decimal data.
- Does not support the use of CICS command in macros.
- Support CICS keywords in mixed case.
- Where CICS expects a fixed-length character string (such as a program name, map name, or queue name), you must pad the literal with blanks up to the required length if it is shorter than expected.
- Take care not to define a variable with a field name. That behavior will cause C compiler abend.
- `/**/` is used for single line comments. Do not put a comment in the middle of an EXEC CICS command.
- ART CICS does not support `argc`, `argv`, and `envp`.
- ART CICS provides two methods to access `COMMAREA`:
  - ADDRESS `COMMAREA`
  - Provide an extern global pointer `__commptr` declared by ART CICS pre-processor automatically. `__commptr` is system reserved; users cannot define it as other usages.
- ART CICS provides two methods to access `EIB`:
  - ADDRESS `EIB`
  - Provide an extern global pointer `__eibptr` declared by ART CICS pre-processor automatically. `__eibptr` is system reserved; users cannot define it as other usage.
- The `EIB` declarations are enclosed in `#ifndef` and `#endif` lines, but are *not* included in all translated files. ART CICS publishes header file `dfheibblk.h` to contain the definition of all the fields in the `EIB`. Each translated file just needs to include this header file and all actions are completed by pre-processor automatically.
- BMS screen attributes definitions: C versions of the `DFHBMSCA` and `DFHAID` files are supplied by CICS, and may be included by the application programmer when using BMS.
- The string handling functions in the C standard library use a null character as an end-of-string marker. Because CICS does not recognize a null as an end-of-string marker,

customers must take care of it when using C functions, for example `strcmp`, to operate on CICS data areas.

- ART CICS provides `iscics()` declared in `cics.h` as well, but users only need to modify `makefile` to include the header file path.
- Keep `EXEC CICS` as a whole in one line.
- Multiple CICS commands in one line is not support.
- `#pragma` will be automatically translated to comments.
- C function `exit()` will be translated to `return`.
- Keep C function `main()`, its parameter list, and parenthesis in one line. For example,  

```
void main(int argc, char **argv)
```
- `COMMAREA` should be a pointer. ART CICS only supports specifying a struct by pointer, not value.

## Accessing EIB from C

The address of the EXEC interface block (EIB) is not passed as an argument to a C main function; however, users can use the following two methods to obtain the address of the EIB:

- Using `ADDRESS EIB`
- Using global pointer `__eibptr` which points to EIB

## Accessing COMMAREA from C

The address of `COMMAREA` is not passed as an argument to a C main function; however, users can use the following two methods to obtain the address of the `COMMAREA`:

- Using `ADDRESS COMMAREA`
- Using global pointer `__commptr` which points to `COMMAREA`

## CICS Command Translator

ART CICS provides `prepro-cics-C.pl` for CICS/COBOL APIs translation. For more information about `prepro-cics-C.pl`, please refer to [prepro-cics-C.pl](#).

## C Program Compilation

In order to make sure the C programs could be successfully loaded by COBOL runtime, please build C programs using COBOL compiler rather than gcc/g++.

Use `cob` to compile C source code as a callable shared object. Please note that dynamic library must have the same name as the C source file name in uppercase.

For example,

```
CPYINC=../includes
```

```
cob -z,CC zample_treated.c -o ZAMPLE.so -CC -I${CPYINC}
```

## Implementing CICS Applications



# Reference

## Cross Reference of .desc Configuration Files Used by CICS Runtime Servers

The following table lists the configuration .desc files used per each CICS Runtime server. The value of 1 at the intersection of a server and a file means that they are linked.

**Table 5-1 Resources Configuration ".desc" File**

Servers	List of Group	PROG RAMS	TRANCL ASSES	TRANSA CTIONS	TSQM ODEL	ENQUE MODEL	TDQ INTRA	TDQ EXTRA	TYPET ERM	MAP SET	SYS TEM	TERM INALS	CON NEC TION	WEB SER VICE	ROGR AMS LIST	POOL	Liste ner	Total
ARTTCPL /H									1									1
ARTCNX													1					1
ARTATRI	1	1	1	1	1	1	1	1			1		1	1		1		12
ARTATRN	1	1	1	1	1	1	1	1			1		1	1		1		12
ARTSTR1	1	1	1	1	1	1	1	1		1	1		1	1		1		13
ARTSTRN	1	1	1	1	1	1	1	1		1	1		1	1		1		13
ARTCTR1	1	1	1	1	1	1	1	1		1	1		1			1		12
ARTCTRN	1	1	1	1	1	1	1	1		1	1		1			1		12
ARTWTR1	1	1	1	1	1	1	1	1		1	1		1	1		1		13
ARTWTRN	1	1	1	1	1	1	1	1		1	1		1	1		1		13

## Reference

**Table 5-1 Resources Configuration ".desc" File**

Servers	List of Group	PROG RAMS	TRANCL ASSES	TRANSA CTIONS	TSQM ODEL	ENQUE MODEL	TDQ INTRA	TDQ EXTRA	TYPET ERM	MAP SET	SYS TEM	TERM INALS	CON NEC TION	WEB SER VICE	ROGR AMS LIST	POOL	Liste ner	Total
ARTDPL	1	1		1	1	1	1	1			1		1	1	1	1		12
ARTTSQ	1				1													2
ARTTSQP	1				1											1		3
ARTTDQ	1						1	1										3
ARTADM																		0
ARTCKTI																		0
ARTSRM	1	1		1					1		1	1						6
ARTCSKL																	1	1
<b>Total</b>	<b>13</b>	<b>10</b>	<b>8</b>	<b>10</b>	<b>11</b>	<b>9</b>	<b>10</b>	<b>10</b>	<b>2</b>	<b>6</b>	<b>10</b>	<b>2</b>	<b>9</b>	<b>7</b>	<b>1</b>	<b>10</b>	<b>1</b>	<b>129</b>

# Oracle Tuxedo Application Runtime for CICS CSD Converter

This chapter contains the following topics:

- [Overview](#)
- [Resource Definition Online \(RDO\) Mapping](#)

## Overview

The administration of CICS Runtime is based on Oracle Tuxedo native tools with the addition of a limited number of configuration tables for features that are specific to CICS. In CICS configurations, resources are currently defined in the CICS system definition file (CSD).

The `tcxcsdcvr` tool (located in the `$KIXDIR/tools` directory), maps the CSD file to resource descriptive files (including transaction, transaction class, program, files, TS Queue, ENQ, TD Queue extra partition, TD Queue intra partition, mapset, and typeterm).

This tool is used to set the target CSD file in argument, and the translated resource configuration files resides in current directory by default. You can also specify other target directories to store the configuration files.

## Resource Definition Online (RDO) Mapping

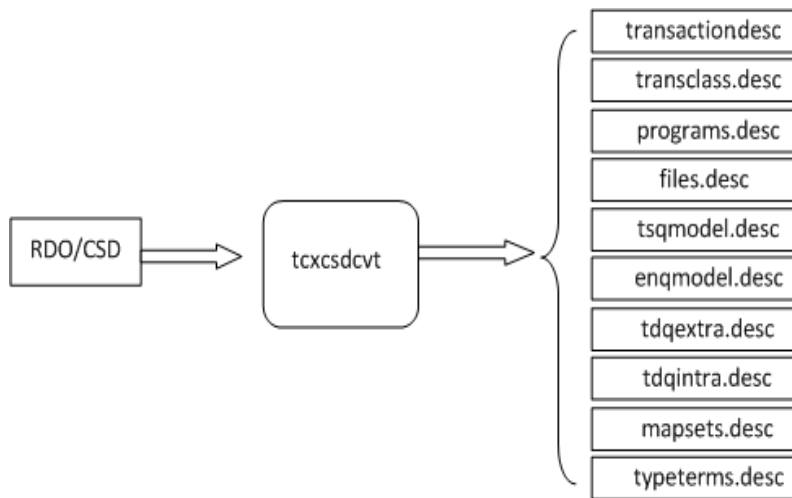
Resource Definition Online (RDO) Mapping consists of two parts:

1. Files conversion from RDO/CSD on z/OS to resource configuration files of all types on universal platform, such as transactions, programs, mapsets and etc.

2. For each type, tool `tcxcscvrt` reads the value of all fields, and then generates a record in the corresponding resource configuration file. For more information, see "[CICS Runtime Configuration Files](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

Figure 6-1 depicts the file data stream model.

**Figure 6-1 File Data Stream Model**



Tables 1~10 describe detailed correspondence between RDO/CSD and target resource configuration files, which have ".desc" as suffix mentioned above. These mappings include:

- [TRANCLASS Mapping](#)
- [PROGRAM Mapping](#)
- [FILE Mapping](#)
- [Journaling Attributes in FILE Mapping](#)
- [TSQUEUE MODEL Mapping](#)
- [ENQMODEL Mapping](#)
- [TDQUEUE Extra Partition Mapping](#)
- [TDQUEUE Intra Partition Mapping](#)
- [MAPSET Mapping](#)

- [TYPETERM Mapping](#)

**Note:** Since some field names are new options added in CICS Runtime Configuration Files, they are not defined or supported by RDO/CSD. To mark these attributes "----" is used.

**Table 6-1 TRANCLASS Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
TRANCLASS	TRANCLASS	Name of the transaction class.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual comment zone for description of the resource.
MAXACTIVE	MAXACTIVE	Defines the degree of parallelism of execution.

**Table 6-2 PROGRAM Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
PROGRAM	PROGRAM	Name of the program.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual comment zone for description of the resource.
LANGUAGE	LANGUAGE	The language of the program, required to know how to communicate with it.
EXECKEY	EXECKEY	Reserved for future use. Concerns memory protection of CICS shared structures.
STATUS	STATUS	Specifies the program status.

**Table 6-2 PROGRAM Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
REMOTESYSTEM	REMOTESYSTEM	Specifies that the program is not offered locally but in a DPL server.
REMOTENAME	REMOTENAME	Specifies for a DPL program the name of the program on the distant site.

**Table 6-3 FILE Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
FILE	FILE	Name of the file; logical name of the file used in EXEC CICS related to this file.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual comment zone for description of the resource.
DISPOSITION	DISPOSITION	Specifies the disposition of this file.
DSNAME	DSNAME	Specifies the data set name to be used for this file.
JOURNAL	JOURNAL	Specifies whether you want automatic journaling for this file.
KEYLENGTH	KEYLENGTH	Specifies the length in bytes of the logical key of records in remote files, and in coupling facility data tables that are specified with LOAD (NO).
OPENTIME	OPENTIME	Specifies when the file is opened.
READINTEG	READINTEG	Specifies the level of read integrity required for files defined with RLSACCESS (YES).

**Table 6-3 FILE Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
RECORDSIZE	RECORDSIZE	Specifies the maximum length in bytes of records in a remote file or a coupling facility data table.
REMOTENAME	REMOTENAME	Specifies the name of the file on the remote system.
REMOTESYSTEM	REMOTESYSTEM	On source, specifies the name of the connection that links the local system to the remote system where the file resides.  On the target platform, will be used only in case of file shipping to another system, either another TUXEDO system or native CICS system.
STATUS	STATUS	Specifies the initial status of the file following a CICS initialization.

**Table 6-4 Journaling Attributes in FILE Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
JNLADD	JNLADD	Specifies if you want the add operations recorded on the journal nominated by the JOURNAL attribute.
JNLREAD	JNLREAD	Specifies the read operations you want recorded on the journal nominated by the JOURNAL attribute.
JNLSYNCREAD	JNLSYNCREAD	Specifies whether you want the automatic journaling records, written for READ operations to the journal, to be synchronous.

**Table 6-4 Journaling Attributes in FILE Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
JNLSYNCWRITE	JNLSYNCWRITE	Specifies whether you want the automatic journaling records, written for WRITE operations to the journal, to be synchronous.
JNLUPDATE	JNLUPDATE	Specifies whether you want REWRITE and DELETE operations recorded on the journal.

**Table 6-5 TSQUEUE MODEL Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
TSMODEL	TSMODEL	Name of the TS Queue model.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual zone for description of the resource.
LOCATION	LOCATION	Specifies the kind of storage to use: file or memory.
PREFIX XPREFIX	PREFIX XPREFIX	Specifies the character string that is to be used as the prefix for this model.
RECOVERY	RECOVERY	Specifies whether or not queues matching this model are to be recoverable.
POOLNAME	POOLNAME	Specifies the 8-character name of the shared TS pool definition that you want to use with this TSMODEL definition.



**Table 6-5 TSQUEUE MODEL Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
REMOTE_SYSTEM	REMOTESYSTEM	On source platform, specifies the name of the connection that links the local system to the remote system where the temporary storage queue resides.  On the target platform, used only in case of TS shipping to another system, either another TUXEDO system or native CICS system.
REMOTEPREFIX XREMOTEPREFIX	REMOTEPREFIX XREMOTEPREFIX	Specifies the character string that is to be used as the prefix on the remote system. The prefix may be up to 16 characters in length
SECURITY	SECURITY	Specifies whether security checking is to be performed for queues matching this model.

**Table 6-6 ENQMODEL Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
ENQMODEL	ENQMODEL	Name of the ENQ model.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual zone for description of the resource.
ENQNAME	ENQNAME	Specifies the 1 to 255-character resource name.

**Table 6-6 ENQMODEL Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
ENQSCOPE	ENQSCOPE	If omitted or specified as blanks, matching enqueue models will have a local scope, else they will have a global scope
STATUS	STATUS	E = Enabled; D = Disabled.

**Table 6-7 TDQUEUE Extra Partition Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
TDQUEUE	TDQUEUE	Specifies the 1- to 4-character name of a transient data queue.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual zone for description of the resource.
DDNAME	DDNAME	Specifies a 1-to 8-character value that may refer to a data set defined in the startup JCL.
DISPOSITION	DISPOSITION	Specifies the disposition of the data set (MOD; OLD; SHR).
ERRORPTION	ERRORPTION	(UNSUPPORTED) Specifies the action to be taken if an I/O error occurs.
OPENTIME	OPENTIME	(UNSUPPORTED) Specifies the initial status of the data set.
RECORDFORMAT	RECORDFORMAT	Specifies the record format of the data set.
PRINTCONTROL	PRINTCONTROL	(UNSUPPORTED) Specifies the control characters to be used.

**Table 6-7 TDQUEUE Extra Partition Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
RECORDSIZE	RECORDSIZE	Specifies the record length in bytes.
TYPEFILE	TYPEFILE	Specifies the type of data set the queue is to be associated with an input or output dataset.
DSNAME	DSNAME	Specifies the name of the file that is to be used to store records written to this extra partition queue.
SYSOUTCLASS	SYSOUTCLASS	(UNSUPPORTED) Specify the class of the SYSOUT data set.
TRT	----	Allow integrators and customers to make their own specific implementation of extra-partition queues.

**Table 6-8 TDQUEUE Intra Partition Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
TDQUEUE	TDQUEUE	Specifies the 1- to 4-character name of a transient data queue.
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual zone for description of the resource.
RECOVSTATUS	RECOVSTATUS	Specifies if the queue is logically recoverable or not.
TRANSID	TRANSID	Specifies the name of the transaction that is to be automatically initiated when the trigger level is reached.

**Table 6-8 TDQUEUE Intra Partition Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
TRIGGERLEVEL	TRIGGERLEVEL	Specifies the number of records to be accumulated before a task is automatically initiated to process them.
USERID	USERID	Specifies the userid you want CICS to use for security checking when verifying the trigger-level transaction specified in the TRANSID field.
WAIT	WAIT	(INACTIVE field) Accepted only in the resource loading.
WAITACTION	WAITACTION	(INACTIVE field) Accepted only in the resource loading.
QSPACE	----	Specify the name of the tuxedo /Q QSPACE into which this queue is physically stored.
TRT	----	Allow integrators and customers to make their own specific implementation of intra-partition queues.

**Table 6-9 MAPSET Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
NAME	MAPSET	Name of the MAPSET.
GROUP	GROUP	Installation group name.
DESCRIPTION	DESCRIPTION	A general description of the MAPSET resource.
RESIDENT	RESIDENT	YES=preload. NO=load on first use.
swastatus	STATUS	Sets the status of the resource, to specify if it is available.

**Table 6-9 MAPSET Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
Usage	USAGE	Specifies the caching scheme to be used once the MAPSET is loaded.
FILENAME	----	Specifies the physical (binary) file name of the mapset, which is generated by the <code>tcxmapgen</code> utility (refer to section).

**Table 6-10 TYPETERM Mapping**

Field Name in ART FOR CICS	Resource Attribute in RDO/CSD	Description
NAME	TYPETERM	Name of the <code>typeterm</code> .
GROUP	GROUP	The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.
DESCRIPTION	DESCRIPTION	A small textual zone for description of the resource.
color	COLOR	Designates extended color attributes.
defscreencolumn	DEFSCREEN (rows, columns)	Number of columns of the default screen size.
defscreenrow	DEFSCREEN (rows, columns)	Number of rows of the default screen size.
hilight	HILIGHT	Indicates whether a terminal supports the highlight feature.
logonmsg	LOGONMSG	Indicates whether the "Good Morning" (CSGM) transaction is automatically started on the terminal.
outline	OUTLINE	Indicates whether the terminal supports field outlining.
swastatus	STATUS	Specifies the resource status (whether available).

**Table 6-10 TYPETERM Mapping**

<b>Field Name in ART FOR CICS</b>	<b>Resource Attribute in RDO/CSD</b>	<b>Description</b>
uctran	UCTRAN	Specify whether translate lowercase alphabetic characters to uppercase, or only translate the transaction ID from lowercase to uppercase, or not translate any characters.
userarealen	USERAREALEN	The terminal control table user area (TCTUA) area size for the terminal.
INTERCODE	----	Specifies which encoding type of inbound data is used.
EXTERCODE	----	Specifies which encoding type of outbound data is used.
SOSI	SOSI	Specifies whether mixed EBCDIC and double-byte character set (DBCS) is supported.
PROGSYMBOLS	PROGSYMBOLS	Specifies whether the programmed symbol (PS) facility is supported.

# ECI Client Support

This chapter contains the following topics:

- [Overview](#)
- [Platform](#)
- [Installation and Setup](#)
- [Encoding and Decoding](#)
- [Security](#)
- [Failover](#)
- [Diagnostic](#)
- [Limitation and Compatibility](#)

## Overview

### Purpose

The external interfaces allow non-CICS applications to access and update CICS resources by calling CICS programs. When communicating with CICS, the external interfaces enable non-CICS programs to access and update resources on any CICS system. This method of using the external interfaces supports such activities as the development of graphical user interface

(GUI) front ends for CICS applications and allows the integration of CICS systems and non-CICS systems.

Oracle Tuxedo Application Runtime for CICS and Batch provides the target environment for migrating mainframe applications from IBM z/OS to an open systems application grid running Oracle Tuxedo.

When customers choose to use Oracle Tuxedo Application Runtime for CICS, ECI emulator enables customers to keep using their existed program without code change.

## Introduction

CICS Transaction Gateway and CICS Universal Client provide ECI programming interface for C, C++, COBOL, COM, .NET, and JAVA. Regarding that the implementation of programming interface is done in dynamic load library (DLL), the basic idea of ECI emulator is to replace the library and so users can keep using their application as if nothing changed; in DLL, ECI request is routed to Tuxedo and leverages all the benefits provided by Tuxedo.

This figure demonstrates how ECI Emulator works. In this document, we focus on ECI C API.



**Note:** For ECI C API, only ECI v1 API `CICS_ExternalCall (ECI_Parms)` is supported.

For more information, please refer to [Supported ECI C API Parameters](#) in [CICS Commands and Parameters Coverage](#).

## Platform

The supported platforms are listed in the [Supported Platforms](#).



# Installation and Setup

## Installation

The delivery of ECI Emulator is a ZIP package, including one `cclapi32.dll` file and other Tuxedo dependency files. To use ECI Emulator, users need to install CICS Transaction Gateway or CICS Universal Client and unzip the package under the directory, where the original `cclapi32.dll` is located in, to replace the original DLL with the delivery.

## ECI Connection to ART CICS

In typical case, user application runs on Windows while ART CICS runs on Linux. To build connection between ECI client and ART CICS, Tuxedo /WS is used.

Users need to set environment variables before calling ECI client application. For example:

```
set TUXDIR=<where the delivered package will be unzip to>
```

```
set WSNADDR=//<machine>:<port>
```

## Configuration on ART CICS

CICS programs that are invoked by an ECI request must be configured as DPL programs on ART CICS Runtime as the following.

- Configure the program in `programs.desc`, and specify the system name in `REMOTESYSTEM` field.
- Configure ARTDPL server in `ubbconfig`, and specify the system name using `ARTDPL -s` option.

In ECI call `CICS_ExternalCall (ECI_Parms)`, specify the program name using `eci_program_name` parameter, and the system name using `eci_system_name` parameter.

**Note:** For more information about ART CICS DPL configuration, please refer to [Implementing Distributed Program Link \(DPL\)](#).

## Encoding and Decoding

If the code page of the user application is different from that of the server, data conversion in a `COMMAREA` must be performed, and then two cases, legacy user application and new created user application, are required to handle.

For legacy application, the data in COMMAREA could be converted to EBCDIC before such legacy application connects server; however, regarding that the data flow on ART CICS is in ASCII format, ECI emulator needs to both convert COMMAREA from EBCDIC to ASCII before ECI request is routed to ART CICS and convert returned information back to EBCDIC.

For new created application, users could use ASCII directly without any conversion operation needed. To do this, we introduce an environment variable `CTG_CLIENT_CHARSET`. If its value is set to "EBCDIC", ECI Emulator is to perform data conversion for COMMAREA between EBCDIC and ASCII; by contrast, if this environment is not set or its value is not "EBCDIC", no conversion will be done.

## Security

In ECI parameter block fields, the supplied user ID and password are used in subsequent security checking in the server. Usually, `eci_userid` and `eci_password` are used, but they are 8 character fields. Therefore, if a user ID or password more than 8 characters is required, users should set `eci_userid` and `eci_password` to nulls, and use fields `eci_userid2` and `eci_password2` instead.

Regarding that the security mechanism of ART CICS is different from the security mechanism in ECI request, a problem occurs if security is enabled on ART CICS requiring user ID, password, and application password while there is no application password defined in ECI parameter. To solve this problem, ECI Emulator offers two alternatives for users.

1. Set application password as empty when enabling security in ART CICS side. ECI Emulator can read user ID and password from ECI parameter and then feed application password with an empty string when doing `tpinit()`.
2. Specify application password in an environment variable `CTG_APP_PWD`. If it is defined, the content in this variable will be used as application password.

If security is not enabled in ART CICS, security checking will not be performed even user ID and password are supplied in ECI parameter.

## Failover

Tuxedo / WS allows to configure alternative network address connecting to /WS remote clients. This feature is used in ECI Emulator to implement failover. More precisely, the feature enables users to configure multiple WSL servers in `UBBCONFIG` and set environment variable `WSNADDR` with all alternative network address on client side.

For example, users can set environment variables as “set WSNADDR=//bjlinux16:46249, //bjlinux16:46246” when first address is not available; then the next address will be picked up automatically.

## Diagnostic

This emulator provides a mechanism for users to diagnose the problems they met. To enable emulator log, users can set environment variables `CTG_CLIENT_TRACE_FILE` to specify log file name and `CTG_CLIENT_TRACE_LEVEL` to set log level.

The log level is ranged from 0 to 4 inclusive: 0 (no log), 1 (error), 2 (warning), 3 (info), and 4 (debug).

If environment variable `CTG_CLIENT_TRACE_FILE` is not set, `userlog` will be used.

If environment variable `CTG_CLIENT_TRACE_LEVEL` is not set, default log level will be set to error, meaning only error log will be printed. If `CTG_CLIENT_TRACE_LEVEL` is set with invalid log level, such as a negative number, debug level will be used.

## Limitation and Compatibility

### Limitation

Only supports encoding/decoding on the whole `eci_commarea` between ASCII and EBCDIC; MBCS is not supported.

### Compatibility

- Most ECI return code does not have matched error code in Tuxedo. If ECI call is failed and there is no matched Tuxedo error code, this emulator will only return a general error code `ECI_ERR_SYSTEM_ERROR` and users can view detailed error information and diagnose problem by enabling log output. Supported ECI defined return codes are listed below.
  - `ECI_NO_ERROR`
  - `ECI_ERR_SYSTEM_ERROR`
  - `ECI_ERR_INVALID_EXTEND_MODE`
  - `ECI_ERR_INVALID_CALL_TYPE`
- ECI emulator supports both customer legacy ECI program and new created ECI client.

ECl Client Support

# IMS DB Access Support

This chapter contains the following topics:

- [Overview](#)
- [Configurations](#)
- [Supported Platforms](#)
- [Tips](#)

## Overview

ART for CICS enables you to use `DL/I CALL 'CBLTDLI'` to access IMS DB through Oracle ODBA. With this feature, ART for CICS application programs can do operations in IMS DB, such as adding, searching, and deleting data.

## Configurations

Configure the followings for the IMS DB Access Support.

- [Configure ART for CICS for Accessing IMS DB](#)
- [Configuring ART for CICS Servers](#)
- [Configuring Environment Variables](#)
- [Configuring IMS](#)

## Configure ART for CICS for Accessing IMS DB

ART for CICS servers provide you IMS DB access environment by dynamically loading DL/I library (`libcicsdli.so` and `libcicsdlib.so`), which works as a plug-in component in ART for CICS.

Therefore, you should get these two libraries from Oracle Tuxedo Application Runtime for IMS and locate them in ART for CICS environment variable `LD_LIBRARY_PATH`.

- On Linux platforms, you can get DL/I library from Oracle Tuxedo Application Runtime for IMS Rolling Patch 025 or later.
- On AIX platforms, you can get DL/I library from Oracle Tuxedo Application Runtime for IMS Rolling Patch 026 or later.

The DL/I library is located in

- `art12.1.3.0.0/IMS_RT/lib/libcicsdli.so`
- `art12.1.3.0.0/IMS_RT/lib/libcicsdlib.so`

For more information about `LD_LIBRARY_PATH`, see "[Environment Variables](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Configuring ART for CICS Servers

It is required to configure `--IMSDB` argument in ART for CICS Server CLOPT in `UBBCONFIG` file. For example,

```
--IMSDB -x -o wasa.us.oracle.com:6799:IMSDB
```

**Note:** `--IMSDB` argument must be the last argument in CLOPT.

For more information, see "[IMS DB Argument](#)" in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Configuring Environment Variables

It is required to set the following environment variables.

- `ART_IMS_CONFIG`
- `ART_IMS_DB`
- `KIX_IMSDB_TRACE_LEVEL`

- LD\_LIBRARY\_PATH

For more information, see ["Environment Variables"](#) in *Oracle Tuxedo Application Runtime for CICS Reference Guide*.

## Configuring IMS

ART for CICS uses the following configuration files for accessing IMS DB:

- Application Definition - imsapps.desc
- Database Definition - imsdbs.desc
- PSB Definition - \$appname.psb
- Segments Definition - segments.desc
- Segment Definition - \$segname.desc

For more information, see ["Configuration Files"](#) section in *Oracle Tuxedo Application Runtime for IMS Reference Guide*.

## Supported Platforms

This feature supports all *Linux/AIX 64-bits platforms* that Oracle Tuxedo Application Runtime for CICS and Batch 12c Release 2 (12.1.3) supports, which are listed in the ["Supported Platforms"](#) in *Oracle Tuxedo Application Runtime for CICS and Batch Installation Guide*.

## Tips

When using COBOL-IT COBOL, you should use option `-falloc-unused-linkage` to compile programs which include `DLIUIB` in linkage section.

## IMS DB Access Support



# UDB Linking

## Installation Time UDB Linking

The CICS Server Build Tool `buildartcics` is provided to help you generate CICS Runtime servers. CICS Runtime servers can be linked with an Oracle database or a UDB (LUW) database.

### Rebuilding Servers for UDB

The servers delivered are built to be used with Oracle; to rebuild a server for UDB, you can run `buildartcics` with `UDB_XA` as the RM. For example:

```
buildartcics -r UDB_XA -o ARTSTRN_UDB
```

For detailed information of this tool, please see CICS Runtime Server Build Tool in [Reference Guide](#).

Besides, the file `makefile_sample` is provided as an example to help you generate multiple CICS Runtime servers at a time:

1. Open the `<ART_INSTALL_DIR>/Cics_RT/tools` directory;
2. Set correct environment variables as required, such as “`DB2DIR`”, “`TUXDIR`”, “`KIXDIR`”, and “`COBDIR/COBOLITDIR`”. For more information of each environment variable, see [CICS Runtime Server Build Tool](#) in Reference Guide;
3. Make sure all the servers in “`ALL_EXECUTABLES`” target in `makefile_smaple` are exactly the targeted servers (`*_UDB`) you need;
4. Run “`gmake -f makefile_sample all`”.

## UDB Linking

**Note:** For UDB linking, make sure that you have the following line in the Tuxedo RM file:

```
UDB_XA:db2xa_switch_std:-L${DB2DIR}/lib64 -ldb2 -ldb2gmf
```

# Rebuilding ART Servers for CICS

You need to rebuild the ART CICS servers in case of updates on one of the following components:

- Major OS version
- Tuxedo
- RDBMS: Oracle or UDB
- WebSphere MQ
- COBOL compiler: Micro Focus COBOL or COBOL-IT
- C++ compiler

## Rebuilding the ART CICS Servers

To rebuild the a server with Oracle as the RM, you can simply use the CICS Server Build Tool `buildartcics`. For example:

```
buildartcics -r Oracle_XA -o ARTSTRN
```

For detailed information of this tool, please see [CICS Runtime Server Build Tool](#) in Reference Guide.

Besides, the file `makefile_sample` is provided as an example to help you generate multiple CICS Runtime servers at a time:

1. Open the `<ART_INSTALL_DIR>/Cics_RT/tools` directory;

## Rebuilding ART Servers for CICS

2. Set correct environment variables as required, such as “ORACLE\_HOME”, “TUXDIR”, “KIXDIR”, and “COBDIR/COBOLITDIR”. For more information of each environment variable, see [CICS Runtime Server Build Tool](#) in Reference Guide;
3. Make sure all the servers in "ALL\_EXECUTABLES" target in `makefile_sample` are exactly the targeted servers you need;
4. Run “`gmake - f makefile_sample all`”.

# External CICS Interface (EXCI)

## Overview

The external CICS interface is an application programming interface which enables a non-CICS program running in MVS to:

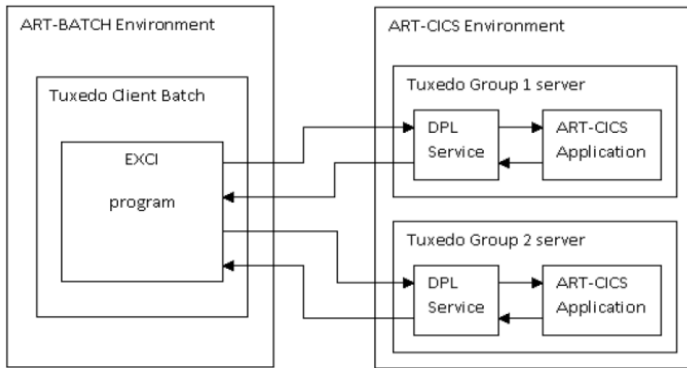
- Allocate and open sessions to a CICS system, and issue DPL requests on these sessions.
- Call a server program running in a CICS environment, pass and receive data by means of a communications area.

The external CICS interface provides two forms of programming interface:

- The `EXCI CALL` interface, which consists of six commands that allow you to:
  - Allocate and open sessions to a CICS system from non-CICS programs running under the MVS.
  - Issue DPL requests on these sessions from the non-CICS programs.
  - Close and release the sessions on completion of the DPL requests.
- The `EXCI EXEC CICS` interface, which provides a single composite command that performs all six commands of the `EXCI CALL` interface in one invocation.

# EXCI in Oracle Tuxedo Application Runtime

Figure C-1 EXCI in ART



Each EXCI ART CICS program must be defined as a DPL service in the `resource/program.desc` file. The seventh column must contain the CICS SYSID, and the service is advertised with the name: `<SYSID>_<PROGRAM>`.

If a mirror transaction is defined in `transaction.desc` using `DFHMIRS`, DPL advertises a service with name `<APPLID>_MIRROR_<TRANSID>` or `MIRROR_<TRANSID>`.

If all DPL requests are done with `SYNCONRETURN` and not under the RRMS control, no Resource Manager is required in the Tuxedo Client. The initialization is done during the first EXCI request process and the Tuxedo session is terminated at the end of the client process.

If RRMS is used or one DPL request is done without `SYNCONRETURN`, the Tuxedo Client process must be built with one Resource Manager. At the initialization, the Resource Manager is opened and the transaction is begun at the beginning of the Client process. If one of these steps is not successful, the Client process aborts. At the normal end of the client process, the transaction is committed if it was not explicitly done by the client program (`RSSCMIT`). At the abnormal end of the client process, the transaction is rolled back. In each of these cases the Resource Manager is closed and the Tuxedo session is terminated.

## Supported EXCI Interface

The EXCI precompiler option must be used for CICS client batch program.

The COBOL precompiler supports EXCI CALL or only one CICS command, EXEC CICS LINK with the next described options. The C precompiler only supports EXCI EXEC CICS LINK.

ART CICS supports DFHXCIS API for making EXCI requests (DFHXCIS is a procedure API that client programs make EXCI CALL).

In case of EXEC CICS LINK, the RETCODE command option is mandatory with EXCI but forbidden with NOEXCI, and the APPLID option is EXCI specific. Without EXCI the SYSID option can be used.

With the EXCI precompiler option neither DFHEIBLK nor DFHCOMMAREA is generated as PROCEDURE DIVISION USING parameter.

The EXCI precompiler option is set by inserting a COBOL comment line containing from the seventh column:

```
*KIX--OPTION EXCI
```

before IDENTIFICATION DIVISION line.

The EXCI C precompiler option is set by "-B".

ART CICS supports the following EXCI EXEC CICS LINK commands: PROGRAM (name), APPLID (name), and TRANSID (name). DATALENGTH (data-value) is recognized.

## Precompiler Controls

- PROGRAM () and RETCODE () are required for the LINK command in EXCI.
- SYSID is not recognized in EXCI.
- COMMAREA must be present if LENTGH or DATALENGTH is present.
- EXCI CICS LINK is the only supported command in EXCI.

## Access Authorization

The Tuxedo configuration SECURITY level drives the access authorization.

The MT\_EXCIAPPPROFILE environment variable provides the application profile file name generated by the genappprofile ARTKIX tool. The default file name is \$HOME/.tuxAppProfile.

In DPL program which is issued by EXCI client, the EXEC CICS ASSIGN USERID() command returns:

The value of `$USER` environment variable when there is no security level set inside the Tuxedo `ubbconfig`. The value of `USERID` input in `genappprofile` tool when enabling security level inside the Tuxedo `ubbconfig`, the value of `USERID` is got from `.tuxAppProfile` and passed by DPL request.

See [Oracle Tuxedo Application Runtime for Batch](#) documentation for more details.

In DPL program which is issued by EXCI client, the `EXEC CICS ASSIGN USERNAME()` command returns:

The value of `$USERNAME` environment variable no matter whether there is any security level set inside the Tuxedo `ubbconfig` or not.

## ART CICS Implementation

The programs linked via the EXCI interface are advertised by the `ARTDPL` server. They are named as `<program>_<sysid>`, where `<program>` is the linked program name (option `PROGRAM(<program>)` of `EXCI EXEC` interface), and `<sysid>` is the CICS system ID.

The EXCI interface uses the `<applid>` CICS application ID to address the appropriate CICS region. The relationship between `<applid>` and `<sysid>` is made via a specific DPL server service named `<applid>_info`.

The `-a` user parameter value of the DPL server command line (`CLOPT`) is used as `<applid>` value for the `_info` service.

If the `<applid>` is omitted by the client (without `APPLID(<applid>)` `EXCI EXEC` interface option), the `default_info` service is called. This service is advertised by the first DPL booted server.

The `_info` service returns the `<sysid>` associated to the server by the `-s` user parameter of the server command line.

Since ART CICS rolling patch 015, four new services are advertised by the `ARTDPL` server:

- `<applid>_CSMI`, where `<applid>` is the CICS application ID to address the appropriate CICS region. This service is called if `<transid>` is not specified but `<applid>` is specified in EXCI interface by the client.
- `CSMI`. This service is called if both `<applid>` and `<transid>` are not specified in EXCI interface by the client.
- `<applid>_MIRROR_<transid>`, where `<applid>` is the CICS application ID to address the appropriate CICS region, and `<transid>` is the transaction ID. This service is called if both `<applid>` and `<transid>` are specified in EXCI interface by the client.



- MIRROR\_<transid>, where <transid> is the transaction ID. This service is called if <applid> is not specified but <transid> is specified in EXCI interface by the client.

## ART Restrictions

### Common EXCI Interfaces ART Restrictions

- The TRANSID has no meaning. There is no control on it. It is only passed to the DPL service in the EIBTRNID field in DFHEIBLK structure.
- The COMMAREA length is limited to 32763 bytes.

### EXCI CALL Interface ART Restrictions

- Only VERSION-1 is supported.
- The initial user USER-NAME is only used to generate a user-token without any control.
- The DPL UOWID is kept for compatibility only, and is not set and tested.
- The PIPE-TYPE has no meaning. The recognized values for PIPE-TYPE are only X'00' and (X'C3' or X'D8) (X'C3' and X'D8' are the possible ASCII values for X'80' EBCDIC depending code-page). On other value the response code is set to 12 and the reason code to 498.
- The recognized values for SYNC-TYPE are only X'00' and (X'C3' or X'D8') (X'C3' and X'D8' are the possible ASCII values for X'80' EBCDIC depending code-page). On other value the response code is set to 12 and the reason code to 499.

### EXCI EXEC Interface ART Restrictions

The DFHXCRM replaceable-module is not treated.

## SRRCMIT/SRRBACK Functions

The SRRCMIT and SRRBACK functions are available. ATRCMIT and ATRBACK functions are not supported.

SRRCMIT and SRRBACK functions must be coded as:

```
01  SRR-RETCODE                PIC 9(8) COMP-5.
CALL "SRRCMIT" USING SRR-RETCODE
CALL "SRRBACK" USING SRR-RETCODE
```

## Configuration Files Declaration for EXCI EXEC CICS LINK

To use EXCI EXEC CICS Link command, the `system.desc`, `transactions.desc`, and `program.desc` configuration files should be configured.

Following are configuration examples.

### Listing C-1 `system.desc`

---

```
[SYSID]
APPLID={applid}
```

---

### Listing C-2 `transactions.desc`

---

```
#TRANSMNAME;GROUPNAME; DESCRIPTION; PROGRAM,hardcode with DFHMIRS
ECCI;SIMPDPDPL;pg for simpapp;DFHMIRS
```

---

### Listing C-3 `program.desc`

---

```
#PROGRAM;GROUP;DESCRIPTION;LANGUAGE;EXECKEY;STATUS
TOUPSVR;SIMPDPDPL;pg for simpapp;COBOL; ;ENABLED;KIXD
```

---

# COBOL Program Debugging and Error Processing in CICS Runtime

This chapter contains the following topics:

- [Debugging COBOL Programs in CICS Runtime](#)
- [Error Processing in CICS Runtime](#)

## Debugging COBOL Programs in CICS Runtime

ART for CICS enables you to debug COBOL application programs online without modifying the program. The supported COBOL compilers are Micro Focus COBOL and COBOL-IT COBOL; you can use Animator tool to debug Micro Focus COBOL programs while use Deet tool to debug COBOL-IT COBOL programs. Whichever tool you use, the tool intercepts execution of the application program at various points before displaying information about the program. Any screens that the application program sends are displayed by the tool, so that you can converse with the application program during testing, just as you would on the production system.

ART for CICS supports cross-session debugging with Micro Focus COBOL and COBOL-IT COBOL. Cross-session debugging enables you to use the Animator tool or the Deet tool in a different terminal window from that in which the program to be debugged is running.

- [Debugging with Micro Focus COBOL](#)
- [Debugging with COBOL-IT COBOL](#)
- [Configuration](#)
- [Dynamically Load the Debug Configuration File](#)

### Notes:

- ART for CICS 12.1.3 Rolling Patch 019 or later is required.
- COBOL-IT COBOL version 3.7.43 or later is required.

## Debugging with Micro Focus COBOL

Follow these steps for debugging with Micro Focus COBOL.

- First, create `config/resources/kix_cobol_dbg.cfg` configuration file. For more information, see [Configuration](#).
- Second, use `prepro-cics.pl` utility to preprocess the COBOL program. For more information, see [Configuration](#).
- Next, restart your application by using `tmshutdown/tmboot` or following the instructions in [Dynamically Load the Debug Configuration File](#).
- Last, start Animator in one session at first and the Animator remains in waiting state until it attaches to a Micro Focus COBOL program that has been started in another session.

## Debugging with COBOL-IT COBOL

Follow these steps for debugging with COBOL-IT COBOL. For more information about Deet graphic UI, see *COBOL-IT COBOL documentation*.

- First, create `config/resources/kix_cobol_dbg.cfg` configuration file. For more information, see [Configuration](#).
- Second, use `prepro-cics.pl` utility to preprocess the COBOL program. For more information, see [Configuration](#).
- Next, restart your application by using `tmshutdown/tmboot` or following the instructions in [Dynamically Load the Debug Configuration File](#).
- Next, start your transaction. It will hang before the COBOL program to run and wait you to start debug session.
- Next, use `vncserver` to start a VNC environment.

In VNC xterm, start debug session with command `deet -p myAnimSrvID1`. It starts a Deet graphic UI and attaches the COBOL program. Note that you should start your transaction at first, and then start debug session with Deet tool.

- Last, you can debug the COBOL program step by step in Deet graphic UI.

**Note:** COBOL-IT COBOL Deet tool does not support LINK (local) inside one transaction. To debug LINK (local) cases, see [Use Case 4: One user wants to debug two programs with LINK \(remote\)](#).

## Configuration

Do the following configurations before debugging your COBOL programs in CICS Runtime.

- Configure `kix_cobol_dbg.cfg` configuration file.  
For more information, see [Debug Configuration File](#).
- Use the `prepro-cics.pl` utility to preprocess the COBOL program.

```
prepro-cics.pl -type_output=orig < RSSBT000.cbl > RSSBT000.cob
```

### Notes:

- Whenever a CICS COBOL application program runs, ART for CICS application server checks the above configurations to determine whether to enable debugging; therefore, you must complete all above configurations before debugging.
- We recommend you to delete all `.gnt` files under COBOL source code directory.
- The Linux user account that starts up the ART for CICS server must be the same as the Linux user account that runs the `anim` (for Micro Focus COBOL programs) / `deet` (for COBOL-IT COBOL programs) command line. Only the `ANIMSRVID` which the `anim` / `deet` utility specifies will be debugged.

## Dynamically Load the Debug Configuration File

You can dynamically load the debug configuration resource file `kix_cobol_dbg.cfg` without restarting the ART for CICS.

Do the following steps to dynamically load this configuration file.

1. Launch the `artadmin` utility.

For more information, see [artadmin \(1\)](#).

2. Input `config_update (cu)`.

`config_update (cu)` propagates the configuration changes and requests the application servers to take in the changes in the configuration.

3. Input `perform (p)`.

`perform (p)` performs the commands submitted to the server and clears the commands buffer.

If the buffer is not empty, the buffer container is displayed and a confirmation is required.

If the submission fails, the message "Perform cancelled" is displayed, and the error is logged into the `USERLOG`.

4. Input `quit (q)`.

Input `quit (q)` to quit this session.

## See Also

- [Implementing COBOL Program Debugging in CICS Runtime](#)

## Error Processing in CICS Runtime

CICS runtime can detect the exception of CICS verbs and then output the relevant error message and `ABEND` code; besides that, ART for CICS installs the error procedure, which is running when COBOL LE (language environment) error occurs. In this error procedure, ART for CICS can report the detailed error line and the reason why COBOL program ends abnormally, and then CICS runtime can abort the COBOL program with CICS `ABEND` code `ASRA` to avoid the CICS runtime server from dying.

## Prerequisite

No matter which COBOL compiler you use (Micro Focus COBOL or COBOL-IT), CICS runtime installs error procedure by default.

ART for CICS provides environment variable `KIX_CBL_TRAP_ERROR` to enable or disable the error procedure. Its default value is `Y`, meaning the error procedure is enabled. If `KIX_CBL_TRAP_ERROR=N` is specified, the error procedure will be disabled and the CICS runtime server dies with core file generated when COBOL LE error occurs.

Specially, if CICS runtime is running with COBOL-IT, you should also compile COBOL program with `-debug` compiler flag to enable the error procedure function.

For more information, see [KIX\\_CBL\\_TRAP\\_ERROR](#).

## Memory Dump

If CICS runtime is running with COBOL-IT, ART for CICS will not only provide error procedure function but also dump final memory information of the program when COBOL LE error occurs.

You can enable this memory dump function by specifying the environment variable `KIX_DUMP_FILE` as a valid local file name. When COBOL LE error occurs, ART for CICS

activates the error procedure at first, and then dumps the final memory information of program into the dump file which `KIX_DUMP_FILE` specifies.

The memory dump function also works when CICS verbs error occurs.

For more information, see [KIX\\_SO\\_SUBSYS\\_WRAPPER](#).

## COBOL Program Debugging and Error Processing in CICS Runtime



# Integrating Client Applications Using CPI-C

This chapter contains the following topics:

- [Overview](#)
- [Supported CPI-C Scenarios](#)
- [Server Side Configuration](#)
- [Client Side Configuration](#)
- [Oracle Tuxedo Timeout Controls](#)
- [Security](#)
- [Scaling](#)
- [Diagnostics](#)
- [Packaging/Installation](#)

## Overview

The Common Programming Interface for Communications (CPI-C) provides a common application programming interface (API) for implementing APPC. CPI-C provides a consistent set of functions for program-to-program communication across different platforms.

Customer applications running on open systems (Windows, Linux, Unix) can use the CPI-C interfaces for APPC communication with mainframe applications running under IBM CICS and IMS TM. After rehosting such mainframe applications to Tuxedo ART, the open systems applications want to preserve the CPI-C/APPC interfaces for communicating with the rehosted mainframe applications in order to avoid or minimize any change. Using Tuxedo ART CICS support for CPI-C interfaces, customer applications can continue to interface with the rehosted mainframe applications without changing any application code.

ART CICS CPI-C support covers the followings.

**Table 0-1 ART CICS CPI-C Support Coverage**

CPI-C APPLICATIONS (CLIENT)		CPI-C APPLICATIONS (SERVER)
<b>Windows VS Client</b>	<b>WebLogic JAM Client</b>	<b>CICS Applications</b>
Over Tuxedo Workstation Client (WSC)	Over WTC	ARTCTRN/1 WSL/WSH for WSC Domain Gateway for WTC
MS Visual Studio C/C++, 32-bits	JDK 1.6 or higher, 64-bits	COBOL/C, 64-bits

ART CICS CPI-C integration provides the support for Windows VS C/C++ applications over Tuxedo WSC and Java applications over WTC, the applications acts as client and use APPC protocol to interoperate with rehosted Mainframe CPI-C applications running under ART CICS server.

Tuxedo ART CICS CPI-C support includes the following components:

- For Windows, a library `kixcpiows.dll`, which provides CPI-C interfaces as a replacement for Windows CPI-C/SNA library, using Tuxedo WSC for communications instead of Windows SNA support.
- For WebLogic, a set of Java classes that implement CPI-C interfaces, replacing JAM beans and using WTC for Tuxedo Domains communications support.
- For Tuxedo ART CICS runtime, a CPI-C library that works in `ARTCTRN` servers to support COBOL/C programs using CPI-C.

See [Supported CPI-C Scenarios](#) for more information.

## Client Applications Impact

### Windows Visual Studio C/C++ Environment

Windows applications need to be rebuilt/re-linked using the ART CICS CPI-C library provided for this support. The new library supports the same APIs so no code changes are involved once it's linked in with the application.

This library uses Tuxedo Workstation Client (WSC) on Windows as its communications channel with Tuxedo. This Tuxedo component has to be installed and configured on Windows. See [Client Side Configuration](#) for more information.

## WebLogic Java Environment

WebLogic users need to regenerate the EJB package with new version of `callService()` class that provides main access point to CPI-C interfaces and java class files which provide the ART CICS implementations of CPI-C interfaces.

These classes use WebLogic-Tuxedo Connector (WTC) as its communications channel with Tuxedo. This WebLogic component has to be installed and configured on WebLogic Server. See [Client Side Configuration](#) for more information.

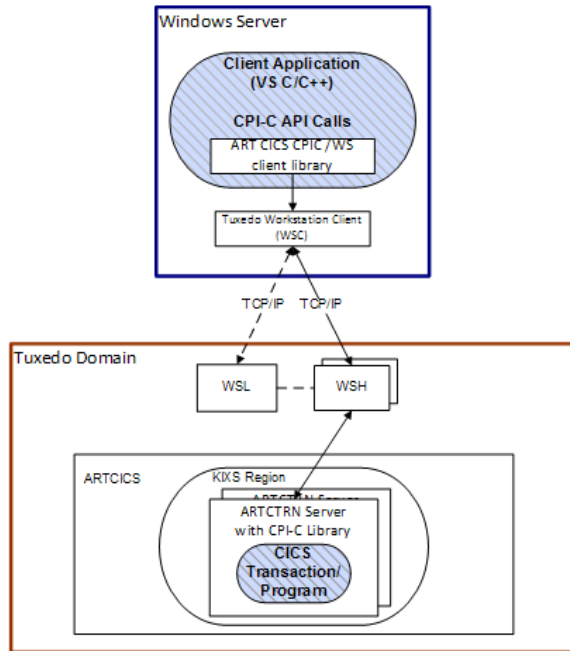
## ASCII-EBCDIC Data Conversion

Mainframe applications use EBCDIC data encoding and require Windows and WebLogic applications to specify ASCII-EBCDIC conversion in their calls. When rehosted to ART CICS, these mainframe applications run using ASCII data encoding, so no conversion is required. If Windows or WebLogic applications specify ASCII-EBCDIC data conversion, it should be disabled when working with Tuxedo ART CICS applications.

# Supported CPI-C Scenarios

## Windows Application Calling Rehosted CICS Transactions

The client application on Windows server communicates with CICS server application on Tuxedo using the CPI-C APIs in both client and server side code. Windows application with ART CICS CPI-C library uses Tuxedo workstation client, and CPI-C APIs are handled by internal Tuxedo ATMI calls by the ART CICS CPI-C libraries. `ARTCTRN` server also provides CPI-C interfaces in ART CICS runtime. The rehosted CICS application can directly use CPI-C interfaces without any code changes.

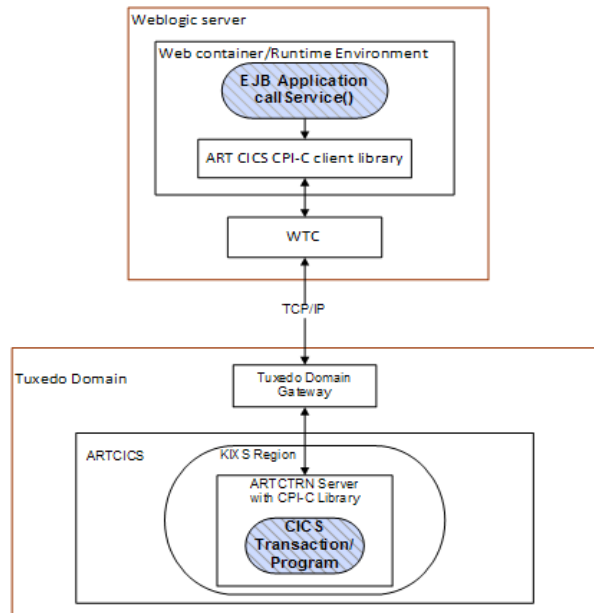


The diagram above shows CPI-C client application on Windows server which communicates with a CPI-C server CICS program in KIXS region in Tuxedo Domain through Tuxedo Workstation Protocol conversation. CPI-C client library establishes a connection with a `tpconnect()` call to a CICS transaction running in one or more ARTCTRN servers (ART CICS application server for conversational transactions/programs) and uses Tuxedo conversational `tpsend()/tprecv()` API calls to mimic `CMSSEND/CMRCV`. Application code on both sides represented by blue shaded areas remains unchanged, with Tuxedo ART infrastructure components providing all the necessary API and communications support.

## WebLogic Application Calling Rehosted CICS Transactions

The Java client application in WebLogic communicates with CICS application in server side over iWay JAM. The client side uses RMI `callService()` provided by JAM, and the server side uses CPI-C interfaces and CICS APPC interfaces. ART CICS overloads the `callService()` method in client side, `callService()` is composed of a series of CPI-C calls to exchange the application data with CICS application. ART CICS provides the source codes of an overloaded

`callService()` method, and provides a series of Java class files to provide the implementations of CPI-C interfaces.



As shown in the diagram above, EJB application runs as CPI-C client on WebLogic server and communicates to CPI-C server CICS program in `KIXS` region through WTC connection to Tuxedo Domain Gateway using conversational protocol. CPI-C client library establishes a connection through `tpconnect()` call for a CICS transaction published by `ARTCTRN` (ART CICS application server for Conversational transactions/programs) and uses Tuxedo conversational `tpsend()/tprecv()` API calls to mimic `CMSSEND/CMRCV`. Application code on both sides represented by blue shaded areas remains unchanged, with Tuxedo ART infrastructure components providing all the necessary API and communications support.

## Server Side Configuration

- [ART CICS Resources Configuration](#)
- [Oracle Tuxedo Configuration](#)

## ART CICS Resources Configuration

- [CICS Region Definitions in systems.desc](#)
- [CICS APPC Connection Definitions in connections.desc](#)
- [CICS Transaction Definitions in transactions.desc](#)

### CICS Region Definitions in systems.desc

Configure `sysid` for client and server. In the following example, in `KIXA`, for Windows client, `sysid` is `KIXA` and `applid` is `ARTKIXA`. In `KIXB`, for CPI-C server, `sysid` is `KIXB` and `applid` is `ARTKIXB`.

```
[KIXA]
APPLID=ARTKIXA
[KIXB]
APPLID=ARTKIXB
```

### CICS APPC Connection Definitions in connections.desc

Configure `protocol` in `connections.desc`. In the following example, `KIXA` connects to `KIXB`, and `protocol` is `APPC`. Resource group `DTPAPBK` is later used in transaction definition.

```
[KIXA]
group=DTPAPBK
protocol=APPC
netname=ARTKIXA
maximum=5,2

[KIXB]
group=DTPAPBK
protocol=APPC
netname=ARTKIXB
maximum=5,3
```

## CICS Transaction Definitions in transactions.desc

Configure CPI-C transaction in server side. The following example configures BC32 as a CPI-C transaction in server side.

```
BC32;DTPAPBK;APPC server; xxxxxxxx
```

## Oracle Tuxedo Configuration

- [UBBCONFIG Configuration](#)
- [DMCONFIG Configuration](#)

### UBBCONFIG Configuration

- Configure ARTCTRN server in `SERVERS` section. The following example specifies KIXB region in `CLOPTs` with `-s` and CICS resource group with `-l`. The `MIN/MAX` values can be adjusted to match the number of concurrent conversations that the application must support. Since `APPC/CPI-C` is a conversational mode protocol, the `ARTCTRN` server will block on `CMSSEND` until the client does `CMRCV`, and will block on `CMRCV` until the client performs `CMSSEND`.

```
ARTCTRN
    SRVGRP=GRP02
    SRVID=30
    CONV=Y
    MIN=1 MAX=1 RQADDR=QKIX030 REPLYQ=Y
    CLOPT="-o /stdout_ctrn -e /stderr_ctrn -r -- -s KIXB -l
DTPSUB:DTPAPBK"
```

- Configure workstation listener (`WSL`) server in `SERVERS` section. In the following example, two `WSHs` are started initially, up to maximum of five `WSHs` can be started, up to 5 `WS` clients per `WSH`.

```
WSL SRVGRP=G1 SRVID=10 CLOPT="-A -- -n //gumby:9977 -m 2 -M 5 -x5"
```

- Configure `MACHINES` section.
  - `MAXWSCLIENTS`: maximum numbers of `WSCs` for each machine is specified.
  - `MAXACCESSERS`: `MAXWSCLIENTS` + number of Tuxedo servers connected to the bulletin board (including all servers listed in `UBBCONFIG`, plus maximum allowed `WSL/WSH` servers).

- Configure domain servers in `SERVERS` section for connecting WebLogic server. For example,

```
DMADM          SRVID=1030      SRVGRP=DMGRP
GWADM          SRVID=1040      SRVGRP=GWGRP
GWTDOMAIN      SRVID=1050      SRVGRP=GWGRP
```

**Note:** Make sure you compile `UBBCONFIG` with `tmloadcf`.

### DMCONFIG Configuration

Configure `DMCONFIG` for WTC Tuxedo domain configuration. See an example as follows.

```
*DM_LOCAL_DOMAINS
```

```
DOM              GWGRP="GWGRP"
                 TYPE=TDOMAIN
                 DOMAINID=KIXD
```

```
*DM_REMOTE_DOMAINS
```

```
wldom1          TYPE=TDOMAIN DOMAINID=TDOM2  ACL_POLICY=GLOBAL
```

```
*DM_TDOMAIN
```

```
wldom1          NWADDR="//10.0.0.1:5669"
```

```
DOM              NWADDR="//10.0.0.2:5022"
```

```
*DM_LOCAL_SERVICES
```

```
KIXR_CPIS       LDOM=DOM
```

**Note:** Make sure you compile `DMCONFIG` with `dmloadcf`.

## Client Side Configuration

- [Configuration for Windows Client](#)



- [Configuration for WebLogic Client](#)

## Configuration for Windows Client

Windows client uses `sym_dest_name` set by `cminit()` to connect the target ART CICS server. `sym_dest_name` should be set as the `sysid` of target ART CICS server.

- To use Tuxedo/WS client, two environment variables should be set on Windows:

- `TUXDIR=c:/tuxedo`

This is the Tuxedo install location.

- `WSNADDR=//gumby:9977`

This is the hostname and port of Tuxedo server WSL connection from `-n` parameter in WSL's CLOPTs.

- `KIX_CPI-C_WSSYSID` is introduced to specify CPI-C /WS client `sysid`. It is needed to establish the connection between CPI-C /WS client and CPI-C CICS server. It should be set to client `sysid` listed in `connections.desc` (KIXA in this example).
- To track CPI-C client runtime log, a new environment variable is introduced to show the log file path. If `ARTKIX_CLIENT_LOGPATH` is not set, the log will be printed to local directory.

```
ARTKIX_CLIENT_LOGPATH=c:/tmp/ARTKIX_client.log
```

## Configuration for WebLogic Client

No specific configuration is required for using CPI-C java classes other than a basic WTC access point configuration for connecting to Tuxedo Domain as documented in [Administering WTC manual](#). WTC supports `on_startup` and `on_demand` connection policies and can support failover and failback between primary and alternate access points to multiple Tuxedo domains or a single domain deployed across multiple machines (MP mode domain).

# Oracle Tuxedo Timeout Controls

Tuxedo enforces multiple types of timeouts through configuration. The timeout control for a blocked operation depends on `SCANUNIT` and `BLOCKTIME` settings in `UBBCONFIG`. Both of these can be set globally in `RESOURCES` section, and `BLOCKTIME` can additionally be set per service in `SERVICES` section of `UBBCONFIG`. See [Settings in UBBCONFIG](#) for more information.

This is an example of global timeout management in `UBBCONFIG`. In this example, the `BLOCKTIME` x `SCANUNIT` is 40 seconds. The client will be blocking on `CMRCV` until it really gets the response. If it does not complete in 40 seconds, it then returns `TPETIME` to report timeout and `CMRCV` return error `CM_RESOURCE_FAILURE_RETRY`.

```
*RESOURCES
```

```
    BLOCKTIME      8
    SCANUNIT       5
```

This is an example of granular timeout management for transactions in `UBBCONFIG`. In this example, it specifies that global timeout setting is 60 seconds (`12` x `5`), but Tuxedo service `KIXB_B32` (which maps to CICS transaction `B32` in `region/SYSID KIXB`) has a timeout of 40 seconds (`8`x`5`). Note that other transactions that have no explicit `BLOCKTIME` specified in `*SERVICES` section will be controlled by the global timeout setting.

```
*RESOURCES
```

```
    SCANUNIT       5
    BLOCKTIME      12
```

```
...
```

```
*SERVICES
```

```
    KIXB_B32 BLOCKTIME 8
```

## Settings in UBBCONFIG

### **SCANUNIT numeric\_value**

The interval of time (in seconds) between which periodic scans are done by the BBL to find old transactions and timed-out blocking calls within service requests. This value is used as the basic unit of scanning by the BBL. It affects the granularity with which transaction timeout values can be specified on `tpbegin()` and the blocking timeout value specified with the `BLOCKTIME` parameter. The `SANITYSCAN`, `BBLQUERY`, `DBBLWAIT`, and `BLOCKTIME` parameters are multipliers of this unit for other timed operations within the system. `SCANUNIT` must be a multiple of 5 greater than 0 and less than or equal to 60 seconds. The default is 10 seconds.

### **BLOCKTIME numeric\_value**

Sets a multiplier of the basic `SCANUNIT` after which a blocking call (for example, receiving a reply) times out. The value of `BLOCKTIME` must be greater than 0. If this parameter is not

specified, the default is set so that (`SCANUNIT * BLOCKTIME`) is approximately 60 seconds.

## Security

ART CICS CPI-C checks user/password passed from CPI-C security interface (`cmscsu/cmscsp`), and then it does local Tuxedo security check inside. The user/password should be added based on security levels set in Tuxedo security framework.

## Scaling

To scale the configuration to support larger number of concurrent connections, you can:

- Configure multiple `WSLS` and increase `WSH` min/max limits in Tuxedo domain for handling more concurrent Windows connections.
- Configure multiple Tuxedo domain gateways to handle multiple `WTC` access points for more concurrent WebLogic connections.
- Configure multiple `ARTCTRN` servers to run more instances of the CICS transactions in parallel. Since CPI-C is a conversational protocol, each server will block waiting for user response. The number of servers you configure should roughly correspond to the number of concurrent users of the CICS transactions.

## Diagnostics

Runtime log will be printed for debugging. You can set `KIX_TRACE_LEVEL` environment variable to control log level from 1 to 9.

## Packaging/Installation

ART CICS runtime provides a dynamic library for ART CICS CPI-C support. The Windows version of the library is named `kixcpicws.dll` and Linux version is named `libkixcpicws.so`. To use ART CICS CPI-C library, you only need to build/link with this library instead of Microsoft/IBM CPI-C library in your build project.

- For Windows/Linux Using Tuxedo Workstation Client

The Windows library is provided in a separate Windows distribution package. Current version of the library `kixcpicws.dll` is built on VC6 and certified with Tuxedo Workstation Client from Microsoft Windows (32-bit) `tuxedo81_win` distribution for

Windows server 2003. To use the library, you need to link `kixcpicws.dll` with the application or dynamically open the `*.dll` in the runtime.

- For WebLogic Server Using WTC

ART CICS provides source code of an overloaded `callService()` method, and provides a series of Java class files to provide the implementations of CPI-C interfaces. You need to regenerate the EJB package with new version of `callService()` and java class files provided by ART CICS.