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 ............................................... 3-13
Using the Tuxedo tmadmin psr Commands ............................................. 3-13
Using the Tuxedo tmadmin psc Commands ............................................. 3-14
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 .................. 4-2
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 tmadmin psr Commands ........................................... 4-13
Using the Tuxedo tmadmin psc Commands .......................................... 4-13
Using the CICS Runtime Application .................................................. 4-14
Presentation of Simple Application on COBOL-IT / BDB .................. 4-16
Configuring ubbconfig File in CICS Runtime ..................................... 4-16
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 ......................................................... 4-18
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 ................ 4-28
Implementing Asynchronous Transactions With ARTSRM Server .... 4-29
Implementing Asynchronous Transactions With /Q .......................... 4-30
Implementing CICS Application Using Temporary Storage (TS) Queues 4-34
Implementing Unrecoverable TS Queues ........................................ 4-37
Implementing Recoverable TS Queues ........................................ 4-38
Managing TD Queue Intrapartitions ......................................... 4-41
  Presentation of the Mechanism on Source Platform ............... 4-41
  Automatic Transaction Initiation (ATI) ................................. 4-43
  Presentation of the Mechanism on Target Platform ............... 4-43
  Runtime CICS Configuration of TD Queue Intrapartition ....... 4-45
  Activating the ARTTDQ in the Tuxedo ubbconfig File .......... 4-48
Implementing Distributed Program Link (DPL) ......................... 4-50
  To Detect That DPL Is Needed ........................................... 4-50
  Modifying the Tuxedo ubbconfig File to Manage the DPL .... 4-52
  Declaring Remote Programs in CICS Runtime ...................... 4-56
Implementing CICS Common Work Area (CWA) ......................... 4-61
Implementing a CICS Transaction Work Area (TWA) ................. 4-62
  Supporting TWA in ARTDPL .............................................. 4-66
Implementing CICS Transaction Trigger Monitor (ARTCKTI) .... 4-67
  Work Flow ................................................................. 4-67
  Command Configuration .................................................. 4-68
Implementing Transferring CICS Regions ................................. 4-69
  Configuring ARTSRM Server ............................................. 4-69
  Configuring Environment Variables .................................. 4-69
  CICS Runtime Configuration Files Declaration ................... 4-69
  Logon ART CICS .......................................................... 4-76
Implementing Intersystem Communication ............................... 4-77
  Implementing Distributed Transaction Processing (DTP) .... 4-77
  Implementing Asynchronous Processing ............................. 4-81
  Implementing Synchronous Processing ................................ 4-84
Implementing Submitting JCL Online ..................................... 4-86
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring the UBBCONFIG File</td>
<td>4-86</td>
</tr>
<tr>
<td>Configuring tdqextra.desc</td>
<td>4-87</td>
</tr>
<tr>
<td>Implementing Printing CICS Runtime Applications Data</td>
<td>4-88</td>
</tr>
<tr>
<td>General Configurations</td>
<td>4-88</td>
</tr>
<tr>
<td>Implementing Printing with a START Command</td>
<td>4-91</td>
</tr>
<tr>
<td>Implementing Printing with Transient Data</td>
<td>4-92</td>
</tr>
<tr>
<td>Implementing Invoking Web Services from CICS Applications</td>
<td>4-93</td>
</tr>
<tr>
<td>Converting WSDL File into MIF</td>
<td>4-93</td>
</tr>
<tr>
<td>Generating VIEW32 Definition from MIF</td>
<td>4-93</td>
</tr>
<tr>
<td>Generating Copybook from VIEW32 Definition (Optional)</td>
<td>4-94</td>
</tr>
<tr>
<td>Modifying VIEW32 Definition Manually (Optional)</td>
<td>4-94</td>
</tr>
<tr>
<td>Configuring webservice.desc</td>
<td>4-98</td>
</tr>
<tr>
<td>Modifying UBBCONFIG</td>
<td>4-99</td>
</tr>
<tr>
<td>CICS Runtime Logs</td>
<td>4-99</td>
</tr>
<tr>
<td>Tuxedo System Log</td>
<td>4-99</td>
</tr>
<tr>
<td>The CICS Runtime Server Logs</td>
<td>4-100</td>
</tr>
<tr>
<td>Disabling and Enabling Programs</td>
<td>4-103</td>
</tr>
<tr>
<td>Disabling Programs</td>
<td>4-103</td>
</tr>
<tr>
<td>Enabling Programs</td>
<td>4-104</td>
</tr>
<tr>
<td>Checking the Change in Program Status</td>
<td>4-104</td>
</tr>
<tr>
<td>Removing and Adding Applications for CICS Runtime</td>
<td>4-106</td>
</tr>
<tr>
<td>CICS Runtime C Program Support</td>
<td>4-108</td>
</tr>
<tr>
<td>Running C Program in CICS Runtime</td>
<td>4-108</td>
</tr>
<tr>
<td>C Programming Restrictions and Requirements</td>
<td>4-108</td>
</tr>
<tr>
<td>Accessing EIB from C</td>
<td>4-110</td>
</tr>
<tr>
<td>Accessing COMMAREA from C</td>
<td>4-110</td>
</tr>
<tr>
<td>CICS Command Translator</td>
<td>4-111</td>
</tr>
<tr>
<td>C Program Compilation</td>
<td>4-111</td>
</tr>
</tbody>
</table>
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

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)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>C-1</td>
</tr>
<tr>
<td>EXCI in Oracle Tuxedo Application Runtime</td>
<td>C-2</td>
</tr>
<tr>
<td>Supported EXCI Interface</td>
<td>C-2</td>
</tr>
<tr>
<td>Precompiler Controls</td>
<td>C-3</td>
</tr>
<tr>
<td>Access Authorization</td>
<td>C-3</td>
</tr>
<tr>
<td>ART CICS Implementation</td>
<td>C-4</td>
</tr>
</tbody>
</table>
CHAPTER 1

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.
Overview of the CICS Runtime

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 thru the `EXEC CICS` API.

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

Listing 2-1  z/OS CICS Calls

```plaintext
*EXEC CICS
*    RECEIVE MAP ('RTSAM10')
```
On UNIX, the CICS Runtime Preprocessor transforms these EXEC CICS into calls to the CICS Runtime library:

Listing 2-2  CICS Runtime Calls

```plaintext
*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
```
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.
The CICS Runtime Tuxedo Servers

- Converse Management server ARTCTR and ARTCTR1.
- The Web Transaction servers ARTWTRN and ARTWTR1: manage synchronous (no-conversational) non-3270s clients oriented transactions.

Server Generation

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

**Note:** ART CICS Runtime provides the `tcxcsdcvt` 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.
CHAPTER 3

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 ...).
This file should then be exported.

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

**Table 3-1 .profile Variables**

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

**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**

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

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

```bash
# <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
FIELDS32=msgflds32
OBJECT_MODE=64

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

# Command executable paths
HAB_TRAN=none

# Other environment
LC_MESSAGES=C
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 (ARTATRN and ARTATR1) that manage asynchronous transaction CICS applications.
• 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=QCNX015 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.

-0
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.
Initial Configuration of the CICS Runtime

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, ARTATRI, 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"
hilight=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.
Initial Configuration of the CICS Runtime

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 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 tmadmin 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 \texttt{tmadmin 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 \texttt{ARTADM}, \texttt{ARTTCPL}, and \texttt{ARTCNX} only are started, the following messages are displayed:

\textbf{Listing 3-8} \texttt{tmadmin psr} Command Example

\begin{verbatim}
# tmadmin ...

> 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
#
\end{verbatim}

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

\textbf{Using the Tuxedo \texttt{tmadmin psc} Commands}

You can also check that the required Tuxedo services are running using the \texttt{tmadmin 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 \texttt{tmadmin psc} Command Example

```
# tmadmin
...

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>2</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>1</td>
<td>AVAIL</td>
</tr>
<tr>
<td>delsess</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>gensess</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>1</td>
<td>AVAIL</td>
</tr>
<tr>
<td>update</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>inquire</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

> 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.
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 Distributed Program Link (DPL)
- Implementing CICS Common Work Area (CWA)
- Implementing a CICS Transaction Work Area (TWA)
- Implementing CICS Transaction Trigger Monitor (ARTCKTI)
- Implementing Transferring CICS Regions
- Implementing Intersystem Communication
- Implementing Submitting JCL Online
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSAM00</td>
<td>Customer maintenance entry menu</td>
</tr>
<tr>
<td>RSSAM01</td>
<td>Customer data inquiry screen</td>
</tr>
<tr>
<td>RSSAM02</td>
<td>Customer data maintenance screen (create, update and delete customer)</td>
</tr>
<tr>
<td>RSSAM03</td>
<td>Customer list screen</td>
</tr>
</tbody>
</table>

Programs

Table 4-2 Simple Application Programs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSAT000</td>
<td>Customer maintenance entry program</td>
</tr>
<tr>
<td>RSSAT001</td>
<td>Customer data inquiry program</td>
</tr>
<tr>
<td>RSSAT002</td>
<td>Customer data maintenance program (new customer, update and delete customer)</td>
</tr>
<tr>
<td>RSSAT003</td>
<td>Customer list program</td>
</tr>
</tbody>
</table>

Transactions Codes

Table 4-3 Simple Application Transactions Codes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA00</td>
<td>Main entry transaction code (program RSSAT000)</td>
</tr>
<tr>
<td>SA01</td>
<td>Customer inquiry (program RSSAT001)</td>
</tr>
<tr>
<td>SA02</td>
<td>Customer maintenance (program RSSAT002)</td>
</tr>
<tr>
<td>SA03</td>
<td>Customer list (program RSSAT003)</td>
</tr>
</tbody>
</table>
VSAM File

Table 4-4  Simple Application VSAM File

<table>
<thead>
<tr>
<th>DDName</th>
<th>DataSetName</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODCSF0</td>
<td>PJ01AAA.SS.VSAM.CUSTOMER</td>
<td>VSAM Main Customer File</td>
</tr>
</tbody>
</table>

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).
Implementing CICS Applications

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 declaration 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]
Implementing CICS Applications

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.
   
   ```bash
   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:

   ```bash
   #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 -1 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.
-I 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:

```java
*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 tmadmin psr Commands

Enter the Tuxedo tmadmin 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 tmadmin psr Simple Application Installation Check

```
# tmadmin
...

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>2</td>
<td>100 ( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101 TCP00</td>
<td>101</td>
<td>0</td>
<td>0 ( IDLE )</td>
<td></td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>2</td>
<td>100 ( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>6</td>
<td>300 ( IDLE )</td>
</tr>
</tbody>
</table>

> quit
```

Using the Tuxedo tmadmin psc Commands

Another possible check can be made by entering the Tuxedo tmadmin 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  tmadmin psc Simple Application Installation Check

```bash
# tmadmin
...

> psc
```

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>1</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>1</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>3</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>3</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

> 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:

```bash
# 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:
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
RVGRP=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.

RVGRP=
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 first three lines of the following text in $TUXDIR/udataobj/RM:

```
# BDB
BDB_HOME=/opt/cobol-it-64-bdb
BERKELEY-DB:db_xa_switch:-L/opt/cobol-it-64-bdb/lib -l_db-5
```

Build BDB TMS

```
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`
  ```bash
  export DD_RBDB02=${DATA}/MTWART.ES.SFI.RCIBDB02.BDB0122
  ``
  
  RBDB02 is the logical file name.

- `COB_ENABLE_XA`
  ```bash
  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 \texttt{MAX} and \texttt{MIN} parameters of the \texttt{ARTSTRN} definition in the \texttt{*SERVERS} section of the Tuxedo file \texttt{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 \texttt{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 \texttt{MAXACTIVE} superior or equal to 2 are managed by the CICS Runtime Tuxedo Server \texttt{ARTSTRN} and do not required modifying anything else. For the transactions with a \texttt{MAXACTIVE} parameter set to 1, an CICS Runtime Tuxedo Server named \texttt{ARTSTR1} is dedicated to their specific management.

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

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

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

Oracle Tuxedo Application Runtime for CICS User Guide  4-19
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.

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

**Note:** All of the CICS Runtime Transaction Servers (ARTSTRN, ARTSTR1, ARTATRN and ARTATR1) share the same CICS Runtime Transaction Group Servers, no modifications are required to the ubbconfig Server Group Section (*GROUPS).*
Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances

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:

```plaintext
TRCLASS1;SIMPAPP ; Tranclass with maxactive set to 1; 1
TRCLASS2;SIMPAPP ; Tranclass with maxactive set to 2; 2
TRCLASS10;SIMPAPP ; Tranclass with maxactive set to 10; 10
```

The first tranclass TRCLASS1 has is maxactive parameter equal to 1, indicating that all the transaction 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: TRCLAS10

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

**Listing 4-11  Example 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; ; ; ; ; ; ; ;TRCLASS1
SA02;SIMPAPP; Customer Maintenance Screen of the Simple Application;RSSAT002; ; ; ; ; ; ; ;TRCLASS2
SA03;SIMPAPP; Customer List of the Simple Application;RSSAT003; ; ; ; ; ; ; ;TRCLASS10
```
Implementing Synchronous CICS Transactions With a Limited Number of Parallel Instances

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 tmadmin psr Commands

The ARTSTR1, is shown below:

Listing 4-12  Checking the ARTSTR1 Server with the tmadmin psr Commands

```plaintext
# tmadmin
...

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load</th>
<th>Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>3</td>
<td>150</td>
<td></td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>
```
Using the Tuxedo tmadmin 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 tmadmin psc Commands**

```bash
# tmadmin
...

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

> 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 *tmadmin 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 (*SYNC_SA00* to *SYNC_SA03*) to allow them to run in an asynchronous mode.

Listing 4-14  tmadmin Commands Showing Parallel Asynchronous Transactions

```
# tmadmin
...
```
Implementing CICS Applications

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>4</td>
<td>200</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTATRN</td>
<td>QKIXATR</td>
<td>GRP02</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QUEUE</td>
<td>ASYNC_QUEUE</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA03</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA02</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>
Using Non-Parallel Asynchronous Transactions

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

Please refer to the section describing the reasons and the installation of the \texttt{ARTSTR1} server to install the \texttt{ARTSTR1} server.

To check your setting, you can use also the Tuxedo \texttt{tmadmin psr} and \texttt{psc} commands.

For the Simple Application example we can see that:

- The \texttt{psr} command shows that a new server is running \texttt{ARTATR1}.
- The \texttt{psc} command shows that no new services are running.

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

<table>
<thead>
<tr>
<th># tmadmin</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&gt; psr</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTATR1</td>
<td>00012.00300</td>
<td>GRP02</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>4</td>
<td>200</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>
Implementing CICS Applications

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

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

> 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

4-28 Oracle Tuxedo Application Runtime for CICS User Guide
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_srm -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 TRAANCTL(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

```
Implementing Asynchronous CICS Delayed Transactions

# /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:
Implementing Asynchronous CICS Delayed Transactions

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

```plaintext
# tmadmin
...

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTATR1</td>
<td>00012.00300</td>
<td>GRP02</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>4</td>
<td>200</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>3001</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>3002</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMQUEUE</td>
<td>01000.01010</td>
<td>GQUEUE</td>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMQFORWARD</td>
<td>01000.01020</td>
<td>GQUEUE</td>
<td>1020</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTATRN</td>
<td>QKIXATR</td>
<td>GRP02</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>3001</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>3002</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QSPACE</td>
<td>TMQUEUE</td>
<td>TMQUEUE</td>
<td>GQUEUE</td>
<td>1010</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

Oracle Tuxedo Application Runtime for CICS User Guide 4-33
### Implementing CICS Applications 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.

---

<table>
<thead>
<tr>
<th>Queue Name</th>
<th>Transaction Type</th>
<th>Artistic Name</th>
<th>Group ID</th>
<th>QP M</th>
<th>XSR</th>
<th>AVAIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>ASYNC_QUEUE</td>
<td>ASYNC_QUEUE</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA03</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA02</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA01</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
<tr>
<td>ASYNC_SA00</td>
<td>atrsvc</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0 AVAIL</td>
</tr>
</tbody>
</table>

> quit

#
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 \texttt{tmadmin psr} and \texttt{psc} commands to check that the server is running and that six
new services are published:

\begin{verbatim}
# tmadmin
...

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTATR1</td>
<td>00012.00300</td>
<td>GRP02</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>3</td>
<td>150</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTSQ</td>
<td>00012.00040</td>
<td>GRP02</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\end{verbatim}
Implementing CICS Application Using Temporary Storage (TS) Queues

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 enqueuing 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 enqueuing 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 Server 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

```
# tmadmin
```
Implementing CICS Applications

...  

> psr

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

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS ORA</td>
<td>GRP02</td>
<td>30001</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS ORA</td>
<td>GRP02</td>
<td>30002</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS ORA</td>
<td>GRP02</td>
<td>30003</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>
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 (\texttt{WRITEQ TD} command).
- Read data from a transient data queue (\texttt{READQ TD} command).
- Delete an intrapartition transient data queue (\texttt{DELETEQ TD} command).

\textbf{Note:} In this document we concentrate exclusively on intrapartition TD queues.

\section*{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:

\textbf{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.

\textbf{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**

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

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 ARTATRN.

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.

QSPACENAME

New argument needed for /Q: defining into which QSPACE the current /Q is stored. This argument is mandatory and must match the QSPACENAME 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 mkTd_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 contain 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 QSPACENAME 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 enqueuing (top, msgid, [default=none]):  none
Retries [default=0]:  5
Retry delay in seconds [default=0]:  0
```
Managing TD Queue Intrapartitions

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 enqueuing
  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 rollbacked 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/demoLogs/TUX/sysout/stdout_tdq -e
/home2/work9/demoLogs/TUX/sysout/stderr_tdq -r -- -s KIXR -1 SIMPAPP"
...

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.
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 4-27 Checking for Remote Programs

```plaintext
DEF PROGR
OVERTYPE TO MODIFY                                    CICS RELEASE = 0610
CEDA DEFINE PROGRAM(                                  )
  PROGRAM     ==>                                    
  Group       ==>                                    
  DESCRIPTION ==>                                    
  ....
```
REMOTE ATTRIBUTES

DYnamic ==> No               No ! Yes
REMOTESystem ==> XXXX
REMTENae ==> YYYYYYY
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).

REMTENAME(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 4-28  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, ARTSTRI, ARTATRN, ARTATRI`).

**Listing 4-29  ubbconfig File Example of a *SERVERS Section Describing the ARTDPL Server.**

```
*SERVERS
```
Implementing Distributed Program Link (DPL)

... 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.

-l 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 4-30  tmadmin Commands to Check ARTDPL Server**

```
# tmadmin
...

> psr
```

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load</th>
<th>Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTDPL</td>
<td>QKIXDPL</td>
<td>GRP02</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTATR1</td>
<td>00012.00300</td>
<td>GRP02</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>5</td>
<td>250</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMSORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARRTCKPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>30002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMSORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMSORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30003</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>
Implementing Distributed Program Link (DPL)

<table>
<thead>
<tr>
<th>Service</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMQUEUE</td>
<td>01000.01010</td>
<td>GQUEUE</td>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
<tr>
<td>TMQFORWARD</td>
<td>01000.01020</td>
<td>GQUEUE</td>
<td>1020</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
<tr>
<td>ARTATRN</td>
<td>QKIXATR</td>
<td>GRP02</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
<tr>
<td>ARTTSQ</td>
<td>00012.00040</td>
<td>GRP02</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>(IDLE  )</td>
<td></td>
</tr>
</tbody>
</table>

> PSC

<table>
<thead>
<tr>
<th>Service</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>30001</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_ORA</td>
<td>GRP02</td>
<td>30001</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>30002</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_ORA</td>
<td>GRP02</td>
<td>30002</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_ORA</td>
<td>GRP02</td>
<td>30003</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QSPACE</td>
<td>TMQUEUE</td>
<td>TMQUEUE</td>
<td>GQUEUE</td>
<td>1010</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA02</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QUEUE</td>
<td>ASYNC_QUEUE</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>
Implementing CICS Applications

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 4-29).

     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 4-31  Simple Application programs.desc Configuration of Remote Programs

```plaintext
#PROGRAM;GROUP;DESCRIPTION;LANGUAGE;EXECKEY;STATUS;REMOTESYSTEM;REMTONAME
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
```
2. Shutdown and reboot Tuxedo.

3. Using the Tuxedo `tmadmin 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 4-32  Using `tmadmin` Commands to Check DPL Services

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

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTDPL</td>
<td>QKIXDPL</td>
<td>GRP02</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTATR1</td>
<td>00012.00300</td>
<td>GRP02</td>
<td>300</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>00012.00200</td>
<td>GRP02</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>BBL</td>
<td>200933</td>
<td>KIXR</td>
<td>0</td>
<td>5</td>
<td>250</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_ORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEUE_TMS</td>
<td>GQUEUE</td>
<td>30002</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_ORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30002</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_ORA</td>
<td>GRP02_TMS</td>
<td>GRP02</td>
<td>30003</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMQUEUE</td>
<td>01000.01010</td>
<td>GQUEUE</td>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>
```
## Implementing CICS Applications

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMQFORWARD</td>
<td>01000.01020</td>
<td>GQUEUE</td>
<td>1020</td>
<td>0</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP02</td>
<td>20</td>
<td>0</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ARTATRN</td>
<td>QKIXATR</td>
<td>GRP02</td>
<td>30</td>
<td>0</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ARTTSQ</td>
<td>00012.00040</td>
<td>GRP02</td>
<td>40</td>
<td>0</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>

> psc

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>KIXD_RSSAT0+</td>
<td>dplsvc</td>
<td>ARTDPL</td>
<td>GRP02</td>
<td>500</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>KIXD_RSSAT0+</td>
<td>dplsvc</td>
<td>ARTDPL</td>
<td>GRP02</td>
<td>500</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>3001</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>3002</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>TMS</td>
<td>TMS</td>
<td>TMS_QM</td>
<td>GQUEUE</td>
<td>3003</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QSPACE</td>
<td>TMQUEUE</td>
<td>TMQUEUE</td>
<td>GQUEUE</td>
<td>1010</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA03</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA01</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>SA00</td>
<td>kixsvc</td>
<td>ARTSTRN</td>
<td>GRP02</td>
<td>20</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>ASYNC_QUEUE</td>
<td>ASYNC_QUEUE</td>
<td>ARTATRN</td>
<td>GRP02</td>
<td>30</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>
Implementing Distributed Program Link (DPL)

ASYNC_SA03  atrsvc   ARTATRN   GRP02     30       KIXR       0 AVAIL
ASYNC_SA01  atrsvc   ARTATRN   GRP02     30       KIXR       0 AVAIL
ASYNC_SA00  atrsvc   ARTATRN   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
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 psc 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 4-33  Using tmadmin Commands to Check Specific DPL Service in Verbose Mode**

```bash
# tmadmin
...

> verbose
Verbose now on.

> psc -i 500
```
Implementing CICS Applications

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
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 4-34 COBOL Example of CWA Usage**

```cobol
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.

  * Set up addressability to the CWA
  EXEC CICS ADDRESS
      CWA(ADDRESS OF COMMON-WORK-AREA)

  END-EXEC.
```

Requests Done: 0
Current status: AVAILABLE

> quit
#
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.
Implementing a CICS Transaction Work Area (TWA)

Listing 4-35  A COBOL Example of Use of the TWA

```cobol
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).
   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:
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

<table>
<thead>
<tr>
<th>Transaction</th>
<th>TWA Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA00</td>
<td>0</td>
</tr>
<tr>
<td>SA01</td>
<td>100</td>
</tr>
<tr>
<td>SA02</td>
<td>200</td>
</tr>
<tr>
<td>SA03</td>
<td>300</td>
</tr>
</tbody>
</table>

### Listing 4-36  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 4-37  stderr_strn TWA Example**

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANS</td>
<td>GROUP</td>
<td>PROGRAM</td>
<td>ALIAS</td>
<td>PRI</td>
<td>E</td>
<td>E</td>
<td>KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>R</td>
<td>TRAN</td>
<td>TWA</td>
<td>MAX</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>C</td>
<td>SIZ</td>
<td>ACT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRAN</td>
<td>GROUP</td>
<td>PROGRAM</td>
<td>ALIAS</td>
<td>PRI</td>
<td>E</td>
<td>E</td>
<td>KEY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>R</td>
<td>TRAN</td>
<td>TWA</td>
<td>MAX</td>
<td>DATA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TASK</td>
<td>C</td>
<td>SIZ</td>
<td>ACT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Oracle Tuxedo Application Runtime for CICS User Guide 4-65
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 4-38  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*.

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

   **Listing 4-39  stdout_dpl TWA Example**

   ```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTIONS loaded : &lt;  1&gt;</td>
</tr>
<tr>
<td>----------------------------------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
</tbody>
</table>
   ```
Implementing CICS Transaction Trigger Monitor (ARTCKTI)

Implementing 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**

![WebSphere MQ Trigger Condition Diagram]

**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.
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 4-40  Example for system.desc Configurations**

[kixr]

APPLID=DBDCkixR

INITPARM=(ASINTP='Hello world')

[kixl]
Implementing CICS Applications

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 4-41  Example for transactions.desc and programs.desc Configurations**

```
transactions.desc:
ISSSS;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.
Implementing Transferring CICS Regions

Listing 4-42  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 4-43  Example for DDR Configurations

GRP00

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

GRP01

GRPNO=11
ENVFILE="/home2/work9/demo/config/tux/envfile"
Implementing CICS Applications

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
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 --"

...
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 4-44  Example for Environment Variable Declaration

/home2/work9/demo> tmadmin> tmadmin
...

> psr

<table>
<thead>
<tr>
<th>Prog Name</th>
<th>Queue Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>RqDone</th>
<th>Load</th>
<th>Done</th>
<th>Current Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>BBL</td>
<td>34790</td>
<td>KIXR</td>
<td>0</td>
<td>55</td>
<td>2750</td>
<td>( IDLE )</td>
<td></td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEKIXL_T+</td>
<td>GQUEKIXL</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
<td></td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEKIXR_T+</td>
<td>GQUEKIXR</td>
<td>30001</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
<td></td>
</tr>
<tr>
<td>ARTTCPL</td>
<td>00001.00101</td>
<td>TCP00</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>( IDLE )</td>
<td></td>
</tr>
</tbody>
</table>
### Implementing Transferring CICS Regions

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS_QM</td>
<td>GQUEKIXL_T+</td>
<td>GQUEKIXL</td>
<td>30002</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMS_QM</td>
<td>GQUEKIXR_T+</td>
<td>GQUEKIXR</td>
<td>30002</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMQUEUE</td>
<td>01020.01210</td>
<td>GQUEKIXL</td>
<td>1210</td>
<td>1</td>
<td>50</td>
<td></td>
<td>( IDLE )</td>
</tr>
<tr>
<td>TMQUEUE</td>
<td>01010.01110</td>
<td>GQUEKIXR</td>
<td>1110</td>
<td>2</td>
<td>100</td>
<td></td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTADM</td>
<td>00011.00010</td>
<td>GRP01</td>
<td>10</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX015</td>
<td>GRP01</td>
<td>15</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>QCNX016</td>
<td>GRP02</td>
<td>16</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTLOGN</td>
<td>QLGN018</td>
<td>GRP00</td>
<td>18</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>QKIX110</td>
<td>GRP12</td>
<td>20</td>
<td>0</td>
<td></td>
<td>0</td>
<td>( IDLE )</td>
</tr>
</tbody>
</table>

...>

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Routine Name</th>
<th>Prog Name</th>
<th>Grp Name</th>
<th>ID</th>
<th>Machine</th>
<th># Done</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBDCkixL</td>
<td>TMQUEUE</td>
<td>TMQUEUE</td>
<td>GQUEK+</td>
<td>1210</td>
<td>KIXR</td>
<td>1</td>
<td>AVAIL</td>
</tr>
<tr>
<td>DBDCkixR</td>
<td>TMQUEUE</td>
<td>TMQUEUE</td>
<td>GQUEK+</td>
<td>1110</td>
<td>KIXR</td>
<td>2</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>update</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>inquire</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>authfail</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESF</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CESN</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>CSGM</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP01</td>
<td>15</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>disconnect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP02</td>
<td>16</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>connect</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP02</td>
<td>16</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>update</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP02</td>
<td>16</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
<tr>
<td>inquire</td>
<td>cnxsvc</td>
<td>ARTCNX</td>
<td>GRP02</td>
<td>16</td>
<td>KIXR</td>
<td>0</td>
<td>AVAIL</td>
</tr>
</tbody>
</table>
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 4-5 Logon Screen*
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.
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 4-6, 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

Figure 4-6 Typical End-to-End User Case
• 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 CICB
    • 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
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.

**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 4-45 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 4-45. Following is a configuration example.

Listing 4-46 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 4-47   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 4-47. Following is a configuration example.

Listing 4-48   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+LogDir=/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+LogDir=/LOGS/xa+DbgFl=0x20"

*SERVERS

ARTSTRN

SRVGRP=GRPKIXR
SRVID=1101
CONV=Y
MIN=1 MAX=1 RQADDR=QKIXSTRR REPLYQ=Y
CLOPT="-- -s KIXR -l SIMPAPP"

ARTSTRN

SRVGRP=GRPKIXX
SRVID=1201
CONV=Y
MIN=1 MAX=1 RQADDR=QKIXSTRX REPLYQ=Y
CLOPT="-- -s KIXX -l SIMPAPP"
Implementing Submitting JCL Online

On z/OS, CICS programs can submit JCL by the WRITEQ TD command and pass the JCL 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 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.

*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 Online

Listing 4-49  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 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.

For more information, refer to TD Queue Extra Partition Configuration File in Oracle Tuxedo Application Runtime for CICS Reference Guide.
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 4-50  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"
hilight=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 4-7  Configure a LUNAME**
Implementing Printing CICS Runtime Applications Data

Figure 4-8 Setting the PCOM Session Type

![Image of PCOM Session Type settings](image-url)

Implementing Printing with a START Command

Figure 4-9 Printing with a START Command

![Diagram showing implementation of printing with a START command](image-url)

Figure 4-9 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:

   ```
   #
   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=CICSPRT2`) 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 with 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

Convert your WSDL file into the Tuxedo metadata repository input file (MIF) using the Oracle SALT command utility, wsdlcvt. For more information, refer to Configuring an Oracle SALT Application.

Generating VIEW32 Definition from MIF

ART CICS provides a utility, Mif2View32, to generate VIEW32 definition according to MIF and updates the given FML32 definition file. The generated VIEW32 definition keeps the same structure as MIF.

Run the following command:

Mif2View32 -i miffile -o viewfile -f.fml32file -w webservice.desc [-v]

Where:

- i miffile
  Specifies the metadata input file name.
- o viewfile
  Specifies the output VIEW32 definition file name.
- f.fml32file
  Specifies the updated FML32 definition file.
- w webservice.desc
  Specifies the webservice.desc configuration file.
Indicates whether to output details or not. For more information, refer to Mif2View32(1).

**Generating Copybook from VIEW32 Definition (Optional)**

For applications that are created in ART CICS, run `viewc32 -C -S viewfile` to generate the COBOL copybook from the generated VIEW32 definition. For more information, refer to `viewc, viewc32(1)` in Oracle Tuxedo Command Reference.

**Modifying VIEW32 Definition Manually (Optional)**

For certain types of applications that are migrated from mainframe, you need to modify the VIEW32 definition fields and adjust the offset manually to enable one-to-one mapping between your copybook and VIEW32 definition.

Following instructions apply to the COBOL copybook that has relevant characteristics.

Check the following and make the modifications accordingly:

- **Container fields in copybook**
  
  In copybook, if there are container fields ended with `-cont`, for example:
  
  ```
  09 privateData-cont PIC X(16).
  ```

  The corresponding item in VIEW32 definition might look like:
  
  ```
  struct privateData_v privateData 100 - - 0
  ```

  To map to your copybook, change that line to:
  
  ```
  string privateData1 privateDataList_cont 1 - 16 '"0'
  ```

- **Nillable attributes in copybook**
  
  In the WSDL schema definition, if there are some items containing the attribute `nillable="false"`, for example:
  
  ```
  <element name="walletId" nillable="false" type="xsd:string"/>
  ```

  In copybook, there should be an extra item describing the attribute:
  
  ```
  09 attr-nil-walletId-value PIC X DISPLAY.
  ```

  In VIEW32 definition, since there is no corresponding item, you need to set the offset value to let the next item point to the correct address. For example:
string walletId_v walletId 1 - 255 "-4"

“-4” indicates the VIEW32 structure has four bytes offset from the next item.

- Alignment rule for copybook

Copybook has different alignment rule with C structure. When the alignment rule causes an offset between the copybook and VIEW32 definition, you need to set the offset value in VIEW32 definition appropriately.

- Extra items in VIEW32 definition

In VIEW32 definition, there are helper items indicating the occurrences number. For example:

```
short payment_v_times payment_times 1 - 0
```

In copybook, since there is no corresponding item, remove such items or set the offset value in VIEW32 definition to let the next item point to the correct address.

Here is an example. Suppose you have a copybook like Listing 4-52, and you have generated a VIEW32 definition as shown in Listing 4-53, to map the VIEW32 definition to copybook, you need to modify the VIEW32 to Listing 4-54.

**Listing 4-52  Copybook Example**

```c
03 doAuthorizationRequest.
   06 buyer.
   ...
   09 email-length  PIC S9(4) COMP-5 Sync.
   09 email        PIC X(255).
   09 attr-nil-email-value  PIC X DISPLAY.
   09 shippingAdress.
      12 attr-nil-shippingAdres-value  PIC X DISPLAY.
   12 name-length  PIC S9(4) COMP-5 Sync.
   12 name        PIC X(255).
   ...
   09 walletId-length  PIC S9(4) COMP-5 Sync.
```

Implementing CICS Applications

09 walletId PIC X(255).
09 attr-nil-walletId-value PIC X DISPLAY.
06 privateDataList.
09 attr-nil-privateDataLi-value PIC X DISPLAY.
09 privateData-num PIC S9(9) COMP-5 SYNC.
09 privateData-cont PIC X(16).

Listing 4-53  VIEW32 (Before) Example

VIEW buyer_v
#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
...
short email_v_times email_times 1 - - 0
string email_v email 1 - 255 '\0'
short shippingAdress_v_times shippingAdress_times 1 - - 0
struct shippingAdress_v shippingAdress 1 - - 0
...
short walletId_v_times walletId_times 1 - - 0
string walletId_v walletId 1 - 255 '\0'
END

VIEW shippingAdress_v
#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
short name_v_times name_times 1 - - 0
string name_v name 1 - 255 '\0'
...
END
VIEW privateData_v

#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
short key_v_times key_times 1 - - 0
string key_v key 1 - 255 '\0'
short value_v_times value_times 1 - - 0
string value_v value 1 - 255 '\0'
END

VIEW privateDataList_v

#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
short privateData_v_times privateData_times 1 - - 0
struct privateData_v privateData 100 - - 0
END

VIEW doAuthorizationRequest_v

#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
short buyer_v_times buyer_times 1 - - 0
struct buyer_v buyer 1 - - 0
short privateDataList_v_times privateDataList_times 1 - - 0
struct privateDataList_v privateDataList 1 - - 0
END

Listing 4-54  VIEW32 (After) Example

VIEW buyer_v

#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
... short email_v_times email_times 1 - - 0
string email_v email 1 - 255 "-2"
struct shippingAdress_v shippingAdress 1 - - 0
...
short walletId_v_times walletId_times 1 - - 0
string walletId_v walletId 1 - 255 "-4"
END

VIEW shippingAdress_v
#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
short name_v_times name_times 1 - - 0
string name_v name 1 - 255 '\0'
...
END

VIEW doAuthorizationRequest_v
#TYPE CNAME FBNAME COUNT FLAG SIZE NULL
struct buyer_v buyer 1 - - 0
short privateData_v_times privateData_times 1 - - 0
string privateData1 privateDataList_cont 1 - 16 '\0'
END

---

**Configuring webservice.desc**

Specify `NEWCPY` and `TRANSACTION` in the `webservice.desc` configuration file. Following is an example.
Listing 4-55   Example of Specifying NEWCPY and TRANSACTION in webservice.desc

[DFH0XCMNOperation]
    REQUEST=DFH0XCMNOperation_v
    RESPONSE=DFH0XCMNOperationResponse_v
    NEWCPY=Y
    TRANSACTION=N

Modifying UBBCONFIG

Configure the TMMETADATA and GWWS servers in the UBBCONFIG file. Following is an example.

Listing 4-56   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"

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>
<thead>
<tr>
<th>Server name</th>
<th>-o standard output file</th>
<th>-e standard error file</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTTCPL</td>
<td>stdout_tcp</td>
<td>stderr_tcp</td>
</tr>
<tr>
<td>ARTCNX</td>
<td>stdout_cnx</td>
<td>stderr_cnx</td>
</tr>
<tr>
<td>ARTSTRN</td>
<td>stdout_strn</td>
<td>stderr_strn</td>
</tr>
<tr>
<td>ARTSTR1</td>
<td>stdout_str1</td>
<td>stderr_str1</td>
</tr>
<tr>
<td>ARTATRN</td>
<td>stdout_atrn</td>
<td>stderr_atrn</td>
</tr>
<tr>
<td>ARTATR1</td>
<td>stdout_atr1</td>
<td>stderr_atr1</td>
</tr>
<tr>
<td>ARTTSQ</td>
<td>stdout_tsq</td>
<td>stderr_tsq</td>
</tr>
<tr>
<td>ARTDPL</td>
<td>stdout_dpl</td>
<td>stderr_dpl</td>
</tr>
</tbody>
</table>

**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 4-57  Example of the stdout_strn Just After Start Up for a ARTSTRN Server**

```
Groups loaded: <0001>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>SIMPAPP</td>
</tr>
<tr>
<td>----------</td>
</tr>
</tbody>
</table>
ARTSTRN: Read config done

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANCLASS loaded : &lt; 2&gt;</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>TRANCLASS</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>TRCLASS1</td>
</tr>
<tr>
<td>TRCLASS2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAMS loaded : &lt; 4&gt;</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>PROGRAM</td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>RSSAT000</td>
</tr>
<tr>
<td>RSSAT001</td>
</tr>
</tbody>
</table>
```
### Implementing CICS Applications

<table>
<thead>
<tr>
<th>RSSAT002</th>
<th>SIMPAPP</th>
<th>COBOL</th>
<th>USER</th>
<th>ENABLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSAT003</td>
<td>SIMPAPP</td>
<td>COBOL</td>
<td>USER</td>
<td>ENABLED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RSSAT002</th>
<th>SIMPAPP</th>
<th>COBOL</th>
<th>USER</th>
<th>ENABLED</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSSAT003</td>
<td>SIMPAPP</td>
<td>COBOL</td>
<td>USER</td>
<td>ENABLED</td>
</tr>
</tbody>
</table>

| TRANSACTIONS loaded: < 4> |

---

**Warning:**

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 `-l` 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.
Implementing CICS Applications

Listing 4-58  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 4-59  Log Report Showing Program Status

Groups loaded: <0001>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GROUP</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>SIMPAPP</td>
</tr>
<tr>
<td>----------</td>
</tr>
</tbody>
</table>
Disabling and Enabling Programs

ARTSTRN: Read config done

<table>
<thead>
<tr>
<th>TRANCLASS loaded : &lt; 2 &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANCLASS</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>TRCLASS1</td>
</tr>
<tr>
<td>TRCLASS2</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>PROGRAMS loaded : &lt; 4 &gt;</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>PROGRAM</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>RSSAT000</td>
</tr>
<tr>
<td>RSSAT001</td>
</tr>
<tr>
<td>RSSAT002</td>
</tr>
<tr>
<td>RSSAT003</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>TRANSACTIONS loaded : &lt; 4 &gt;</td>
</tr>
<tr>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>TASK</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>T</td>
</tr>
</tbody>
</table>
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 theses 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 4-60  Example of Application in ARTSTRN Server

<table>
<thead>
<tr>
<th>ARTSTRN</th>
<th>SRVGRP=GRP02</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRVID=20</td>
</tr>
<tr>
<td></td>
<td>CONV=Y</td>
</tr>
<tr>
<td></td>
<td>MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y</td>
</tr>
<tr>
<td></td>
<td>CLOPT=-o /home2/work9/demo/Logs/TUX/sysout/stdout_strn -e /home2/work9/demo/Logs/TUX/sysout/stderr_strn -r -- -s KIXR -l SIMPAPP</td>
</tr>
</tbody>
</table>

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 4-61  Example of Adding group1 and group2 in ARTSTRN Server

<table>
<thead>
<tr>
<th>ARTSTRN</th>
<th>SRVGRP=GRP02</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRVID=20</td>
</tr>
<tr>
<td></td>
<td>CONV=Y</td>
</tr>
<tr>
<td></td>
<td>MIN=1 MAX=1 RQADDR=QKIX110 REPLYQ=Y</td>
</tr>
<tr>
<td></td>
<td>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</td>
</tr>
</tbody>
</table>

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 4-62  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.
Implementing CICS Applications

- The EIB declarations are enclosed in `#ifndef` and `#endif` lines, but are **not** included in all translated files. ART CICS publishes header file `dfeiblk.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,
  ```c
  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}
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.

| Servers                  | List of Group | PR | OQ | TRA | NSA | ASS | DEL | TRA | MO | TSQ | EN | QU | EM | DE | OD | EL | TDQ | INT | RA | EXT | TDQ | ETE | RM | MAP | SET | TEM | MIN | ALS | ECT | VICE | Total |
|--------------------------|---------------|----|----|-----|-----|-----|-----|-----|----|-----|----|----|----|----|----|----|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ARTTCPL/                 |               |    |    |     |     |     |     |     |    |     |    |    |    |    |    |    |     |     |    |     |     |    |    |    |    |    |    |    |    |    |    | 1   | 1   |
| H                        |               |    |    |     |     |     |     |     |    |     |    |    |    |    |    |    |    |     |     |    |     |     |    |    |    |    |    |    |    |    |    |     |     |
| ARTCNX                   |               |    |    |     |     |     |     |     |    |     |    |    |    |    |    |    |    |     |     |    |     |     |    |    |    |    |    |    |    |    |    | 1   | 1   |
| ARTATR1                  | 1             | 1  | 1  | 1   | 1   | 1   | 1   | 1   | 1  | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1   | 1  | 1   | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 11  |
| ARTATRN                  | 1             | 1  | 1  | 1   | 1   | 1   | 1   | 1   | 1  | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1   | 1  | 1   | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 11  |
| ARTSTR1                  | 1             | 1  | 1  | 1   | 1   | 1   | 1   | 1   | 1  | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1   | 1  | 1   | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 12  |
| ARTSTRN                  | 1             | 1  | 1  | 1   | 1   | 1   | 1   | 1   | 1  | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1   | 1   | 1  | 1   | 1   | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 12  |

Oracle Tuxedo Application Runtime for CICS User Guide 5-1
<table>
<thead>
<tr>
<th>Servers</th>
<th>List of Group</th>
<th>PR</th>
<th>OG</th>
<th>NCL</th>
<th>TRA NSA CTIONS</th>
<th>TSQ MODEL</th>
<th>EN QU EM ODEL</th>
<th>TDQ INT RA</th>
<th>TDQ EXT RA</th>
<th>TYP ETE RM</th>
<th>MAP SET</th>
<th>SYS TEM</th>
<th>TER MIN ALS</th>
<th>CO NN ECT ION</th>
<th>WS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTCTR1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTCTR2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTWTR1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTWTR2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTDPL</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTTSQ</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTTDQ</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTADM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTCKTI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARTSRM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>2</td>
<td>9</td>
<td>7</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 `tcxcsdcvt` 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 types, tool `tctxcsdcvt` 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” of CICSREF.

Figure 6-1 depicts the file data stream model.

**Figure 6-1  File Data Stream Model**

![File Data Stream Model](image)

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>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANCLASS</td>
<td>TRANCLASS</td>
<td>Name of the transaction class.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual comment zone for description of the resource.</td>
</tr>
<tr>
<td>MAXACTIVE</td>
<td>MAXACTIVE</td>
<td>Defines the degree of parallelism of execution.</td>
</tr>
</tbody>
</table>

Table 6-2  PROGRAM Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRAM</td>
<td>PROGRAM</td>
<td>Name of the program.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual comment zone for description of the resource.</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>LANGUAGE</td>
<td>The language of the program, required to know how to communicate with it.</td>
</tr>
<tr>
<td>EXECKEY</td>
<td>EXECKEY</td>
<td>Reserved for future use. Concerns memory protection of CICS shared structures.</td>
</tr>
<tr>
<td>STATUS</td>
<td>STATUS</td>
<td>Specifies the program status.</td>
</tr>
</tbody>
</table>
### Table 6-2 PROGRAM Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTESYSTEM</td>
<td>REMOTESYSTEM</td>
<td>Specifies that the program is not offered locally but in a DPL server.</td>
</tr>
<tr>
<td>REMOTENAME</td>
<td>REMOTENAME</td>
<td>Specifies for a DPL program the name of the program on the distant site.</td>
</tr>
</tbody>
</table>

### Table 6-3 FILE Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>FILE</td>
<td>Name of the file; logical name of the file used in EXEC CICS related to this file.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual comment zone for description of the resource.</td>
</tr>
<tr>
<td>DISPOSITION</td>
<td>DISPOSITION</td>
<td>Specifies the disposition of this file.</td>
</tr>
<tr>
<td>DSNAME</td>
<td>DSNAME</td>
<td>Specifies the data set name to be used for this file.</td>
</tr>
<tr>
<td>JOURNAL</td>
<td>JOURNAL</td>
<td>Specifies whether you want automatic journaling for this file.</td>
</tr>
<tr>
<td>KEYLENGTH</td>
<td>KEYLENGTH</td>
<td>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).</td>
</tr>
<tr>
<td>OPENTIME</td>
<td>OPENTIME</td>
<td>Specifies when the file is opened.</td>
</tr>
<tr>
<td>READINTEG</td>
<td>READINTEG</td>
<td>Specifies the level of read integrity required for files defined with RLSACCESS (YES).</td>
</tr>
</tbody>
</table>
**Table 6-3  FILE Mapping**

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORDSIZE</td>
<td>RECORDSIZE</td>
<td>Specifies the maximum length in bytes of records in a remote file or a coupling facility data table.</td>
</tr>
<tr>
<td>REMOTENAME</td>
<td>REMOTENAME</td>
<td>Specifies the name of the file on the remote system.</td>
</tr>
<tr>
<td>REMOTESYSTEM</td>
<td>REMOTESYSTEM</td>
<td>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.</td>
</tr>
<tr>
<td>STATUS</td>
<td>STATUS</td>
<td>Specifies the initial status of the file following a CICS initialization.</td>
</tr>
</tbody>
</table>

**Table 6-4  Journaling Attributes in FILE Mapping**

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNLADD</td>
<td>JNLADD</td>
<td>Specifies if you want the add operations recorded on the journal nominated by the JOURNAL attribute.</td>
</tr>
<tr>
<td>JNLREAD</td>
<td>JNLREAD</td>
<td>Specifies the read operations you want recorded on the journal nominated by the JOURNAL attribute.</td>
</tr>
<tr>
<td>JNLSYNOCREAD</td>
<td>JNLSYNOCREAD</td>
<td>Specifies whether you want the automatic journaling records, written for READ operations to the journal, to be synchronous.</td>
</tr>
</tbody>
</table>
### Table 6-4  Journaling Attributes in FILE Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JNLSYNCWRITE</td>
<td>JNLSYNCWRITE</td>
<td>Specifies whether you want the automatic journaling records, written for WRITE operations to the journal, to be synchronous.</td>
</tr>
<tr>
<td>JNLUPDATE</td>
<td>JNLUPDATE</td>
<td>Specifies whether you want REWRITE and DELETE operations recorded on the journal.</td>
</tr>
</tbody>
</table>

### Table 6-5  TSQUEUE MODEL Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSMODEL</td>
<td>TSMODEL</td>
<td>Name of the TS Queue model.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual zone for description of the resource.</td>
</tr>
<tr>
<td>LOCATION</td>
<td>LOCATION</td>
<td>Specifies the kind of storage to use: file or memory.</td>
</tr>
<tr>
<td>PREFIX</td>
<td>PREFIX</td>
<td>Specifies the character string that is to be used as the prefix for this model.</td>
</tr>
<tr>
<td>XPREFIX</td>
<td>XPREFIX</td>
<td></td>
</tr>
<tr>
<td>RECOVERY</td>
<td>RECOVERY</td>
<td>Specifies whether or not queues matching this model are to be recoverable.</td>
</tr>
<tr>
<td>POOLNAME</td>
<td>POOLNAME</td>
<td>Specifies the 8-character name of the shared TS pool definition that you want to use with this TSMODEL definition.</td>
</tr>
</tbody>
</table>
Table 6-5  TSQUEUE MODEL Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE_SYSTEM</td>
<td>REMOTESYSTEM</td>
<td>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.</td>
</tr>
<tr>
<td>REMOTEPREFIX</td>
<td>REMOTEPREFIX</td>
<td>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</td>
</tr>
<tr>
<td>XREMOTEPREFIX</td>
<td>XREMOTEPREFIX</td>
<td>Specifies whether security checking is to be performed for queues matching this model.</td>
</tr>
</tbody>
</table>

Table 6-6  ENQMODEL Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQMODEL</td>
<td>ENQMODEL</td>
<td>Name of the ENQ model.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual zone for description of the resource.</td>
</tr>
<tr>
<td>ENQNAME</td>
<td>ENQNAME</td>
<td>Specifies the 1 to 255-character resource name.</td>
</tr>
</tbody>
</table>
If omitted or specified as blanks, matching enqueue models will have a local scope, else they will have a global scope.

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENQSCOPE</td>
<td>ENQSCOPE</td>
<td>If omitted or specified as blanks, matching enqueue models will have a local scope, else they will have a global scope.</td>
</tr>
<tr>
<td>STATUS</td>
<td>STATUS</td>
<td>E = Enabled; D = Disabled.</td>
</tr>
</tbody>
</table>

### Table 6-7 TDQUEUE Extra Partition Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDQUEUE</td>
<td>TDQUEUE</td>
<td>Specifies the 1- to 4-character name of a transient data queue.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual zone for description of the resource.</td>
</tr>
<tr>
<td>DDNAME</td>
<td>DDNAME</td>
<td>Specifies a 1-to 8-character value that may refer to a data set defined in the startup JCL.</td>
</tr>
<tr>
<td>DISPOSITION</td>
<td>DISPOSITION</td>
<td>Specifies the disposition of the data set (MOD; OLD; SHR).</td>
</tr>
<tr>
<td>ERROPTION</td>
<td>ERROPTION</td>
<td>(UNSUPPORTED) Specifies the action to be taken if an I/O error occurs.</td>
</tr>
<tr>
<td>OPENTIME</td>
<td>OPENTIME</td>
<td>(UNSUPPORTED) Specifies the initial status of the data set.</td>
</tr>
<tr>
<td>RECORDFORMAT</td>
<td>RECORDFORMAT</td>
<td>Specifies the record format of the data set.</td>
</tr>
<tr>
<td>PRINTCONTROL</td>
<td>PRINTCONTROL</td>
<td>(UNSUPPORTED) Specifies the control characters to be used.</td>
</tr>
</tbody>
</table>
### Table 6-7  TDQUEUE Extra Partition Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECORDSIZE</td>
<td>RECORDSIZE</td>
<td>Specifies the record length in bytes.</td>
</tr>
<tr>
<td>TYPEFILE</td>
<td>TYPEFILE</td>
<td>Specifies the type of data set the queue is to be associated with an input or output dataset.</td>
</tr>
<tr>
<td>DSNAME</td>
<td>DSNAME</td>
<td>Specifies the name of the file that is to be used to store records written to this extra partition queue.</td>
</tr>
<tr>
<td>SYSOUTCLASS</td>
<td>SYSOUTCLASS</td>
<td>(UNSUPPORTED) Specify the class of the SYSOUT data set.</td>
</tr>
<tr>
<td>TRT</td>
<td>----</td>
<td>Allow integrators and customers to make their own specific implementation of extra-partition queues.</td>
</tr>
</tbody>
</table>

### Table 6-8  TDQUEUE Intra Partition Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDQUEUE</td>
<td>TDQUEUE</td>
<td>Specifies the 1- to 4-character name of a transient data queue.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual zone for description of the resource.</td>
</tr>
<tr>
<td>RECOVSTATUS</td>
<td>RECOVSTATUS</td>
<td>Specifies if the queue is logically recoverable or not.</td>
</tr>
<tr>
<td>TRANSID</td>
<td>TRANSID</td>
<td>Specifies the name of the transaction that is to be automatically initiated when the trigger level is reached.</td>
</tr>
</tbody>
</table>
### Table 6-9  MAPSET Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>MAPSET</td>
<td>Name of the MAPSET.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>Installation group name.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A general description of the MAPSET resource.</td>
</tr>
<tr>
<td>RESIDENT</td>
<td>RESIDENT</td>
<td>YES=preload. NO=load on first use.</td>
</tr>
<tr>
<td>swastatus</td>
<td>STATUS</td>
<td>Sets the status of the resource, to specify if it is available.</td>
</tr>
</tbody>
</table>
### Table 6-9 MAPSET Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>USAGE</td>
<td>Specifies the caching scheme to be used once the MAPSET is loaded.</td>
</tr>
<tr>
<td>FILENAME</td>
<td>----</td>
<td>Specifies the physical (binary) file name of the mapset, which is generated by the <code>tcxmapgen</code> utility (refer to section).</td>
</tr>
</tbody>
</table>

### Table 6-10 TYPETERM Mapping

<table>
<thead>
<tr>
<th>Field Name in ART FOR CICS</th>
<th>Resource Attribute in RDO/CSD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>TYPETERM</td>
<td>Name of the typeterm.</td>
</tr>
<tr>
<td>GROUP</td>
<td>GROUP</td>
<td>The group notion of CICS allowing a group of related resources to be declared and instantiated or not by a CICS system when starting.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>DESCRIPTION</td>
<td>A small textual zone for description of the resource.</td>
</tr>
<tr>
<td>color</td>
<td>COLOR</td>
<td>Designates extended color attributes.</td>
</tr>
<tr>
<td>defscreen column</td>
<td>DEFSCREEN (rows, columns)</td>
<td>Number of columns of the default screen size.</td>
</tr>
<tr>
<td>defscreen row</td>
<td>DEFSCREEN (rows, columns)</td>
<td>Number of rows of the default screen size.</td>
</tr>
<tr>
<td>hilight</td>
<td>HILIGHT</td>
<td>Indicates whether a terminal supports the highlight feature.</td>
</tr>
<tr>
<td>logonmsg</td>
<td>LOGONMSG</td>
<td>Indicates whether the &quot;Good Morning&quot; (CSGM) transaction is automatically started on the terminal.</td>
</tr>
<tr>
<td>outline</td>
<td>OUTLINE</td>
<td>Indicates whether the terminal supports field outlining.</td>
</tr>
<tr>
<td>swastatus</td>
<td>STATUS</td>
<td>Specifies the resource status (whether available).</td>
</tr>
<tr>
<td>Field Name in ART FOR CICS</td>
<td>Resource Attribute in RDO/CSD</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>uctrans</td>
<td>UCTRAN</td>
<td>Specify whether translate lowercase alphabetic characters to uppercase, or only translate the transaction ID from lowercase to uppercase, or not translate any characters.</td>
</tr>
<tr>
<td>userarealen</td>
<td>USERAREALEN</td>
<td>The terminal control table user area (TCTUA) area size for the terminal.</td>
</tr>
<tr>
<td>INTERCODE</td>
<td>----</td>
<td>Specifies which encoding type of inbound data is used.</td>
</tr>
<tr>
<td>EXTERCODE</td>
<td>----</td>
<td>Specifies which encoding type of outbound data is used.</td>
</tr>
<tr>
<td>SOSI</td>
<td>SOSI</td>
<td>Specifies whether mixed EBCDIC and double-byte character set (DBCS) is supported.</td>
</tr>
<tr>
<td>PROGSYMBOLS</td>
<td>PROGSYMBOLS</td>
<td>Specifies whether the programmed symbol (PS) facility is supported.</td>
</tr>
</tbody>
</table>
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**

1. 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 `ECT_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

2. ECI emulator supports both customer legacy ECI program and new created ECI client.
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_sample` 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
```
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 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;
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.
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 (`RSCMCIT`). 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. 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/.EXCIappProfile.

In DPL service, the EXEC CICS ASSIGN USERID() command returns:

- $USER environment variable in the EXEC interface
- USERID value passed to the DPL request function of the CALL interface

See ART Batch Runtime documentation for more details.
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 `COMAREA` 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 DFHXCRRM replaceable-module is not treated.

SRRCMIT/SRRBACK Functions

The SRRCMIT and SRRBACK functions are available. ATRCMIT and ATRBACK functions are not supported.

SRRCMI and SRRBACK functions must be coded as:

```sql
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

```plaintext
[SYSID]
APPLID={applid}
```
### Listing C-2  transactions.desc

<table>
<thead>
<tr>
<th>TRANSNAME;GROUPNAME; DESCRIPTION; PROGRAM</th>
<th>hardcode with DFHMIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECCI;SIMPDPL;pg for simpapp;DFHMIRS</td>
<td></td>
</tr>
</tbody>
</table>

### Listing C-3  program.desc

<table>
<thead>
<tr>
<th>PROGRAM;GROUP;DESCRIPTION;LANGUAGE;EXECKEY;STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TQuPSVR;SIMPDPL;pg for simpapp;COBOL; ;ENABLED;KIXD</td>
</tr>
</tbody>
</table>