ASYNCHRONOUS TASK MANAGER

&

EASY ASSEMBLY of COMPONENTS

TYMEAC™

TECHNICAL REFERENCE MANUAL

Cooperative Software Systems, Inc.
www.coopsoft.com

NOTE! Before using this information and the product it supports, please read the general information under “Notices” on page iv.

Second edition (April 1997)

This edition applies to Version 2.0 of the TYMEAC software product, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

CONTENTS

Table of Figures

NOTICES

The following paragraph does not apply to any country where such provisions are inconsistent with local law:

COOPERATIVE SOFTWARE SYSTEMS, INC. PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

CSS may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquires, in writing, to Cooperative Software Systems, Inc., Subject: Legal Department, (info@coopsoft.com).

TRADEMARKS AND SERVICE MARKS

The following terms in this publication are trademarks or service marks of companies as follows:

TYMEAC is a trademark of Cooperative Software Systems, Inc.

IBM, CICS are registered trademarks of International Business Machines Corporation.

PURPOSE

This technical reference manual is a single source of information for using the TYMEAC system across all operating systems. The “Installation and Customization” manual complements this manual for the one time purpose of installing TYMEAC.

This is the second edition of this manual. In this edition we separate the TYMEAC application transactions, (Chapter 3), from the Run Time transactions, (Chapter 5). In so doing, we segregate transactions for use mainly by application developers from transactions for use mainly by systems administrators.

CHAPTER 1 PRODUCT OVERVIEW

This chapter gives an overview of the TYMEAC product.

Definitions are the TYMEAC terms used throughout this manual. There is no separate glossary of computer, operating system, and CICS® terms.

The Directory is a quick reference to the programs, transactions, and copy code provided with the product.

The Structure is a graphic picture of the internal structure of TYMEAC used to complement the copy code loaded on your system during installation.

1.1 DEFINITIONS

SYNCHRONOUS REQUEST

The Request Broker returns the response to the task that initiated the transaction. In TYMEAC, this is a request_with_wait. The initiating task LINKs to the TYMEAC Request Broker to perform a Function and waits for the Function to complete.

ASYNCHRONOUS REQUEST

The Request Broker does not return the response to the task that initiated the transaction. In TYMEAC, this is a request_without_wait. The initiating task LINKs to the TYMEAC Request Broker to perform a Function and does not wait for the Function to complete. There is no time correlation between the request and the reply.

ASYNCHRONOUS TASK

There is no time correlation between the initiating task and the processing task. In CICS® systems, asynchronous task usually refers to task initiation by Transient Data trigger level or Automatic Transaction Initiation through Interval Control, but is not limited to any single method. TYMEAC uses the Interval Control START Command to initiate the asynchronous task.

SERVER APPLICATION PROGRAM

The user-written program that develops a need for TYMEAC processing either internally or through interfacing with the network, and LINKs to the TYMEAC Request Broker.

REQUEST BROKER

The TYMEAC program that separates a complex request from the Server Application Program into its component processes for scheduling as asynchronous tasks. For Synchronous Requests, it combines the output of the asynchronous component processes into a single message and passes that message back to the server application program.

SCHEDULER

The TYMEAC program that receives scheduling requests from the Request Broker or Queue Task program and physically schedules those requests to asynchronous tasks.

QUEUE TASK PROGRAM

The TYMEAC asynchronous program, logically attached to a Queue Area, that handles direct requests from the Scheduler, indirect requests by way of Wait Lists, timed suspensions when no work is pending, and manages the LINK to the Processing Application Program. For Synchronous Requests, the program passes the output of the user-written Processing Application Program back to the Request Broker. For Asynchronous Requests, the Queue Task Program adds the output of the user-written Processing Application Program to a temporary storage queue until all associated Queue Tasks' finish. Then it LINKs to the Scheduler for the Output Agent Queue.

PROCESSING APPLICATION PROGRAM

The user-written program that executes the method as a component process of the original complex request, or, uses an Application Programming Interface to pass the component into the network.

OUTPUT AGENT

The TYMEAC Queue Task Program that concatenates the output of the asynchronous component processes for Asynchronous Requests into an input message and LINKs to the Client Application Program for further user-determined processing.

CLIENT APPLICATION PROGRAM

The user-written program that receives the combined output generated by TYMEAC's asynchronous processing from the Output Agent. It either uses it internally, or, uses an Application Programming Interface to pass it to the network as the client.

MONITORING

The TYMEAC program that scans the internal TYMEAC tables looking for potential problems, compresses the Stall, Temporary Storage, and Request Tables, and frees resources. Section 5.5 Monitoring, documents this function in detail.

1.2 PROCESSING OVERVIEW

TYMEAC is first a CICS® asynchronous task manager. Asynchronous tasks are a means of isolating functions or performing common procedures outside the main task.

Furthermore, TYMEAC is a bridge to Object Technology. Objects, including third generation programs wrapped with TYMEAC, need a means for scheduling, monitoring, and LINKing together. TYMEAC makes assembling these components an easy task.

Evolving from these two base features is a myriad of features without limitation. The well managed, fault tolerant asynchronous tasks are unrestricted in communicating with any network protocol, database management system, queuing facility, or server complex. The gate is open to distributed computing and distributed components.

A Configuration File, (See Section 3.2, Configuration File Maintenance), coalesces the TYMEAC system. The System record describes TYMEAC Program and Mapset names, FCT names, etc. The Function and Queue records are user variables. Queues are individual processes, (including the user-written Processing Application Program that executes the method). A Function is simply a user-determined name and a list of Queues needed to complete the request:

f[name
Uses Queues A, B, and C]

Having the need to perform a given Function, which consists of various individual processes (Queues), the requester task LINKs to the TYMEAC Request Broker for the Function. TYMEAC schedules the individual Queues, collects the output messages therefrom, and presents this reply to the requesting task for Synchronous Requests, or, further schedules the Output Agent Task for Asynchronous Requests.

The following is a functional overview of TYMEAC processing for a single request.
For Non-Queue processing, the Request Broker uses the Non-Queue program name as the object of a CICS® LINK. The Request Broker passes the output from that LINK, if any, back to the Server Application Program. This is a Synchronous Request without TYMEAC Queue processing. See Section 7.1, Isolation and Section 3.2.3, Queue Maintenance for further discussions of this option.

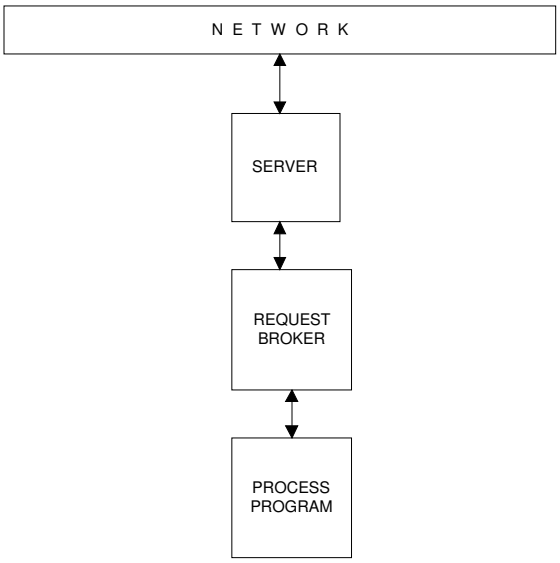


Figure 1.2.1 Non-Queue Processing.

For Queue_with_wait processing, (Synchronous Request): The Request Broker LINKs to the Scheduler for each Queue in the request. As each Queue finishes processing, its output, if any, moves to a common buffer. The Request Broker intersperses the collection of output with WAIT commands until either all Queue's finish or a time-out occurs. The Request Broker then passes the concatenated output from all Queue's back to the Server application program.

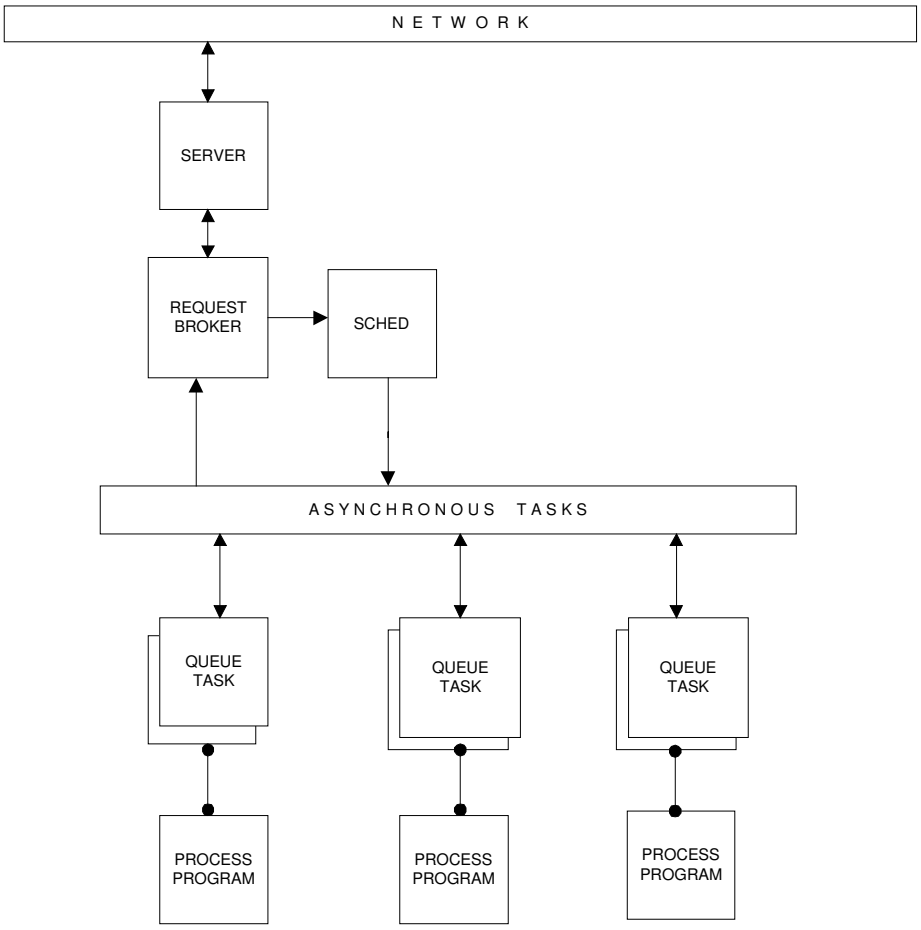


Figure 1.2.2 Queue Processing with Wait.

For Queue_without_wait processing, (Asynchronous Request): The Request Broker LINKs to the Scheduler to schedule each Queue in the request. The Request Broker also LINKs to the Scheduler to check the status of the Output Agent Queue, if any. Return is to the Sever application program. When the last Queue finishes processing, (the order of which is irrelevant), it schedules the Output Agent Task, (itself an asynchronous task). The user-written Client Application Program may then further process the concatenated output messages of the asynchronous tasks or interface with the network.

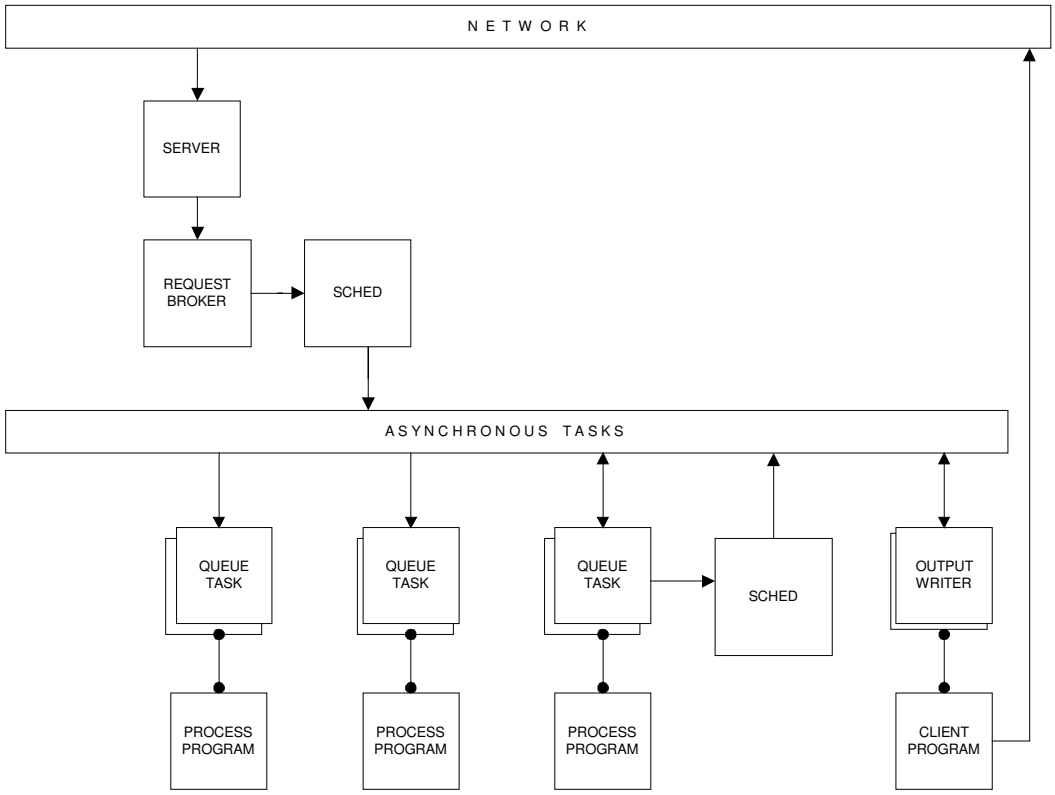


Figure 1.2.3 Queue Processing without Wait.

The following expounds the TYMEAC functions pictured above.

Request Broker:

The Request Broker receives requests from the Server Application Program, or any application program, and separates the request into its component processes (TYMEAC Queues). The Request Broker schedules each component process by LINKing to the Scheduler.

For Synchronous Requests, the Request Broker concatenates the output messages, if any, from each asynchronous process into a single message and passes it back to the originating program.

For Asynchronous Requests, the Request Broker also links to the Scheduler to check the status of the Output Agent Queue, if any. This status check is necessary to avoid a possible stall in processing when the last Queue Task finishes. After the last TYMEAC Queue Task schedules, the Request Broker returns control to the requester.

Scheduler:

The Scheduler searches the Main Table for the Queue name to schedule. It addresses the Area (Queue) Table for the Queue. It posts the individual task entry for a new request, or, if the Queue Task is busy processing another request, the request goes into a Wait List. When the load is heavy according to user thresholds, the Scheduler starts a new CICS® task.

The Scheduler also performs back-out processing for the Request Broker and Queue Task program (when scheduling the Output Agent Queue). Back-out processing occurs when a failure to schedule condition arises, or a time-out occurs (see Section 4.2 Request Broker/Scheduler Return Codes). The Scheduler nulls the task Anchor Point entry for the request, or, marks the Wait List entry 'reset', if not currently processing. See Section 7.2 Stall Table for a detailed description of back-out processing.

Queue Task:

The Queue Task program is a CICS® task, logically attached to a Queue Table. The program receives work either directly when the Scheduler posts its event control block (ECB) or indirectly from the Wait Lists.

The Queue Task processes each request either by using the Request Table entry for Synchronous Requests, or a temporary storage queue, (item 1), for Asynchronous Requests. It forms a Common Area and LINKs to the Processing Application Program. Output from the Processing Application Program moves to shared storage for Synchronous Requests or extends, as additional items, (2-n), the temporary storage queue for Asynchronous Requests. Additionally, for Asynchronous Requests, when all Queue Tasks' finish, the Queue Task LINKs to the Scheduler for the Output Agent Queue.

After each request the Queue Task checks MaxNumberRequest, (see Section 3.2.3 Configuration File, Queue Maintenance). When over this limit, the task restarts itself freeing accumulated resources.

The Queue Task checks the Wait List for the next request and when no work is pending the task ends or issues a WAIT command depending upon MaxWaitTime.

Output Agent:

The Output Agent program is a Queue Task program that only handles completed Asynchronous Requests. Posting and Wait List processing are identical to the Queue Task program. However, the input for the Client Application Program comes from the concatenated output of all the asynchronous Processing Application Programs for the request.

1.3 TRANSACTION*/PROGRAM/COPY CODE DIRECTORY*

The following is a list of program and transaction identification names that comprise TYMEAC.

PROGRAM	TXID	DESCRIPTION
NN00PGM1	NN00	Start up Front-End transaction. Use this to start up TYMEAC. It LINKs to the Start Up program and displays a message when complete.
NN99PGM1	NN99	Shut Down Front-end transaction. Use this to Shut Down TYMEAC. It LINKs to the Shut Down program and displays a message when complete.
NN01PGM1	NN01	Start Up program. Enter the transaction id or run the program at PLT time.
NN02PGM1	NN02	Shut Down program. Enter the transaction id, or run the program at PLT time.
NN03PGM1	NN03	Monitor program. Transaction Id is from the Configuration File. Entered on a 3270 screen, for testing.
NN04PGM1		Request Broker program. Verifies 'Function', and LINKs to the Scheduler for each Queue associated with the Function.
NN05PGM1		Scheduler program. Scans the area table for an available task to process the request. Posts ECB's, enters the request in the Wait List, and may start a new task to process the request.
NN06PGM1	NN06	Queue Task. This program receives work to do by scanning the Wait Lists or its ECB. It passes output back to the Request Broker program for Synchronous Requests, or puts the

		output into a temporary storage queue for Asynchronous Requests.
NN07PGM1	NN07	Output Agent Task. This is a Queue Task that concatenates the output of all the component processes of the Asynchronous Request and LINKs to the Client Application Program.
NN21PGM1	NN21	Configuration File maintenance. Add, change, and delete for the variables necessary to run the system.
NN21PGM2	NN2A	Same as NN21PGM1, above, however, this is the conversational version so that lower case is acceptable.
NN22PGM1	NN22	Log File display. This program displays a list of messages generated during processing and details for each selected entry.
NN23PGM1	NN23	TYMEAC Internal Table display. A formatted display of the main tables used in the system.
NN24PGM1	NN24	Disabled task update. This program resets Queue Tasks 'disabled' due to an abend, time-out, or other problem.
NN24PGM2	NN2B	Same as NN24PGM1, above, however, this is the conversational version so that lower case is acceptable.
NN25PGM1	NN25	Stall table display and update. This program displays the main storage Stall Table and allows deletion of individual entries.
NN26PGM1	NN26	Temporary Storage Table display and update. This program displays the main storage Temporary Storage Table and allows deletion of individual entries.
NN27PGM1	NN27	Request Table display and update. This program displays the main storage Request Table and allows deletion of individual entries.

NN28PGM1	NN28	Request Status Display. This program displays the status of Asynchronous Requests.
NN29PGM1	NN29	This program exports TYMEAC Tables to a file.
NN29PGM2	NN2C	Same as NN29PGM1, above, however, this is the conversational version so that lower case is acceptable.
NN30PGM1	NN30	This program imports TYMEAC Tables from a file built by NN29PGM1.
NN31PGM1	NN31	Live Queue Table update. This program displays a selected Queue and accepts updates to the elements and status changes to the Queue Tasks.
NN31PGM2	NN3A	Same as NN31PGM1, above, however, this is the conversational version so that lower case is acceptable.
NN61PGM1	NN61	Error Notification Transient Data Transaction.
NN62PGM1	NN62	Statistics Transient Data Transaction started at Shut Down.
NN63PGM1	NN63	On request Statistics File Writer Transaction
NN64PGM1		On request, individual statistics formatter designed for remote activation.
NNT1PGM1	NNT1	The single event testing program.
NNT2PGM1	NNT2	Same as NNT1PGM1, above, however, this is the conversational version so that lower case is acceptable.
NNT3PGM1	NNT3	The multiple event test program. It STARTs multiple tasks (NNT4) and monitor the progress thereof.
NNT4PGM1	NNT4	The multiple event test program asynchronous task..

NNT5PGM1		A test Processing Application Program for Non Queue Functions.
NNT6PGM1		A test Processing Application Program for processing Queue ‘AAAA’ data.
NNT6PGM2		A test Processing Application Program for processing Queue ‘BBBB’ data.
NNT6PGM3		A test Processing Application Program for processing Queue ‘CCCC’ data.
NNT6PGM4		A test Processing Application Program for processing Queue ‘EEEE’ data.
NNT7PGM1	NNT7	A test Client Application Program.
NNT8PGM1	NNT8	A test program for use as the destination object of an Output Agent that displays output on a terminal.

MAPSETS FOR THE ABOVE PROGRAMS:

NN21SET	NN22SET	NN23SET	NN24SET	NN25SET
NN26SET	NN27SET	NN28SET	NN29SET	NN30SET
NN31SET	NNT1SET	NNT3SET		

The following is a list of copy code, (COBOL and C), installed with TYMEAC.

NAME	DESCRIPTION
NN28	Common Area for Request Status Program.
NN64	Common Area for Remote Statistics Program.
NNAREA	Internal Queue storage layout.
NNCFG	Configuration File layout.
NNEXIM	Export / Import File layout.
NNFUNCT	Internal Function Table layout

NNGEN	Internal Number Generation Table layout.
NNINCOMM	Common Area for Request Broker.
NNLOG	Log File layout.
NNPASS	Common Area for test programs.
NNQTABLE	Internal Queue List Table, (from the Internal Function Table), layout.
NNREQCA	Internal Request Table layout.
NNRS	Data record layout passed by the Remote Statistics Program.
NNSHUTDW	Transient Data Statistics Record layout.
NNSHUTFL	Transient Data Statistics File layout.
NNSTALL	Internal Stall Table layout.
NNSTRG	Internal Main Table, (list of Queues), layout.
NNT1MAP	Map generated from NNT1SET Mapset for use with test programs.
NNT3MAP	Map generated from NNT3SET Mapset for use with test programs.
NNTCOMMA	Common Area for Processing Application Programs.
NNTDQ	Transient Data Error Notification Record layout.
NNTSQ	Main Temporary Storage Record layout.
NNTSREC	Internal temporary storage record, (used by Queue Tasks), layout.
NNTSTBL	Internal Temporary Storage Table layout
NNWAITL	Internal Wait List Table layout.

1.4 TYMEAC INTERNAL STRUCTURE

TYMEAC is a table driven system. All tables load into shared storage in the Dynamic Storage Area, (EDSA where applicable). TYMEAC uses one main temporary storage record as an Anchor Point. (Copy code is NNTSQ.) For Queue processing, TYMEAC generates the next available unique number from the Number Generation Table. (Copy code is NNGEN.)

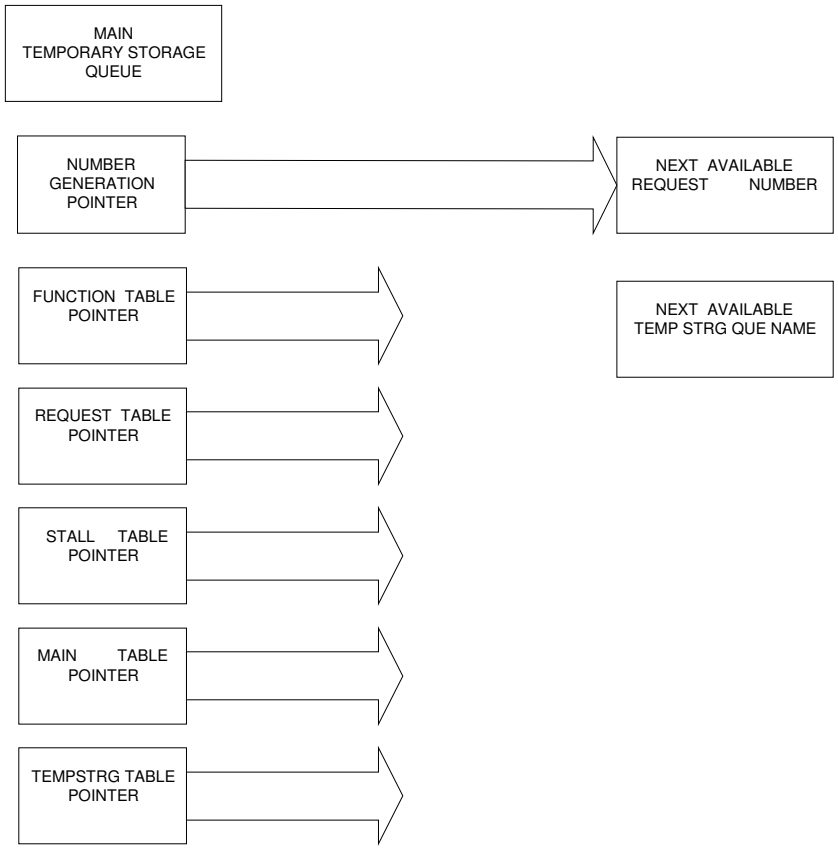


Figure 1.4.1 Main Temporary Storage Record and Number Generation Table.

The Function Table contains the list of all the Function names supported in the system and a Pointer to the Task Queue list for each Function. (Copy code is NNFUNCT.) The Task Queue list contains a list of the task Queue names associated with a Function. (Copy code is NNQTABLE.)

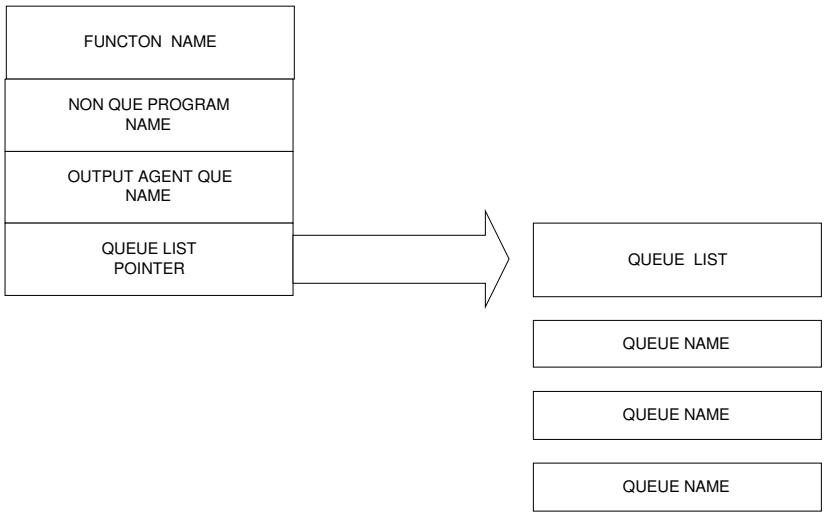


Figure 1.4.2 Function Table.

The Main Table contains a list of all the task Queues supported in the system and a Pointer to the Task Queue Area for each Queue name. (Copy code is NNSTRG.)

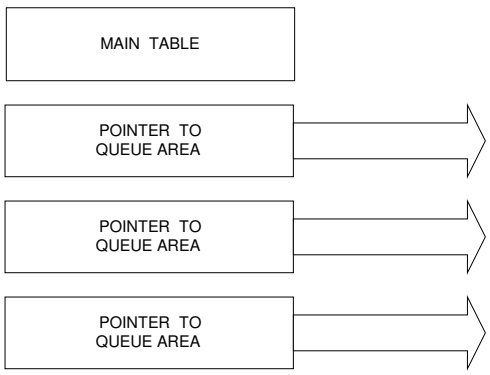


Figure 1.4.3 Main Table.

The Task Queue Area contains information common to a task Queue and an individual entry (Anchor Point), for each physical CICS® task. Part of the common storage is a Pointer to the Wait Lists for the task Queue. (Copy code is NNAREA.)

The Wait Lists contain Pointers to the Request Table entry for waiting tasks and the name of the temporary storage queue for non waiting tasks. (Copy code is NNWAITL.)

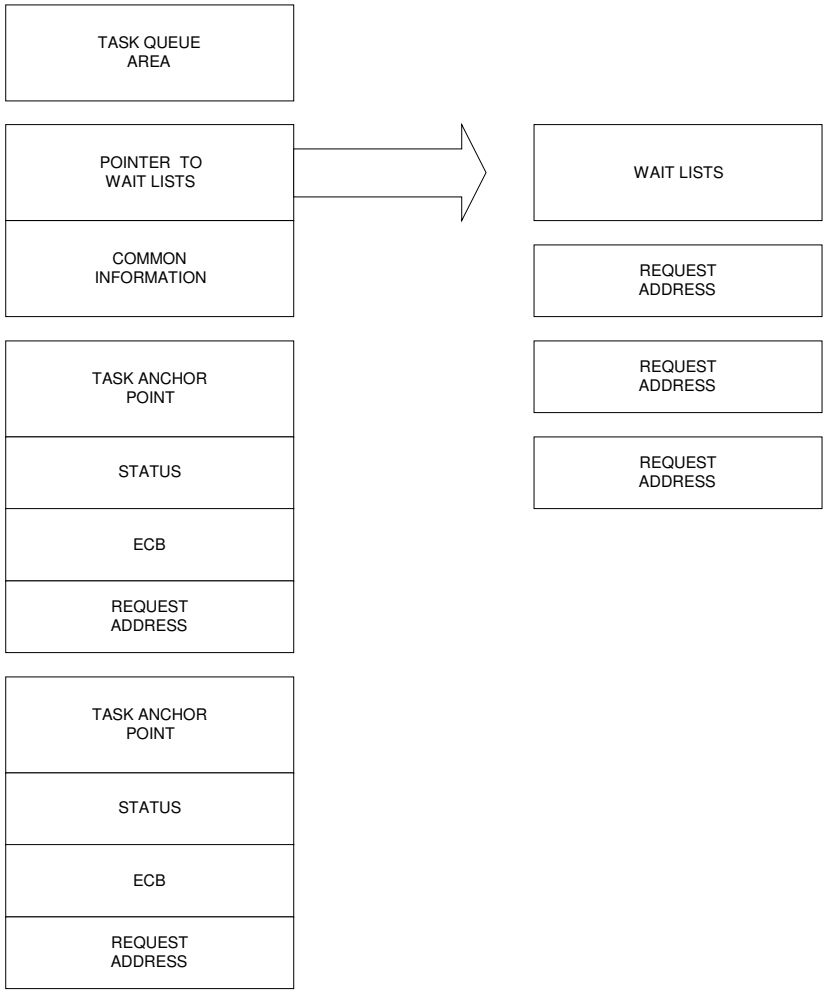


Figure 1.4.4 Task Queue (Area) Table with Wait Lists.

The Temporary Storage Table contains a list of all the temporary storage queues active in the system. TYMEAC uses Temporary storage to pass information to and among asynchronous tasks that are not waiting for completion, (Asynchronous Requests). (Copy code is NNTSTBL.) See Section 3.7 Temporary Storage Table Maintenance for details

about this table. See also Section 6.2 Tuning, (the use of temporary storage in TYMEAC).

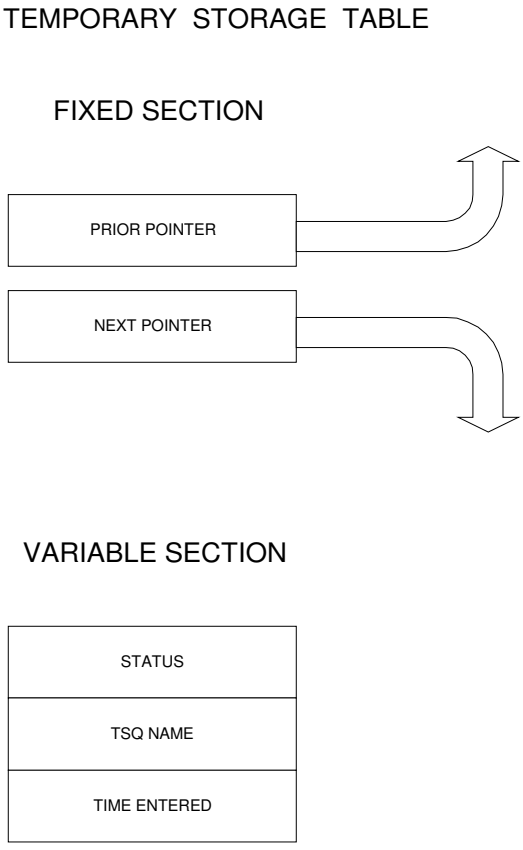


Figure 1.4.5 Temporary Storage Table.

The Stall Table contains a list of Temporary Storage Table entries that remain for longer than two cycles of the Monitor Task. The Monitor Transaction scans the Temporary Storage Table looking for problems, (Queue Tasks not responding, excessive processing or wait time), and places an entry in this table when 'there may be a problem'. The Request Broker also adds an entry to this table during back-out processing, when the back-out does not complete successfully. (Copy code is NNSTALL.) See Section 3.6 Stall Table Maintenance for details about this table. See also Section 7.2 Stall Table, used for problem determination.

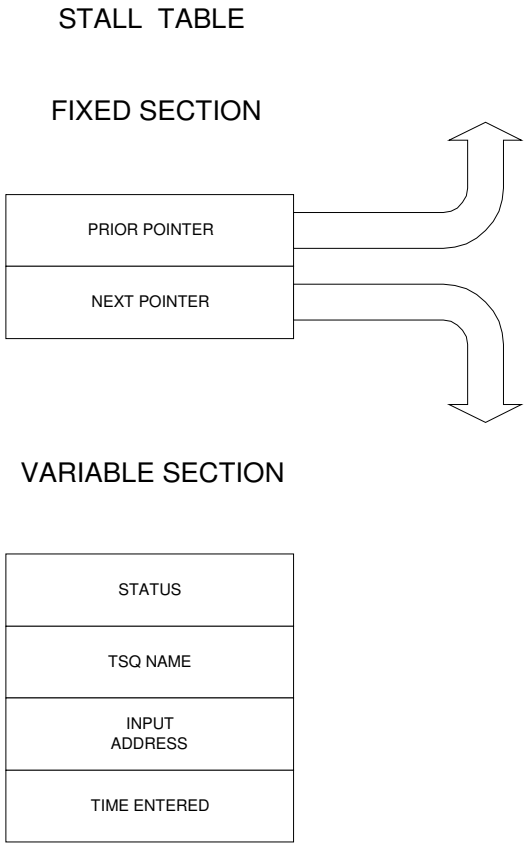


Figure 1.4.6 Stall Table.

The Request Table contains a list of all the active Synchronous Request components active in the system. Each entry contains parameters needed by the Queue Task including an input area and event control block (ECB) for posting. (Copy code is NNREQCA.) See Section 3.8 Request Table Maintenance for details about this table.

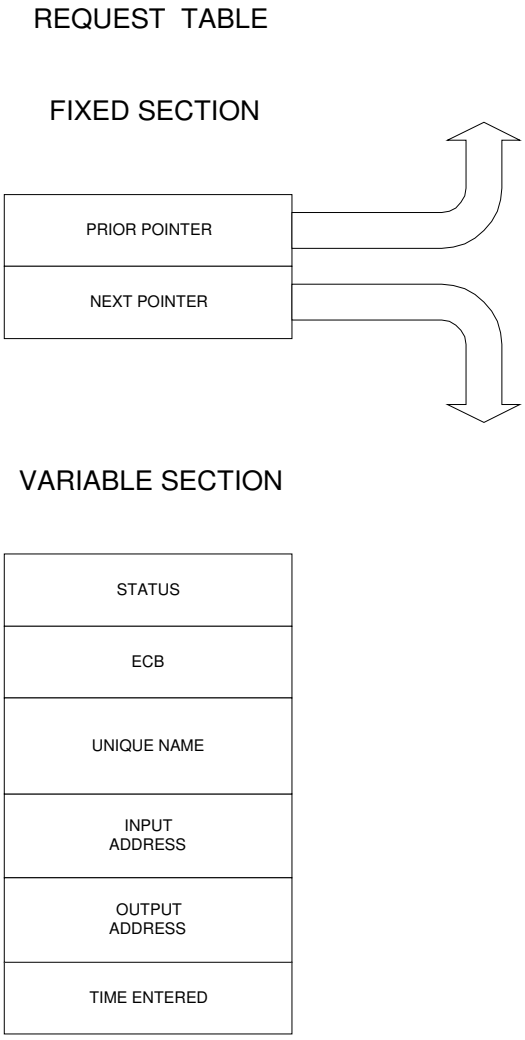


Figure 1.4.7 Request Table.

CHAPTER 2 SETTING UP THE ENVIRONMENT

This chapter points you to the individual “Installation and Customization” manual for each operating system, and details the use of the demonstration system.

The demonstration system is part of the Installation Verification Procedure and is also a learning guide. The demonstration programs, (source code loaded at installation and detailed in Section 4.3, samples), are examples of the simple procedures for using TYMEAC.

2.1 INSTALLATION AND CUSTOMIZATION

Installation and customization differ between operating systems. For detailed installation procedures, refer to the manual, “Installation and Customization”.

TYMEAC transactions, programs, copy code, and files begin with the prefix ‘NN’. However, the prefix as well as most of the naming structure is changeable by the customer. The only exceptions are the Configuration File, (NNCFG), the Configuration File Maintenance Mapset, (NN21SET), and the internal messages generated by individual TYMEAC programs. Chapter 3, TYMEAC Transactions, details screen messages as well as Log File messages.

The Configuration File controls the TYMEAC system. The FCT name of this file is ‘NNCFG’. The copy code for the file is NNCFG, however, this layout is for customer reference only since future releases may alter this layout. Section 3.2 Configuration File Maintenance, details the use of this file and is advisable reading before proceeding to the demonstration system.

The Request Broker program name is NN04PGM1. This name is hard-coded within the demonstration programs and is the customer’s responsibility to change when customizing the naming conventions.

A full list of TYMEAC transactions, programs and copy code is in Section 1.3, Transaction/Program/Copy Code Directory. This directory uses the default naming convention and is the customer’s responsibility to change when customizing naming conventions.

The Installation Verification Procedure uses the supplied demonstration system. This system is available for all as an aid to familiarization. Once users understand the principles and flow of TYMEAC processing, the supplied sample programs, (see Section 4.3, Samples), are available for cloning and enhancing to meet individual needs.

2.2 DEMONSTRATION SYSTEM

The Demonstration System consists of five Queues

- AAAA
- BBBB
- CCCC
- DDDD (Output Agent)
- EEEE

and eight Functions:

- FUNC0001 - FUNC0007
- FUNC0021

Appendix 2.2.B contains the screen images of the Configuration File needed to run the Demonstration System. This file loads during TYMEAC installation.

TYMEAC provides source code, in COBOL / C, for programs NNT1PGM1 - NNT8PGM1. The code is available for cloning by the user to suit particular needs.

TYMEAC provides source code, in 370 Assembler, for mapsets, NNT1SET and NNT3SET, for programs, NNT1PGM1 and NNT3PGM1. The code is available for altering by the user to suit particular needs.

Section 4.1, Usage, details the parameters for input to the Request Broker and Processing Application Program.

TYMEAC provides the Demonstration System to exhibit the flow of TYMEAC functionality pictured in Section 1.2, Product Overview. The supplied application programs take very simple input fields and form a very simple output message.

Program NNT1PGM1 is the single event, pseudo conversational program that visually expresses all the Functions in the Demonstration System, (Figure 2.2.1). Program NNT2PGM1 is the conversational version of this program to accommodate lower case.

Programs NNT3PGM1 and NNT4PGM1 are the multiple event programs used to place a 'load' on the CICS® system, (Figures 2.2.2 - 2.2.4).

Program NNT5PGM1 is the FUNC0001, Non-Queue Processing Application Program, (pictured in Section 1.2, Figure 1.2.1). The flow for Non-Queue processing is simply to LINK to the Processing Application Program with a Common Area, (copy code NNTCOMMA). This program takes the values for 'A', 'B', and 'C', (copy code NNPASS), and forms a concatenated output message, (see appendix 2.2.A).

Programs NNT6PGM1, 2, 3 are the Queue Task Processing Application Programs for Queue's 'AAAA', 'BBBB', 'CCCC', respectively, as pictured in Section 1.2, Figures 1.2.2 and 1.2.3. They take the values for 'A', 'B', and 'C', (copy code NNPASS), and form an output message, (see appendix 2.2.A).
Program NNT6PGM4 is the Queue Task Processing Application Program for Queue 'EEEE', the nested processing example. It takes the values for 'C' and forms an output

message. Then it LINKs to the Request Broker for 'FUNC0002', and appends the output thereof to the original output.

Program NNT7PGM1 is the Queue Task Processing Application Program for Queue 'DDDD', the Output Agent for Functions 5, 6, and 7. This program takes the output messages from Queue's 'AAAA', 'BBBB', and 'CCCC' and either sends the request to a terminal oriented task, (NNT8), or, writes the result to a File, (NNTEST).

Initially, this program begins with a RETURN. It is the user's responsibility to remove this RETURN and use the supplied code to write to the File or start task, NNT8, with data.

Program NNT8PGM1 retrieves input formed by program NNT7PGM1 and writes it to a terminal.

The TYMEAC Demonstration Functions are as follows:

Function 1 - Non Queue.

Function 2 - Queue AAAA, without Output Agent Queue .
Function 3 - Queue AAAA & BBBB, without Output Agent Queue .
Function 4 - Queue AAAA, BBBB & CCCC, without Output Agent Queue .

Function 5 - Queue AAAA, with Output Agent Queue 'DDDD".
Function 6 - Queue AAAA & BBBB, with Output Agent Queue 'DDDD'.
Function 7 - Queue AAAA, BBBB & CCCC, with Output Agent Queue 'DDDD'.

Function 21 Queue EEEE and AAAA, (nested processing example).

Functions 2, 3, and 4 are redundant. Functions 5, 6, and 7 are the same as Functions 2, 3, and 4 with the addition of an Output Agent Queue, (DDDD). Since the Output Agent Queue is only relevant to Asynchronous Request processing, Functions 5, 6, and 7 are sufficient to demonstrate both Synchronous and Asynchronous Request processing. Functions 2, 3, and 4 simply exemplify Functions without an Output Agent Queue. It is the user-written Sever Application Program that determines Synchronous or Asynchronous Request processing. TYMEAC Queues are unconcerned with the overall processing requirements; they just execute their method. TYMEAC Functions simply tie together the Queues and pass input to, and take output from, the Processing Application Programs. This separates the programs from the computer complex.

TYMEAC provides four transactions:

NNT1 -- Single event
NNT2 -- is the conversational version of the single event.
NNT3 -- Multiple events driver.
NNT4 -- is the multiple events asynchronous task.

Txid NNT1:

This transaction provides a single LINK to the Request Broker program for the Function requested. Functions 1 - 4 and 21 display the output on the terminal at field 'RESULT'. Functions 5 - 7 are for Asynchronous Requests. They display the Req_Id, returned by the Request Broker, as an internally generated message, and the asynchronous tasks write the output to the terminal, or to a File. Program

NNT7PGM1 handles either way. You may alter this program to do either process. Figure 2.2.1, below.

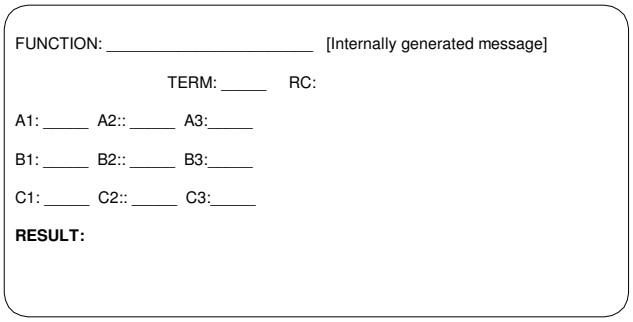


Figure 2.2.1 NNT1 Screen.

Enter txid=NNT1. The screen in Figure 2.2.1 displays.

Enter a Function name, a destination terminal, (optional), for FUNC0005-7, and numeric values for the A1-3, B1-3, C1-3 fields.

See Appendix 2.2.A for results for all the Functions.

Txid=NNT3:

This transaction provides a 'load' on the CICS® system. It simulates the NNT1 transaction being entered on multiple terminals for Functions 2-7 by using asynchronous tasks, (ATI).

Enter txid=NNT3. The screen in Figure 2.2.2, below, displays.

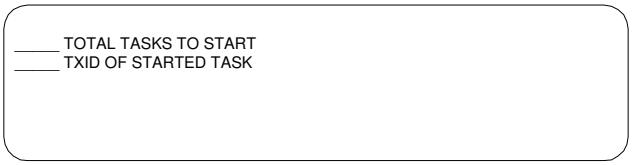


Figure 2.2.2 NNT3 Initial Screen.

Enter the number of tasks to start, from 0001 to 9999. The transaction STARTs as many CICS® tasks as entered here, therefore, one should carefully consider the CICS® region limits on MAX TASK, AMAX TASK, etc.

Optionally enter a TXID (NNT4 is the default).

<ENTER>

The message, in Figure 2.2.3, below, displays.

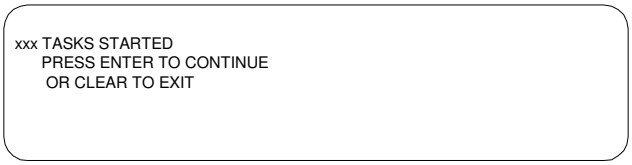


Figure 2.2.3 NNT3 Message Screen.

<ENTER>

The screen in Figure 2.2.4, below, displays.

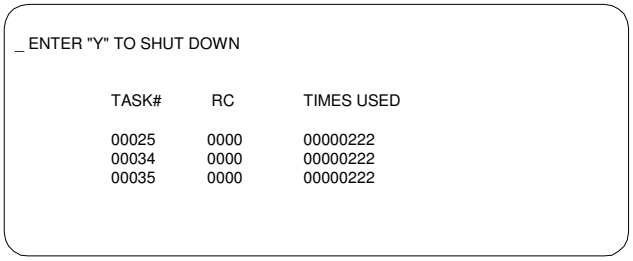


Figure 2.2.4 NNT3 Processing Screen.

TASK #	This is the CICS® task number of the last task to process.
RC	The Return Code from the Request Broker Program. If other than zero, no new task starts, otherwise, the task restarts itself with a one second delay.
USED	The number of times all Functions (02-07) completed with a return code of zero.

<ENTER> Refreshes the display.

<F7, F8> Backward, forward paging.

<CLEAR> Ends the CICS® transaction, but does not stop the processing.

On the first line, enter a 'Y' to stop processing. The message "IN SHUT DOWN MODE" displays. When all tasks complete, the message "ALL TASKS HAVE FINISHED" displays.

A single temporary storage queue, (Name='TMTMTMTM'), controls the action. When all else fails, the CICS® transaction CEBR is available to purge this queue and stop all processing.

The CLEAR Key frees the terminal for other work. Enter transaction NNT3 on this or any other (or multiple) terminals. As long as the temporary storage queue remains, the asynchronous processing continues.

APPENDIX 2.2.A

Values for the input:

A1 : 11 A2 : 22 A3 : 33
B1 : 44 B2 : 55 B3 : 66
C1 : 77 C2 : 88 C3 : 99

FOR FUNCTION 1:

RESULT: NON QUE TEST,RC=0000,A1(11)+A2(22)+A3(33)+B1(44)+B2(55)+B3(66)+ C1(77)+C2(88)+C3(99)=495,TERM=XXXX

FOR FUNCTION 2

RESULT: A---TEST,RC=0000,A1(11)+A2(22)+A3(33)=066,TERM=XXXX

FOR FUNCTION 3: *****

RESULT: B---TEST,RC=0000,B1(44)+B2(55)+B3(66)=165TERM=XXXX
A---TEST,A1(11)+A2(22)+A3(33)=066,TERM=XXXX

FOR FUNCTION 4: *****

RESULT: C---TEST,RC=0000,C1(77)+C2(88)+C3(99)=264,TERM=XXXX
A---TEST,A1(11)+A2(22)+A3(33)=066,TERM=XXXXB---TEST,B1(44)+B2(55)
B3(66)=165,TERM=XXXX

FOR FUNCTION 21:

RESULT: E---TEST,RC=0000,C1(77)+C2(88)+C3(99)=264,TERM=XXXX
A---TEST,RC=0000,A1(11)+A2(22)+A3(33)=066TERM=XXXX

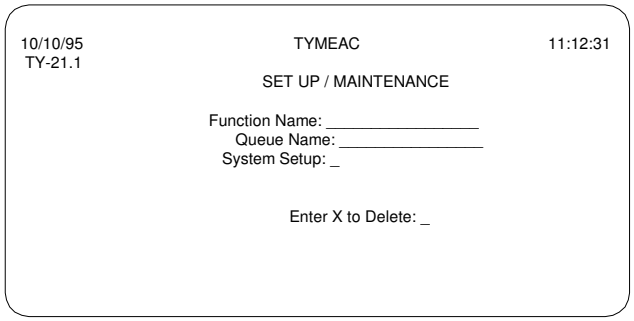
FOR FUNCTIONS 5-7:

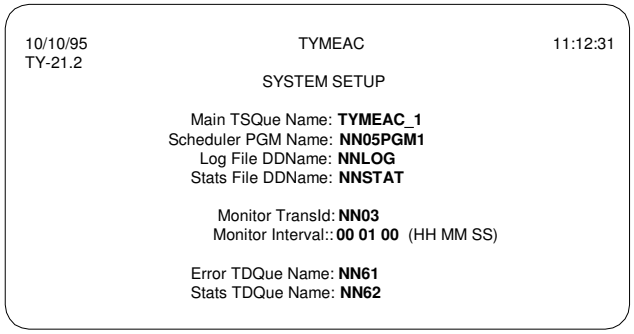
These are Functions without waiting for completion, (Asynchronous Requests). There is no RESULT. The message area contains the internal TYMEAC Request Identification (REQ_ID=), when the return code is zero.

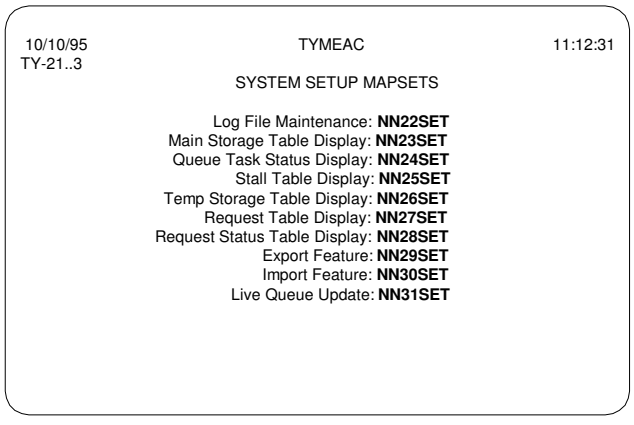
- The output message consists of the concatenated output of each Queue, in the order that each Queue finishes processing. The Request Broker schedules Queue's in the order loaded from the Configuration File. Execution variables determine the order of completion.

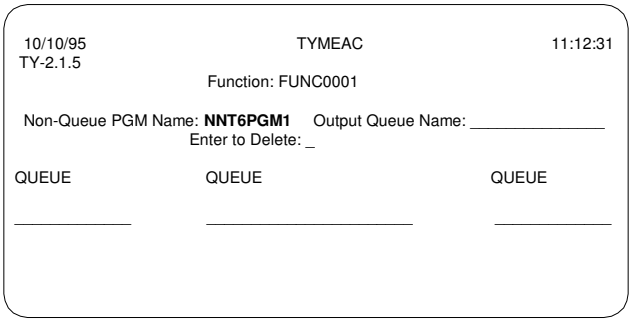
APPENDIX 2.2.B

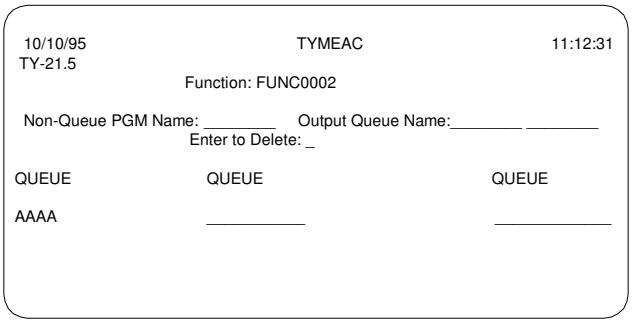
These are the screen images of the Configuration File loaded on your system at installation. Section 3.2, Configuration File Maintenance, gives the details of each screen and the data thereon.

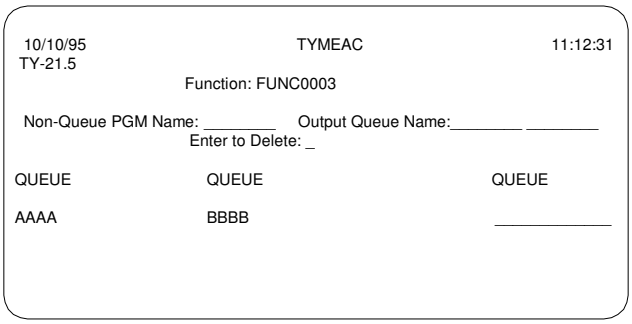


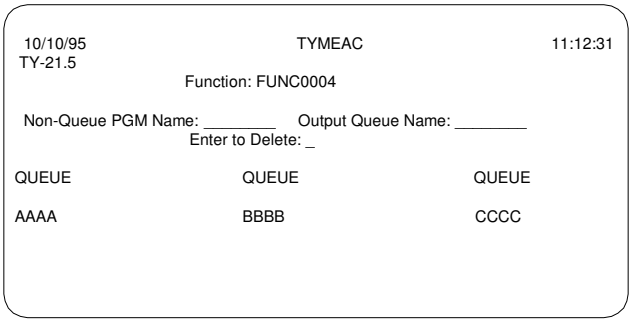


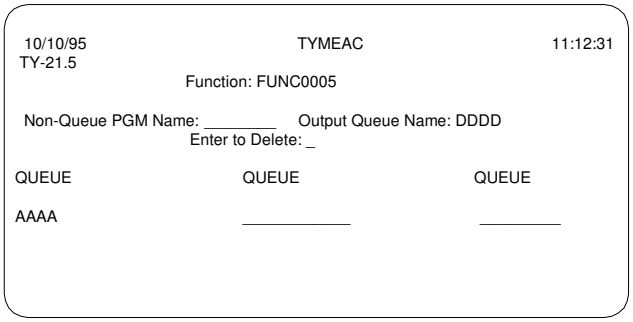


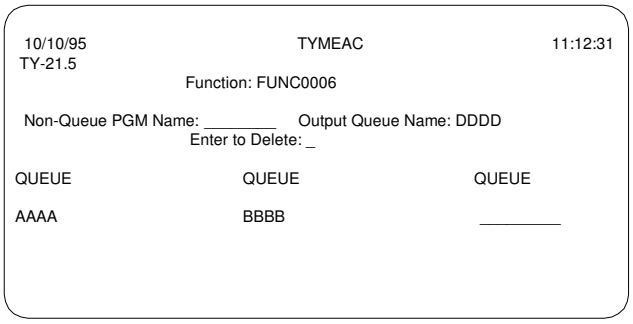

10/10/95 TYMEAC 11:12:31
TY-2.1.5
Function: FUNC0001
Non-Queue PGM Name: NNT6PGM1 Output Queue Name: _____
Enter to Delete: _
QUEUE QUEUE QUEUE
_____ _____ _____

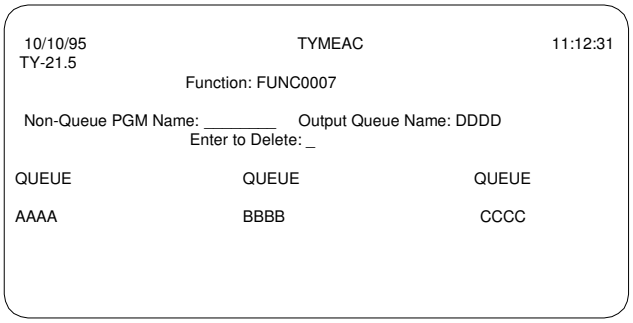

10/10/95 TYMEAC 11:12:31
TY-21.5
Function: FUNC0002
Non-Queue PGM Name: _____ Output Queue Name:_____ _____
Enter to Delete: _
QUEUE QUEUE QUEUE
AAAA _____ _____

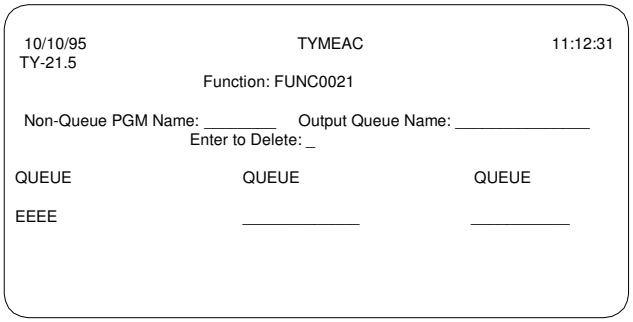

10/10/95 TYMEAC 11:12:31
TY-21.5
Function: FUNC0003
Non-Queue PGM Name: _____ Output Queue Name:_____ _____
Enter to Delete: _
QUEUE QUEUE QUEUE
AAAA BBBB _____

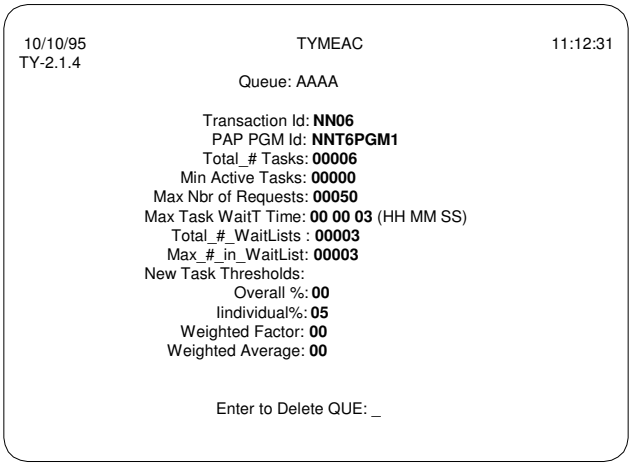

10/10/95 TYMEAC 11:12:31
TY-21.5
Function: FUNC0004
Non-Queue PGM Name: _____ Output Queue Name: _____
Enter to Delete: _
QUEUE QUEUE QUEUE
AAAA BBBB CCCC

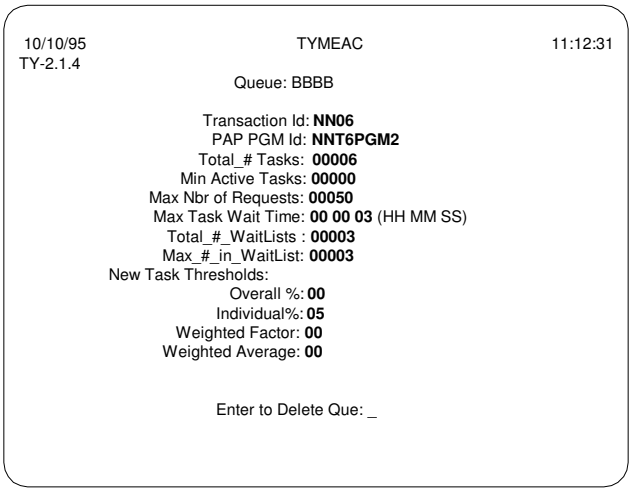


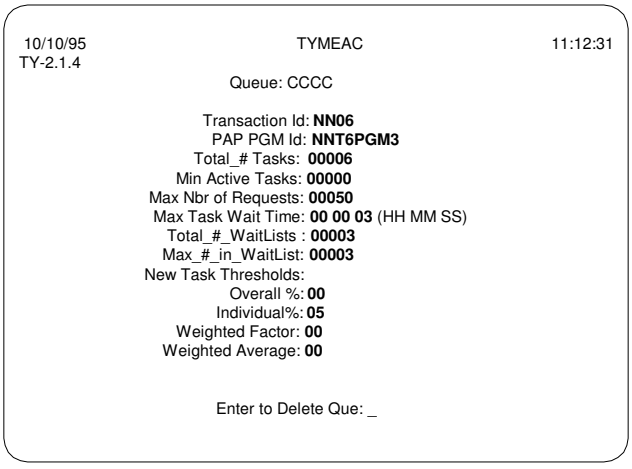


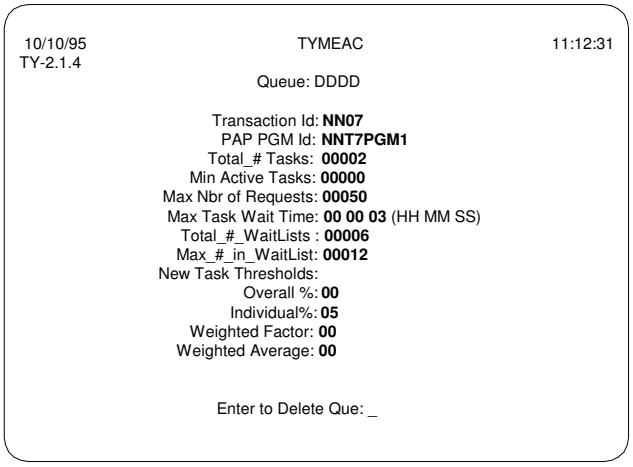


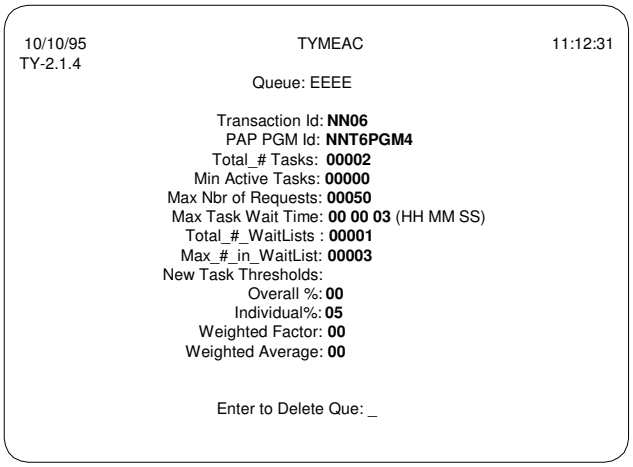



10/10/95 TYMEAC 11:12:31
TY-2.1.4
Queue: AAAA
Transaction Id: NN06
PAP PGM Id: NNT6PGM1
Total_# Tasks: 00006
Min Active Tasks: 00000
Max Nbr of Requests: 00050
Max Task WaitT Time: 00 00 03 (HH MM SS)
Total_#_WaitLists : 00003
Max_#_in_WaitList: 00003
New Task Thresholds:
Overall %: 00
Iindividual%: 05
Weighted Factor: 00
Weighted Average: 00
Enter to Delete QUE: _


10/10/95 TYMEAC 11:12:31
TY-2.1.4
Queue: BBBB
Transaction Id: NN06
PAP PGM Id: NNT6PGM2
Total_# Tasks: 00006
Min Active Tasks: 00000
Max Nbr of Requests: 00050
Max Task Wait Time: 00 00 03 (HH MM SS)
Total_#_WaitLists : 00003
Max_#_in_WaitList: 00003
New Task Thresholds:
Overall %: 00
Individual%: 05
Weighted Factor: 00
Weighted Average: 00
Enter to Delete Que: _


10/10/95 TYMEAC 11:12:31
TY-2.1.4
Queue: CCCC
Transaction Id: NN06
PAP PGM Id: NNT6PGM3
Total_# Tasks: 00006
Min Active Tasks: 00000
Max Nbr of Requests: 00050
Max Task Wait Time: 00 00 03 (HH MM SS)
Total_#_WaitLists : 00003
Max_#_in_WaitList: 00003
New Task Thresholds:
Overall %: 00
Individual%: 05
Weighted Factor: 00
Weighted Average: 00
Enter to Delete Que: _





CHAPTER 3 TYMEAC TRANSACTIONS

This chapter documents the application development transactions. However, additional transactions, (Chapter 5, Run Time), are available to complete the system.

3.1 SCREEN CROSS REFERENCE

Each 3270 screen in the TYMEAC system uses the same Function Keys:

CLEAR - Exit to CICS®.
F1 - Help.
F3 - Return (up one level).
Other Keys are screen dependent and are described on each screen.

Each 3270 screen in the TYMEAC system contains an identifier, 'TY-xx.y' in the first position on line three.

Where xx is the TYMEAC transaction and
y is the unique screen within that transaction.

TY-21.1 Configuration File Maintenance, Initial Display
TY-21.2 Configuration File Maintenance, System Setup
TY-21.3 Configuration File Maintenance, System Set up Mapsets
TY-21.4 Configuration File Maintenance, Queue Maintenance
TY-21.5 Configuration File Maintenance, Function Maintenance
TY-21.6 Configuration File Maintenance, Help

TY-22.1 Log File Display, Initial Display
TY-22.2 Log File Display, Browse
TY-22.3 Log File Display, Detail
TY-22.4 Log File Display, Help

TY-23.1 Main Table Display, Initial Display
TY-23.2 Main Table Display, Function Table
TY-23.3 Main Table Display, Queue Table
TY-23.4 Main Table Display, Queue Detail
TY-23.5 Main Table Display, Wait List Detail
TY-23.6 Main Table Display, Help

TY-24.1 Queue Status Update, Initial Display
TY-24.2 Queue Status Update, Help

TY-25.1 Stall Table, Initial Display
TY-25.2 Stall Table, Browse
TY-25.3 Stall Table, Detail
TY-25.4 Stall Table, Help

TY-26.1 Temporary Storage Table, Initial Display

TY-26.2 Temporary Storage Table, Browse
TY-26.3 Temporary Storage Table, Detail
TY-26.4 Temporary Storage Table, Help

TY-27.1 Request Table, Initial Display
TY-27.2 Request Table, Browse
TY-27.3 Request Table, Detail
TY-27.4 Request Table, Help

TY-28.1 Request Status, Initial Display
TY-28.2 Request Status, Help

TY-29.1 Export Tables, Initial Display
TY-29.2 Export Tables, View Groups
TY-29.3 Export Tables, View Groups, Detail
TY-29.4 Export Tables, Select Export Groups
TY-29.5 Export Tables, Export Functions
TY-29.6 Export Tables, Help

TY-30.1 Import Tables, Initial Display
TY-30.2 Import Tables, View Group Detail
TY-30.3 Import Tables, Help

TY-31.1 Live Queue Status, Initial Display
TY-31.2 Live Queue Status, Queue Elements
TY-31.3 Live Queue Status, Queue Tasks
TY-31.4 Live Queue Status, Help

3.2 CONFIGURATION FILE MAINTENANCE

Transaction id: NN21, Program: NN21PGM1.

This File contains all the variables necessary to control TYMEAC processing and all the Function and Queue names for user processing.

The File is a keyed file. TYMEAC maintains three types of records:

System:	Variables to control TYMEAC processing.
Functions:	User defined Functions.
Queues:	User defined Queues.

TYMEAC provides a second program, NN21PGM2, which is conversational and allows Function and Queue names in lower case. The transaction id is NN2A, program, NN21PGM2.

3.2.1 MENU

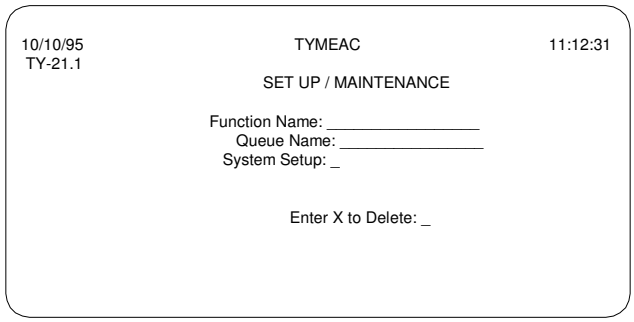


Figure 3.2.1 System Setup / Maintenance Menu.

System:
Tab to SYSTEM. Enter any character.

Queue processing:
Enter up to twenty-four characters. Repeat this step for each Queue name required. The Output Agent, for each non-waiting Function, (Asynchronous Request), is a Queue.

Function processing:

Enter up to twenty-four characters. Repeat this step for each Function name required.

Exit:
Enter any character to exit to or use the Clear Key.

3.2.2 SYSTEM SETUP

Page One

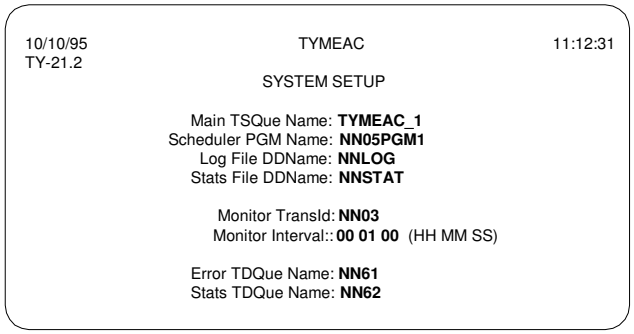


Figure 3.2.2.A System Setup.

MAIN TSQUEUE NAME: Eight characters. Required.
TYMEAC uses one Main Temporary Storage record as an Anchor Point for all main storage table Pointers. This is the name of that Queue. Enter any characters.

SCHEDULER PGM NAME: Eight characters. Required.
This is the name of the 'schedule asynchronous task' program.
Default, NN05PGM1.

LOG FILE DDNAME: Eight characters. Optional, but highly recommended.
This is the keyed file to which TYMEAC tasks write log messages.
Copy code for the File layout is NNLOG.
STATS FILE DDNAME: Eight characters. Optional, but highly recommended.
This is the file to which the TYMEAC Shut Down and On Request Programs write system statistics.

See Statistics TDQueue, below. Copy code for the File layout is NNSTAT.

EX/IM FILE DDNAME: Eight characters. Optional.

This is the keyed file for the Export/Import feature. TYMEAC writes Function and Queue records to this file for exporting to other images. Copy code for the File layout is NNEXIM.

MONITOR TRANSID: Four Characters. Required.
Queue tasks run asynchronously and can stall for a variety of reasons. The monitor tasks runs every monitor interval, below, looking for problems. This is optional for test systems. Default, NN03.

See Section 5.5 Monitoring, for further details.

MONITOR INTERVAL: Two, two, and two digits in the format Hours, Minutes, Seconds. Required.
This is the interval the Monitor Task runs. Reinitiating itself by a CICS® START for this interval. This is optional for test systems.

ERROR TDQUEUE NAME: Four characters. Optional.
When TYMEAC encounters an error, this is the Notification Transient Data queue sent a message. Default, NN61.

See Section 5.3 Notification, for further details.

STATS TDQUEUE NAME: Four characters. Optional.
This is the Transient Data queue to which TYMEAC sends system statistics messages On Request and at Shut Down. Default, NN62. Copy code for the message layout is NNSHUTDW.

See Section 5.4 Statistics, for further details.

Page Two

These are the CICS® Mapset names used by their respective transactions:

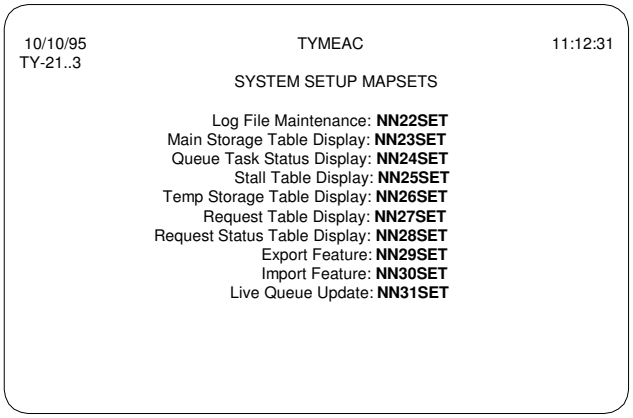


Figure 3.2.2.B System Mapsets.

LOG FILE MAINTENANCE:
Default NN22SET.
This is the name of the Log File Maintenance Transaction Mapset.

MAIN STORAGE DISPLAY:
Default NN23SET.
This is the name of the Main Storage Display Transaction Mapset.

QUEUE STATUS UPDATE:
Default NN24SET.
This is the name of the Queue Task Update Transaction Mapset.

STALL TABLE DISPLAY:
Default NN25SET.
This is the name of the Stall Table Display Transaction Mapset.

TEMPORARY STORAGE TABLE DISPLAY:
Default NN26SET.
This is the name of the Temporary Storage Display Transaction Mapset.

REQUEST TABLE DISPLAY:
Default NN27SET.
This is the name of the Request Table Display Transaction Mapset.

REQUEST STATUS DISPLAY:
Default NN28SET.
This is the name of the Request Status Display Transaction Mapset.

EXPORT FEATURE:
Default NN29SET.
This is the name of the Export feature Transaction Mapset.

IMPORT FEATURE:
Default NN30SET.
This is the name of the Import feature Transaction Mapset.

LIVE QUEUE UPDATE:
Default NN31SET.
This is the name of the Live Queue Update Transaction Mapset.

3.2.3 QUEUE MAINTENANCE

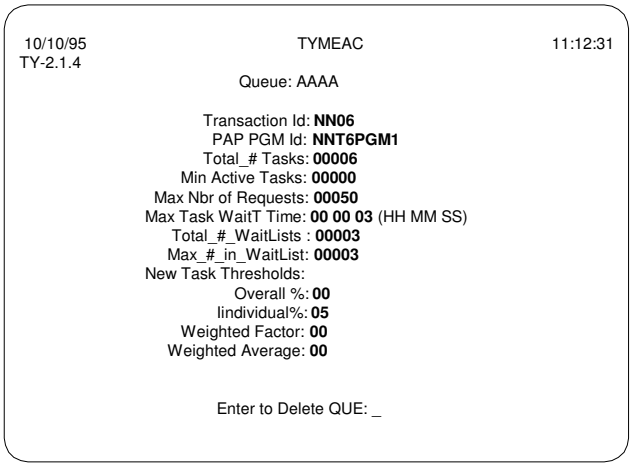


Figure 3.2.3 Queue Screen.

TRANSACTION ID: Four characters. Required.
The CICS® transaction identification for this Queue. This transaction id may be any value needed to control access, tuning, or security.

The CICS® PCT target program is the significant parameter. For Queues that are Queue Tasks, program NN06PGM1 is the target program. For Queues that are Output Agents, program NN07PGM1 is the target program.

The uncustomized demonstration system uses:
Txid=NN06, Pgm=NN06PGM1 for all Queue Tasks,
Txid=NN07, Pgm=NN07PGM1 for all Output Agent Tasks.

PAP PROGRAM ID: Eight characters. Required.
The name of the user-written Processing Application Program to which the TYMEAC Queue Task Program or Output Agent Program LINKs to process the user request. This is **NOT** the PCT target program associated with the transaction id, above.

TOTAL # TASKS: Five digits. Required.
The total number of Anchor Points in this Queue, for CICS® tasks, to process requests. This is the maximum number of tasks, for this Queue, which can be active at any time.

This number is significant for the CICS® Maximum Tasks and Maximum Active Tasks parameters in the System Initialization Table. This number is especially critical for those operating systems that need to pre-define the maximum number of Non Terminal Tasks in the image. These are those Non Terminal, asynchronous tasks. See Section 6.2, Tuning, under Maximum Tasks.

MIN ACTIVE TASKS: Five digits. Required, default, zero.
Used only during TYMEAC Start-Up -- the number of CICS® tasks STARTed, for this Queue, before any request for processing is present.

TYMEAC does not require a PLTPI start-up. One may place the transaction to Start Up anywhere in the system. See Section 5.1, Start Up.

This entry is for the latter. When the designer knows that a surge of requests comes in at one time, and it is more efficient to start several tasks at TYMEAC Start Up, or, when the TYMEAC Task performs a background function such as establishing a connection to another facility and then remaining suspended.

MAX NBR OF REQUESTS: Five digits. Required.
This is the maximum number of requests processed by a single CICS® task.

This is a balancing act. Command Level CICS® tasks require implicit GetMains. The user task never knows these were made and has no way of FreeMaining this storage. Part of the usefulness of TYMEAC is its ability to 'batch' requests. The Queue Task maintains a user-area between LINKs to the Processing Application Program so that control blocks needed by the Processing Application Program remain. However, this 'batching' may cause a lengthy storage chain. The balance is: for non storage intensive applications, use a high number, otherwise, use a low number. When this number is reached, the Queue Task issues a CICS® START for itself and returns to CICS®. This frees up the task's accumulated storage. The new task continues to process requests until it reaches the maximum number again, or, there is no more work. Queue Tasks are long running tasks. This is the means to control how long, taken together with Wait Time, below.

MAX TASK WAIT TIME: Two, two, and two digits in the format Hours, Minutes, Seconds. Optional, default, zero.
The time the Queue task waits when no work is pending. When zero, there is no WAIT.

This is the CICS® task suspend time. The CICS® task remains suspended, holding resources, until either its ECB is handposted or the wait time expires. When the wait time expires, the task ends. Section 4.1 Usage, under 'TCOM-WORK-PTR' and Section 6.2, Tuning, under Maximum Tasks, details the coordination of this field and Maximum Requests, above .

TOTAL # WAITLISTS: Five digits. Required, minimum one.
This is the total number of Wait Lists associated with this Queue. Waitlist 1, is priority 1. Waitlist 2, is priority 2, etc. The requesting task, when LINKing to the Request Broker, specifies the priority of the request in case no Queue Task is available to immediately process the request. The pending request goes into the Wait List corresponding to the priority, unless that Wait List is full. A no space available condition places the pending request into the next higher Wait List. This is an overflow. See the discussion of Wait Lists in Section 6.2, Tuning.

MAX # IN WAITLIST: Five digits. Required, minimum one.

This is the number of entries in each waitlist. This is a fixed number for all Wait Lists in this Queue. Multiply this number by the total number of Wait Lists for the total available entries for the Queue.

Total number and number in, are dependent upon the application. See the discussion of Wait Lists in Section 6.2, Tuning.

NEW TASK THRESHOLDS:

TYMEAC provides every opportunity to ensure the CICS® system is fully tunable. The fields below determine when to start a new CICS® task.

OVERALL %: Two digits. Required, default, zero.

INDIVIDUAL %: Two digits. Required, default, zero.

WEIGHTED FACTOR: Two digits. Required, default, zero.

WEIGHTED AVERAGE: Two digits. Required, default, zero.

TYMEAC STARTs a new task when a task Anchor Point is available for a CICS® task and the condition, described below, occurs:

1. No CICS® task is actively processing or about to process the Queue. Actively processing is a Queue Task Entry status of 'busy' or 'in link' (appl or schd). About to process is a Queue Task Entry status of 'started' or 'posted'.

2. The entry of a request into the Wait List results in an overflow. When the requesting Wait List is full, the request goes into the next available Wait List. This is an overflow.

3. When the percentage of active entries, to total Wait List entries, exceeds the **Overall Percent**. The total number of entries is the total Wait Lists multiplied by the number of entries in each list.

4. When not specifying a **Weighted Factor --** only considering the intended Wait List. When the percentage of busy Wait List entries to number in list entries exceeds the **Individual Percent**.

5. When specifying a **Weighted Factor** -- only considering the intended Wait List. When the percentage of busy Wait List entries to number in list entries, plus the **Weighted Factor** for that Wait List, exceeds the **Individual Percent**.

6. When specifying a **Weighted Average --** TYMEAC makes a detailed calculation considering all Wait Lists from the first to the current and the number of currently active tasks processing the Queue. When this calculation exceeds the **Weighted Average.**

TYMEAC ignores zero value variables.

Specifying 99 for Overall, Individual, and Weighted Average results in a new task for 1 and 2 above, and only when all Wait Lists are full.

EXAMPLES

One may not consider Wait Lists priority Wait Lists. You may use only two Wait Lists. One as a primary and a second in the event the primary is full.

If an overflow occurs, TYMEAC STARTs a new task. However, when using **Overall Percent**, you must factor the entries in subsequent Wait Lists into the calculation.

E.G. For two Wait Lists, when only considering the first -- 50% is the first Wait List completely full, (that is, all entries in the first Wait List are in use and no entry in the second Wait List is in use). 25% is the first Wait List half full. For this example, when one desires a new task to START when the primary Wait List exceeds half full, then the following values accomplish this purpose:

Overall %	**25**	|	**Overall %**	**00**
Individual	**00**	|	**Individual**	**50**
Factor	**00**	|	**Factor**	**00**

For priority Wait Lists, **Overall Percent**, is more significant. Consider a situation with four Wait Lists used as follows:

1. The hot request, process immediately.
4. The background request, defer until the load is light.
2. The normal request.
3. The overflow for Wait List two (2).

Irrespective of the number of CICS® tasks currently processing this Queue, when the overall load becomes thirty percent, start a new task. Specifying **Overall Percent** at **30** accomplishes this goal.

When the **Overall Percent** does not start a new task, then TYMEAC considers **Individual Percent**.

Individual Percent, without a **Weighted Factor** only considers the intended Wait List. Using the above example, when the load, overall, is greater than thirty percent or when the load on the desired Wait List is greater than fifty percent, then start a new task. The following accomplishes this goal.

Overall %: 30
Individual %: 50
Weighted Factor: 00

Priority Wait Lists are infinitely tunable. This is where the **Weighted Factor** can be significant.

The basis of a **Weighted Factor** is that a priority 1 is more significant than a priority 2, etc. TYMEAC multiplies the **Weighted Factor** by the reciprocal, (i.e., 1/x), of the Wait List number and rounds up the product. When there are three Wait Lists, with a **Weighted Factor** of five, then the **Weighted Factor** assigned to each list is:

Wait List 1 reciprocal is 1.000. 1.0*5=5
Wait List 2 reciprocal is 0.500. 0.5*5=3
Wait List 3 reciprocal is 0.333. 0.3*5=2

Individual Percent, with a **Weighted Factor** only considers the intended Wait List, plus the **Weighted Factor** for that Wait List. Using the above example, when the load, overall is greater than thirty percent or when the load on the desired Wait List is greater than fifty percent, plus a **Weighted Factor** of five (times the reciprocal of the Wait List number), then START a new task. The following accomplishes this goal.

Overall %: 30
Individual %: 50
Weighted Factor: 05

The **Weighted Factor** is totally dependent on the application and whether the Wait Lists are priority or non-priority. Therefore, specific examples would be misleading and tend to limit the usefulness of this parameter.

When the **Individual Percent** does not start a new task, then TYMEAC considers the **Weighted Average**.

Weighted Average:

The Scheduler multiplies the **Weighted Factor** by the reciprocal of the Wait List number, (when zero, the product is zero). The Scheduler adds this product to the percentage of busy entries, to number of entries, for each Wait List from one to the current (the intended Wait List). The Scheduler divides the sum of the percentages by the number of Wait Lists participating in the calculation, (ignoring those with zero busy). The Scheduler divides this percentage by the number of active tasks for the Queue, not to exceed the number of Wait Lists participating in the calculation. When the result is greater than **Weighted Average**, the Scheduler STARTs a new CICS® task.

The following is an example of the algorithm for **Weighted Average**.

There are 10 Wait Lists with 10 entries each.

The **Weighted Factor** is 5.
The new request went into Wait List 4.
There are 2 active tasks processing the Queue.

Wait List 1 has 3 busy entries.
Wait List 2 has 0 busy entries.
Wait List 3 has 5 busy entries.
Wait List 4 has 7 busy entries.

The **Weighted Factors** for the four lists are 5, 3, 2, 1 respectively.

	% busy	factor	total
Wait List 1 --	30%	+ 05	= 35%
Wait List 2 --	0		

Wait List 3 -- 50% + 02 = 52%
Wait List 4 -- 70% + 01 = 71%

158 / 3 (participating) = 51 / 2 (active tasks) = 25

When 25 is greater than **Weighted Average,** TYMEAC STARTs a new task.

The Live Queue Update transaction is available for experimentation. (Section 3.12.2).

DELETE QUE: To delete this record, enter a “D” and use F5.

3.2.4 FUNCTION MAINTENANCE

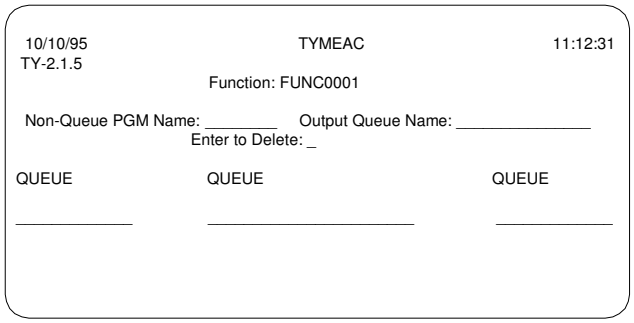


Figure 3.2.4 Function Screen.

NON-QUEUE PGM NAME: Twenty-four characters. Optional.
For compatibility with existing systems, and, for testing and isolation, (see Section 7, Isolation). The TYMEAC Request Broker forms a Common Area and LINKs to this program. The Request Broker returns the output message, if any, to the requester.

Not all legacy systems are going to change. Either because they are no longer comprehensible or there is no benefit in migrating them. TYMEAC wraps these programs with the Non Queue Option. This result may look entirely object oriented to developers and end users without the need for a conversion.

OUTPUT QUEUE NAME: Twenty-four characters. Optional.
The name of the Output Agent Queue to further process the Asynchronous Request.

For Queue processing without waiting for completion, (Asynchronous Request), the Request Broker schedules the Queues, below, and passes the parameters needed for processing in a temporary storage queue, item one. The Processing Application Program of each Queue may form an output message. The Queue Task adds this message to the same temporary storage queue, with item number incremented. When the last Queue Task Processing Application Program completes, the TYMEAC Queue Task program schedules the Queue named here. When not specified, The Queue Task discards this output.

This may be a process that sends this accumulated output to a destination or performs work depending on the success or failure of the asynchronous tasks for this Function.

This is a Queue. Define this Queue, as with all Queues, in Queue Maintenance, (Section 3.2.3).

DELETE:
Enter a 'D' and use F5 to delete this record.

QUEUE: Twenty-four characters. Required (except for Non-Queue processing).

The names (up to 165) of each Queue associated with this Function. The order of entry is the order of scheduling.

Define this Queue, as with all Queues, in Queue Maintenance, (Section 3.2.3).

APPENDIX 3.2.A

SCREEN ERROR MESSAGES

MESSAGE STRUCTURE

NN21 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 I ENTER FUNCTION, QUEUE, OR SYSTEM DATA

Reasons and solutions:

Enter a Function Name for Function processing, a Queue Name for Queue processing or any character in the System field for System processing.

010 S TEMP STORAGE ERROR

Reasons and solutions:

A WRITEQ TS Command failed.
The program uses temporary storage to hold information between screens. Check for CICS® system damage or lack of resources.

015 S GETMAIN FAILED

Reasons and solutions:

A GETMAIN Command failed.
Check for System damage or lack of resources.

020 S READ/UPDATE SYSTEM RECORD ERROR

Reasons and solutions:

A READ with UPDATE was issued for the SYSTEM record and the response was invalid.
Check for system damage or damage to the Configuration File.

025 S WRITE SYSTEM RECORD ERROR

A WRITE Command failed.
Check for system damage or damage to the Configuration File.

030 I OK

Reasons and solutions:

The last request processed successfully.

035 S ADD QUEUE HEADER RECORD ERROR

Reasons and solutions:

For a Configuration File without any Queue Records:
The Queue Record added successfully. The write for the header record failed. Check for system damage or lack of resources.

040 S READ SYSTEM RECORD ERROR

Reasons and solutions:

A READ for the System Record in the Configuration File failed.
Check for system damage or lack of resources.

045 S WRITE TEMP STORAGE ERROR

Reasons and solutions:

A WRITEQ TS Command failed.
The program uses temporary storage to hold information between screens. Check for system damage or lack of resources.

050 S REWRITE TEMP STORAGE ERROR

Reasons and solutions:

A WRITEQ TS with rewrite Command failed.
The program uses temporary storage to hold information between screens. Check for system damage or lack of resources.

055 S DELETE FUNCTION RECORD ERROR

Reasons and solutions:

A DELETE Command failed.
The program issued a Delete for the Function record.
Check for system damage or damage to the Configuration File.

060 S READ/UPDATE FUNCTION RECORD ERROR

Reasons and solutions:

A READ with Update Command for the Function record failed.
Check for system damage or damage to the Configuration File.

065 S ADD FUNCTION RECORD ERROR

Reasons and solutions:

A WRITE Command for the Function record failed.
Check for system damage or damage to the Configuration File.

070 S UPDATE FUNCTION RECORD ERROR

Reasons and solutions:

A REWRITE Command for the Function record failed.
Check for system damage or damage to the Configuration File.

075 S ADD FUNCTION HEADER RECORD ERROR

Reasons and solutions:

For a Configuration File without any Function Records:
The Function Record added successfully. The write for the header record failed. Check for system damage or damage to the Configuration File.

080 S UPDATE FUNCTION HEADER ERROR

Reasons and solutions:

The Function Record added successfully. The rewrite for the header record failed. Check for system damage or damage to the Configuration File.

085 S READ FUNCTION RECORD ERROR

Reasons and solutions:

A READ for the Function record failed. Check for system damage or damage to the Configuration File.

090 S WRITE TEMP STORAGE ERROR

Reasons and solutions:

See 045 S, above.

095 S REWRITE TEMP STORAGE ERROR

Reasons and solutions:

See 050 S, above.

100 S ERRORS DETECTED

Reasons and solutions:

Highlighted fields are in error. Correct and re-enter.

105 S THIS IS THE FIRST PAGE

Reasons and solutions:

Back paging was attempted on the first page.

110 S THIS IS THE LAST PAGE

Reasons and solutions:

Forward paging was attempted on the last page.

115 S DELETE QUEUE RECORD ERROR

Reasons and solutions:

A DELETE Command failed.
The program issued a Delete for the Queue record.
Check for system damage or damage to the Configuration File.

120 S READ/UPDATE QUEUE RECORD ERROR

Reasons and solutions:

A READ with Update for the Queue Record failed.
Check for system damage or damage to the Configuration File.

125 S ADD QUEUE RECORD ERROR

Reasons and solutions:

A WRITE for the Queue Record failed.
Check for system damage or damage to the Configuration File.

130 S UPDATE QUEUE RECORD ERROR

Reasons and solutions:

A REWRITE for the Queue Record failed.
Check for system damage or damage to the Configuration File.

135 S ADD QUEUE HEADER ERROR

Reasons and solutions:

For a Configuration File without any Queue Records:
The Queue Record added successfully. The write for the header record failed. Check for system damage or damage to the Configuration File.

140 S UPDATE QUEUE HEADER RECORD ERROR

Reasons and solutions:

The Queue Record added successfully. The rewrite for the header record failed. Check for system damage or damage to the Configuration File.

145 S READ QUEUE RECORD ERROR

Reasons and solutions:

A READ for the Queue Record failed.
Check for system damage or damage to the Configuration File.

150 S WRITE TEMP STORAGE ERROR

Reasons and solutions:

See 045 S, above.

155 S REWRITE TEMP STORAGE FAILED

Reasons and solutions:

See 050 S, above.

160 S INTERNAL PROGRAM ERROR

Reasons and solutions:

Call customer service.

200 S QUEUE NAMES EXIST IN FUNCTIONS BEGINNING WITH:

Reasons and solutions:

An attempt is being made to delete this Queue. However, the Queue

exists in Functions beginning with the one named. To delete a Queue, it may not exist in any Functions either as an asynchronous Queue or an Output Agent Queue.

205 S ERRORS DETECTED

Reasons and solutions:

See 100 S, above.

210 S ERRORS DETECTED ON PRIOR PAGE

Reasons and solutions:

The update cannot proceed since errors exist on the prior page.

215 S ERRORS DETECTED ON NEXT PAGE

Reasons and solutions:

The update cannot proceed since errors exist on the next page.

220 I NO ERRORS DETECTED

Reasons and solutions:

The update can proceed. No errors exist.

3.3 LOG FILE MAINTENANCE

Transaction id: NN22, Program: NN22PGM1

TYMEAC Programs write messages to the Log File when certain conditions occur. This program displays those messages.

Section 3.3.4 contains a full list of all log messages.

Appendix 3.3.A contains a full list of all screen messages.

3.3.1 LOG FILE MENU

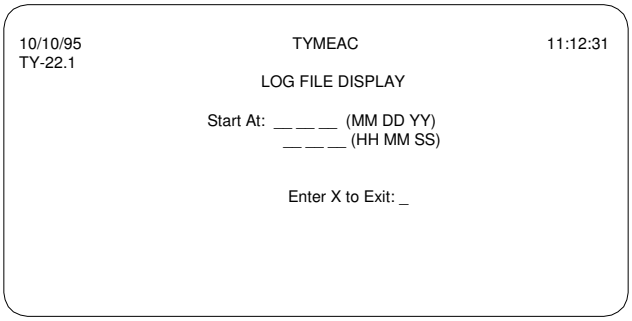


Figure 3.3.1 Log File Menu.

Start Date/Time:
Optionally enter a date and/or time to start the browse. The high order key is date/time, and, if entered, the display starts there.

TYMEAC maintains all dates with a four digit year, however, entering only the two digit year is sufficient here.

3.3.2 LOG FILE BROWSE

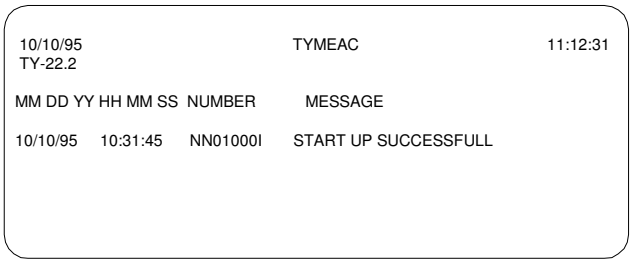


Figure 3.3.2 Log File Browse.

Date/Time:
Record key date/time entered.

Number:
The TYMEAC error number. See Section 3.3.4 for a listing.

Message:
A limited explanation of the message.

Place the cursor on a displayed line and <ENTER> for details.

3.3.3 LOG FILE DETAIL

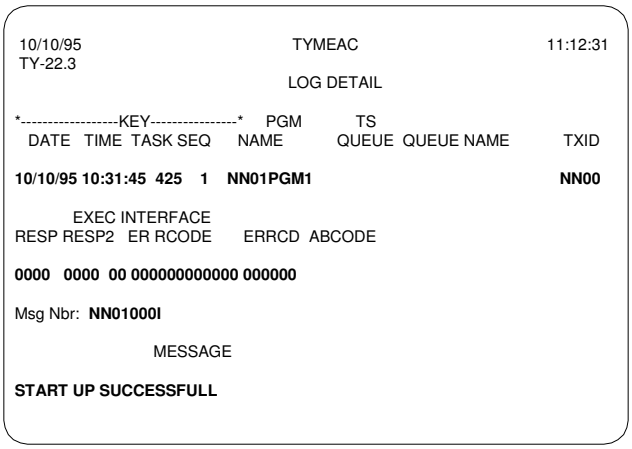


Figure 3.3.3 Log File Detail.

Key:
Full key of the record: Date, Time, CICS® task number, Sequence number.

Pgm Name:
The name of the program issuing the message.

TS Queue:
The name of the Temporary Storage Queue, (Req_Id), in decimal, if this is an Asynchronous Request.

Queue Name:
The TYMEAC Queue associated with the request, if any.

TXID:
The CICS® transaction id associated with the request.

EXEC Interface:
The EXEC Interface fields: Resp[2] (decimal), Errcd (hex), Rcode (hex), Errcd (hex), Abcode (character).

Msg Number: / Message:
The TYMEAC error number / The 79 character message.

APPENDIX 3.3.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN22 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 I NO RECORD FOUND FOR DATE/TIME

Reasons and solutions:

No record found equal to or greater than by date and/or time.

015 I INVALID CURSOR POSITION FOR DETAIL

Reasons and solutions:

Place the cursor on a data line and press ENTER.
There is no refresh of data on the screen. The transaction builds a temporary storage queue to support paging. To refresh the screen, use the Prior Key and start over.

020 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

025 I THIS IS THE LAST PAGE

This is the last page of data. To refresh the display, use the Prior Key and start over.

030 C NO RECORD FOUND

Reasons and solutions:

The selected record no longer exists. Possibly, another program deleted the record

035 I 60 PAGES IS THE MAXIMUM

Reasons and solutions:

Sixty pages is the maximum allowed for paging..

3.3.4 LOG FILE MESSAGES

As with all computer programs, similar errors occur at different points. In the case of TYMEAC, each error, whether similar to another or not, receives a unique message number. An example is the message “shut down detected”. The Queue Tasks examine the shut_down_byte after each CICS® command. When they find a Shut Down request, the programs send a unique message. The unique message number is especially relevant to TYMEAC personnel when customers encounter problems.

MESSAGE STRUCTURE

PPPP S NNN Where:

PPPP is the program issuing the message,

NN01 - Start Up
NN02 - Shut Down
NN03 - Monitor
NN04 - Request Broker
NN05 - Scheduler
NN06 - Queue Task
NN07 - Queue Task Output Agent
NN62 - Statistics TD Queue

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

The following pages describe log messages for each log number within an issuing program.

NN01 Start Up Program

000 I START UP SUCCESSFUL

Reasons and solutions:

TYMEAC Start Up was successful. The System is available for processing.

020 S CONFIGURATION FILE CORRUPTED

Reasons and solutions

The program issued a Start Browse Command on the Configuration File beginning at the first 'Function' record. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

030 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Configuration File for a 'Function' record. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

031 S NUMBER OF FUNCTIONS IN CFG FILE IS ZERO

Reasons and solutions:

There are no Function definitions in the configuration file. Therefore, no TYMEAC processing is possible.

Use the Configuration File Maintenance transaction to define Queues and Functions.

TYMEAC Initialization terminates.

040 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

The program issued a GetMain Command To build a main storage table. The Command failed. x.xxK was the amount of storage requested.

EIBRESP contains the return code.

TYMEAC obtains storage using the Flength option. TYMEAC does not use the UserDataKey option, therefore, the key of the program determines the key of the storage.

Short on Storage conditions at Start Up can only get worse. Therefore, consult with the Systems Administration staff for possible solutions.

TYMEAC Initialization terminates.

050 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Initialization File for a 'Function' record. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

060 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Configuration File for a 'Function' record. The record returned was not a 'Function' record.

The Configuration File, Function Header record, is in error. This record contains the total number of Functions. Start Up verifies this value. Check for system damage or damage to the Configuration File.

TYMEAC Initialization terminates.

070 S x.xxK GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

080 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Start Browse Command on the Configuration file beginning at the first 'Queue' record. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

090 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Configuration file for ‘Queue’ records. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

100 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Initialization File for a ‘Queue’ record. The record returned was not a ‘Queue’ record.

The Configuration File, Queue Header Record, is in error. This record contains the total number of Queues. Start Up verifies this value. Check for system damage or damage to the Configuration File.

TYMEAC Initialization terminates.

110 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

120 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

The program issued a Read Next Command on the Configuration file for ‘Queue’ records. The Command failed.

EIBRESP contains the CICS® return code. Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

130 S CONFIGURATION FILE CORRUPTED

Reasons and solutions:

See 100 S, above.

140 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

150 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

170 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

180 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

190 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

See 040 S, above.

200 S MAIN TSQ WRITE FAILED

Reasons and solutions:

The program issued a WriteQ TS for the main Temporary Storage record from which all TYMEAC storage chains. The Command failed.

EIBRESP contains the return code.
Make the alterations suggested by the above return code.

TYMEAC Initialization terminates.

210 C START FOR QUEUE TASK FAILED, ENTRY DISABLED=xxx

Reasons and solutions:

The program issued a CICS® START for a task in accordance with 'Minimum Number of Tasks'. The start failed.

xxx identifies the Queue Name and task Anchor Point

EIBRESP contains the return code.
Make the alterations suggested by the above return code.

NN02 Shut Down Program

010 S READ MAIN TSQ FAILED, SHUT DOWN ABORTED

Reasons and solutions

The program issued a ReadQ TS for the TYMEAC main TSQ record. The Command failed.

EIBRESP contains the return code. Make the alterations suggested by the above return code. This simply may be the result of TYMEAC not being active.

TYMEAC Shut Down terminates.

020 S REWRITE MAIN TSQ FAILED, SHUT DOWN ABORTED

Reasons and solutions

The program issued a rewrite for the TYMEAC main TSQ record, after updating the Shut Down byte. The Command failed.

EIBRESP contains the return code. Make the alterations suggested by the above return code.

TYMEAC Shut Down terminates.

025 S INVALID MAIN STORAGE ID, SHUT DOWN ABORTED

Reasons and solutions

All TYMEAC shared storage starts with a character marker (Id). The marker for the main storage area is invalid.

1. TYMEAC was not Shut Down before CICS® was Shut Down leaving the main temporary storage record. As a result of a warm or emergency restart the addresses within the main temporary storage record are now invalid.

2. TYMEAC storage is in error.

TYMEAC deletes the main temporary storage record and Shut Down terminates.

Depending on the operating system environment an invalid address may result in an ASRA abend or a TRAP. Manually purge the main temporary storage record with the CEBR transaction, if not already purged.

030 I STATS TD QUEUE ERROR, SHUT DOWN CONTINUING

Reasons and solutions

The program issued a WriteQ TD for the Statistics Transient Data Queue. The Command failed.

EIBRESP contains the return code. Make the alterations suggested by the above return code.

TYMEAC Shut Down continues.

040 I SHUT DOWN INCOMPLETE, TASKS STILL ACTIVE

Reasons and solutions

Queue Task Anchor Points are active or about to be active. TYMEAC cannot free shared storage until all Queue Tasks end.

This is not an error, especially in a testing environment. If one wishes to Shut Down the system and then restart after making changes to the Configuration File, this transaction is the way to go. However, if Queue Tasks are 'hanging' or some other problem exists, then refer to the Live Queue Update Transaction, (Section 3.12).

TYMEAC Shut Down continues.

050 I SHUT DOWN SUCCESSFUL, TASKS FINISHED, STRG FREED

Reasons and solutions

TYMEAC Shut Down is complete.
No Queue Tasks are active.
TYMEAC FreeMained all shared storage.
TYMEAC purged the main TSQueue record.

NN03 Monitor Program

010 I LONG STALL TABLE ENTRY=xxx

Reasons and solutions

The program encountered an entry in the stall table that was there for more than three passes of the Monitor Task.

TYMEAC adds entries to the Stall Table when a 'maybe' condition arises. The Monitor Task removes these entries when it cleans up any problem or the condition resolves itself. Any entry left in the Stall Table may require intervention. xxx identifies the Stall Table Entry name.

020 I QUEUE qqqq WAIT TIME EXCEED BY xxxx TIMES

Reasons and solutions

The Monitor Task found the wait time for this task (qqqq) exceeded the maximum requested by (xxxx) times.

This is an information message. This event may happen when the system is at Maximum Active Tasks. Even though the wait time expired, the task is non-dispatchable. When the number of times is greater than 999, TYMEAC marks the task 'canceled'.

030 I ALL TASKS ARE DISABLED, QUEUE=xxx

Reasons and solutions

The Monitor Task scans all Queue Tasks and for this Queue, all tasks are 'disabled'. xxx identifies the Queue.

Use the Queue Task Status Update Transaction to reset the disabled entries, (Section 3.5).

035 I xxx DISABLED ANCHOR POINTS IN QUEUE=yyy

Reasons and solutions

The Monitor Task scans all Queue Tasks and for this Queue, xxx tasks are 'disabled'. yyy identifies the Queue.

Use the Queue Task Status Update Transaction to reset the disabled entries, (Section 3.5).

040 I XXX WAITLIST ENTRIES WERE KILLED

Reasons and solutions

The Monitor Task scans all Queue Tasks and for this Queue, no task is able to process a request. TYMEAC flushes the Synchronous Requests in the Wait Lists. This message usually accompanies message 030 I, above.

050 I NOTIFICATION SENT

Reasons and solutions

The Monitor Task sent a Transient Data notification message to the TD Error Queue.

060 C x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions:

The program issued a GetMain Command to build a new Stall Table. The Command failed. x.x.K was the amount of storage requested.

EIBRESP contains the return code.

TYMEAC obtains storage using the Flength option. TYMEAC does not use the UserDataKey option, therefore, the key of the program determines the key of the storage.

The Monitor Tasks continues processing.

070 I NEW STALL TABLE ENTRY

Reasons and solutions

The program encountered an entry in the Temporary Storage Table that was entered more than two cycles before. The Monitor added a new Stall Table entry. See also 010 I, above.

This is an information message. The cycles of the Monitor Task depend on the interval set in System Set Up, (Section 3.2.2). If the Log File fills with this message and no other problem is evident, then perhaps the interval is too short. See also Section 5.5 Monitoring.

NN04 Request Broker Program

005 C SCHD FAILED, QUE= XXX, RC=YYY

Reasons and solutions

The Queue Task did not schedule. XXX is the name of the Queue. YYY is the return code from the Scheduler.

The Scheduler return code returns to the calling program. See Section 4.2, Scheduler Return Codes.

010 C SCHD FAILED, QUE= XXX, RC=YYY

Reasons and solutions

This message is the same as 050 C, above, however, this is for non-waiting requests.

900 S ABEND EXIT ENTERED FOR ABCODE=xxxx

Reasons and solutions

An (xxxx) Abend terminates Asynchronous Request processing. Make the correction suggested by the abend code.

NN06 Queue Task Program

010 S INTERNAL TYMEAC ERROR

Reasons and solutions

The START for all Queue Tasks is with Data. Either the CICS® Start Code is not 'SD' or the Retrieve Command failed.

EIBRESP contains the return code. Make the alterations indicated by the return code.

015 S INTERNAL TYMEAC ERROR

Reasons and solutions

The Start for all Queue Tasks is with Data. The data contains the Area name (Queue) and Area number (Queue Task Anchor Point). The Area name is invalid. Main storage is in error.

020 S INTERNAL TYMEAC ERROR

Reasons and solutions

The Start for all Queue Tasks is with Data. The data contains the Area name (Queue) and Area number (Queue Task Anchor Point). The Area number is invalid. Main Storage is in error.

025 I SHUT DOWN DETECTED

Reasons and solutions

The Shut Down Transaction marked the Area System_Going_Down. Further processing terminates.

030 C TASK ALREADY ACTIVE DETECTED AT START UP

Reasons and solutions

The Queue Task starting found the Queue Area storage in a state that another Queue Task is already actively processing requests.

This may happen as a result of a TYMEAC task that was 'reset' from Disabled to Available, without the CICS® task being purged. The Reset Task starts a new task, then the original task resumes execution, and this conflict results.

The new task terminates.

The request for the new task goes in the Wait List. The original task searches the Wait List for work to do and should process the request normally. However, when this message persists, then storage may be in error.

035 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

The program issued a Get Main Command to build a main storage table. The Command failed. x.x.K was the amount of storage requested.

EIBRESP contains the return code.

TYMEAC obtains storage using the Flength option. TYMEAC does not use the UserDataKey option, therefore, the key of the program determines the key of the storage.

040 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

045 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 035 I above.

050 S INVALID RESP FROM LINK, SEE EIBRESP

Reasons and solutions

The Queue Task issued a CICS® LINK to the application program. The response was other than zero. See EIBRESP.

055 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 035 I above.

060 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

065 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 035 I above.

070 S INVALID READQ TS, SEE EIBRESP

Reasons and solutions

Queue tasks, processing Asynchronous Requests, use a temporary storage queue to pass input parameters and hold output. The read for this queue failed.

See EIBRESP for the reason. When other than a NotFound, then storage may be in error.

NotFound is not necessarily an error. The Request Broker deletes the temporary storage queue during back-out processing. Since the Asynchronous Request failed scheduling, the requester receives immediate notification. Look in the Log File for a Request Broker message with the same TS QUEUE name.

075 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 035 I above.

080 S INVALID READQ TS, SEE EIBRESP

Reasons and solutions

See 070 S above.

085 S INVALID QUEUE NAME IN TS RECORD

Reasons and solutions

Queue tasks processing Asynchronous Requests use a temporary storage queue to pass input parameters and hold output. The queue contains the name of each Queue. Either the temporary storage record is in error or the Main Table, containing the Queue names, is in error.

090 S INVALID RESP FROM LINK, SEE EIBRESP

Reasons and solutions

See s 050 above.

095 S INVALID WRITEQ TS RESP, SEE EIBRESP

Reasons and solutions

Queue tasks processing Asynchronous Requests use a temporary storage queue to pass input parameters and hold output. The Queue Task program updates this queue when each Processing Application Program finishes. The rewrite failed. EIBRESP contains the return code.

100 S INVALID WRITEQ TS RESP, SEE EIBRESP

Reasons and solutions

See 095 S above.

105 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

110 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

115 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

120 S INVALID POST RC, WAIT TIME IN ERROR

Reasons and solutions

The Queue Task issued a CICS® POST command using the interval stored in the Queue Header data. This interval loads from the Configuration File at Start Up.

If the Header data is not in error, than the Configuration File Maintenance Transaction improperly accepted the interval.

125 I SHUT DOWN DETECTED

Reasons and solutions

See 025 I above.

130 S INVALID WAIT RC

Reasons and solutions

The Queue Task issued a WAIT command and the command failed. See EIBRESP.

135 S INVALID READQ TS FOR UPDATE

Reasons and solutions

See S 070 above.

140 S INVALID REWRITE TSQ

Reasons and solutions

See S 095 above.

145 S SCHD LINK FAILED, SEE EIBRESP FOR DETAILS

Reasons and solutions

All Queue Tasks finished for the Function.

The last Queue Task to finish LINKs to the Scheduler to schedule the Output Agent task. The schedule failed for the reason contained in the full message. See Section 4.2, Scheduler Return Codes.

150 S SCHD LINK FAILED, SEE EIBRESP FOR DETAILS

Reasons and solutions

See 145 S, above.

900 S ABEND - SEE ABCODE FOR REASON

Reasons and solutions

Execution entered the Queue Task Abend Exit Routine as a result of an application ABEND.

NN07 Queue Task Output Agent Program

005 S SHUT DOWN DETECTED

Reasons and solutions

The Shut Down Transaction marked the Area System_Going_Down. Further processing terminates.

010 C TASK ALREADY ACTIVE DETECTED AT START UP

Reasons and solutions

The Queue Task starting found the Queue Area storage in a state that another Queue Task is already actively processing requests.

This may happen as a result of a TYMEAC task that was 'reset' from Disabled to Available, without the CICS® task being purged. The Request Broker starts a new task, the original task resumes execution, and this conflict results.

The new task terminates.

The request for the new task goes in the Wait List. The original task searches the Wait List for work to do and should process the request normally. However, when this message persists, then storage may be in error.

015 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions

The program issued a GetMain Command to build a main storage table. The Command failed. x.x.K was the amount of storage requested.

EIBRESP contains the return code.

TYMEAC obtains storage using the Flength option. TYMEAC does not use the UserDataKey option, therefore, the key of the program determines the key of the storage.

020 S SHUT DOWN DETECTED

Reasons and solutions

See 005 S above.

025 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 015 S, above.

Task Storage is on the Queue Task's storage chain rather than Shared storage.

030 S READ FOR TS RECORD FAILED

Reasons and solutions

Queue tasks processing Asynchronous Requests use a temporary storage queue to pass input parameters and hold output. The read for this queue failed. See EIBRESP for the reason.

035 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 025 S, above.

040 S READ FOR TS RECORD FAILED

Reasons and solutions

See 030 S above.

045 S x.x.K GETMAIN FOR SHARED STORAGE FAILED

Reasons and solutions

See 025 S, above.

050 S x.x.K GETMAIN FOR TASK STORAGE FAILED

Reasons and solutions

See 025 S, above.

055 S READ FOR TS RECORD FAILED

Reasons and solutions

See 030 S Above.

060 I SHUT DOWN DETECTED

Reasons and solutions

See 005 I above.

065 S INVALID RESP FROM LINK, SEE EIBRESP

Reasons and solutions

The Queue Task issued a CICS® LINK to the application program. The

response was other than zero. See EIBRESP.

070 I SHUT DOWN DETECTED

Reasons and solutions

See 005 I above.

075 I SHUT DOWN DETECTED

Reasons and solutions

See 005 I above.

180 S INVALID RC FROM POST COMMAND, TIME IN ERROR

Reasons and solutions

The Queue Task issued a CICS® POST command using the interval stored in the Queue Header data. This interval loads from the Configuration File at Start Up.

If the Header data is not in error, than the Configuration File Maintenance Transaction improperly accepted the interval..

085 I SHUT DOWN DETECTED

Reasons and solutions

See 005 I above.

090 S INVALID WAIT RC

Reasons and solutions

The Queue Task issued a WAIT command and the command failed. See EIBRESP.

900 S ABEND - SEE ABCODE FOR REASON

Reasons and solutions

Execution entered the Queue Task Abend Exit Routine as a result of an application ABEND.

010 S INVALID TYPE FOUND IN TD RECORD

Reasons and solutions

The TD record written to the queue is in error, which probably means that Main Storage is in error.

020 S INVALID WRITE STAT FILE, SEE EIBRESP

Reasons and solutions

The WRITE to the statistics file failed. See the return code in EIBRESP.

3.4 MAIN STORAGE TABLES DISPLAY

Transaction id: NN23, Program: NN23PGM1

This is the Main storage display of the TYMEAC internal tables.

Appendix 3.4.A contains a full list of all screen messages.

3.4.1 MAIN STORAGE MENU

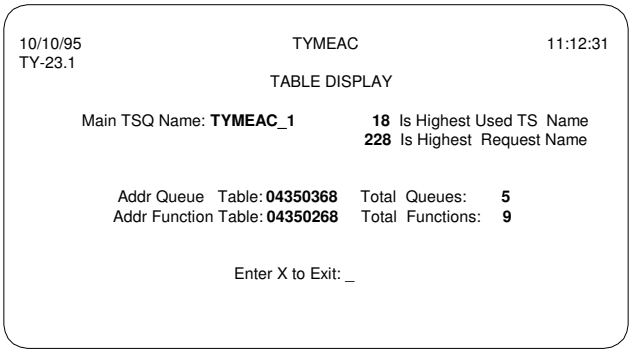


Figure 3.4.1 Main Storage Tables Menu.

TSQ Name:
All TYMEAC storage chains from this Anchor Point main temporary storage record.

High Used TS Name:
TYMEAC uses temporary storage queues to pass Asynchronous Requests to, and among, the Queue Tasks. The name is a fifteen digit number. This is the highest generated number since Start Up. (Also known as the Req_Id.)

High Used Request Name:
TYMEAC uses the Request Table to pass Synchronous Requests to the Queue Tasks. The name is a fifteen digit number. This is the highest number generated since Start Up.

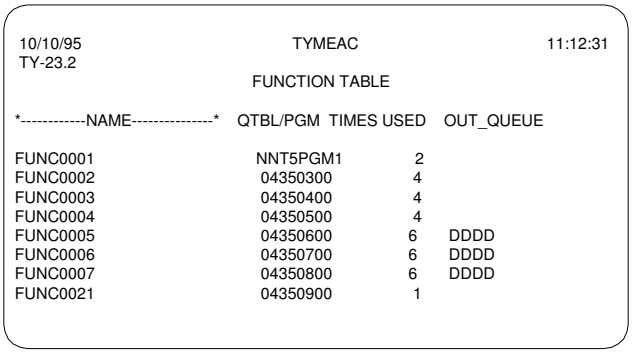
Queue Table:
This is the hex address of the main (list of Queues) table and the total number of Queues.

Function Table:
This is the hex address of the Function Table and the total number of Function defined in the system.

Exit:
Enter any character to return to CICS® or use the Clear Key.

Place the cursor on either the Function or Main line to display further details. <ENTER>

3.4.2 FUNCTION TABLE



10/10/95 TYMEAC 11:12:31
TY-23.2

FUNCTION TABLE

----------NAME----------	QTBL/PGM	TIMES USED	OUT_QUEUE
FUNC0001	NNT5PGM1	2	
FUNC0002	04350300	4	
FUNC0003	04350400	4	
FUNC0004	04350500	4	
FUNC0005	04350600	6	DDDD
FUNC0006	04350700	6	DDDD
FUNC0007	04350800	6	DDDD
FUNC0021	04350900	1	

Figure 3.4.2 Function Table.

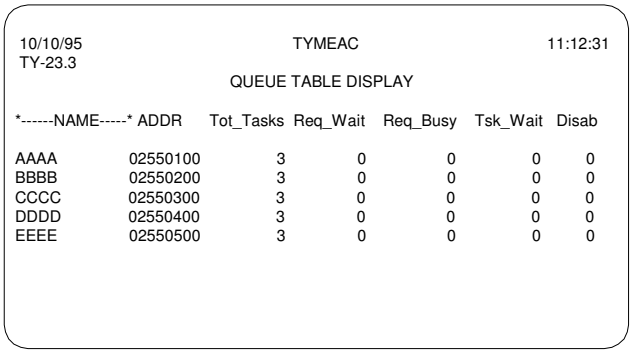
Name:
The name of the Function

QTBL/PGM
The hex address of the Queue Table for this Function or the name of the program, if a Non-Queue Function.

Times Used:
The accumulated requests for the Function since Start Up.

Out Queue:
The name of the Output Agent Queue, (if part of the Function).

3.4.3 MAIN TABLE



10/10/95 TYMEAC 11:12:31
TY-23.3

QUEUE TABLE DISPLAY

------NAME-----	ADDR	Tot_Tasks	Req_Wait	Req_Busy	Tsk_Wait	Disab
AAAA	02550100	3	0	0	0	0
BBBB	02550200	3	0	0	0	0
CCCC	02550300	3	0	0	0	0
DDDD	02550400	3	0	0	0	0
EEEE	02550500	3	0	0	0	0

Figure 3.4.3 Main Table.

NAME:
The name of the Queue.

ADDR:
The main storage address of the Queue, in hex.

TOTAL TASKS:
The maximum number of CICS® tasks that can process this Queue.

REQUESTS WAITING:
The total number of requests that are waiting for processing.

REQUESTS BUSY:
The total number of CICS® tasks that are processing requests.

TASK WAITS:
The total number of tasks waiting for an ECB post.

DISABLED:
The total number of task Anchor Point entries within a Queue marked ‘disabled’.

Place the cursor on a Queue line for a detail display. <ENTER>

3.4.4 QUEUE TABLE DETAIL

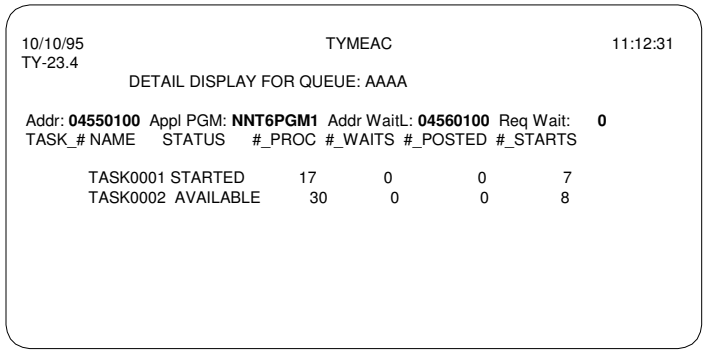


Figure 3.4.4 Queue Detail Table.

ADDR:
The main storage address of the Queue, in hex.

APPL PGM:
The name of the Processing Application Program to process the request.

ADDR WAITLIST:
The main storage address of the Wait List, in hex.

REQ WAITING:
The total number of requests, in all Wait Lists, which are waiting for processing.

Task #:
The CICS® task number of the active task attached to this entry.
NAME:
The TYMEAC internal name, ('TASK' plus a sequence number).

STATUS:

Available	Available for a new task. No CICS® task is processing this Anchor Point.
Processing	A CICS® task is actively processing this request.
In link to Appl	The Queue Task issued a CICS® LINK to the Processing Application Program.

In link to Schd	The Queue Task issued a CICS® LINK to the Scheduler program for the Output Agent Queue.
Posted	Another task posted the Queue Task's ECB.
Started	Another task issued a CICS® START for a new task. The new task has not become active.
Canceled	The Monitor task determined that a problem may exist, i.e., excessive time in a 'in link' status. If the task resumes processing, the status resets to processing.
Disabled	The entry may not accept any new request. See Disabled Queue Transaction, Appendix 3.5.A, for a description of how an entry becomes disabled.

#_Processed:
The total number of requests processed since system Start Up.

#_Waits:
The total number of wait time-outs since Start Up.

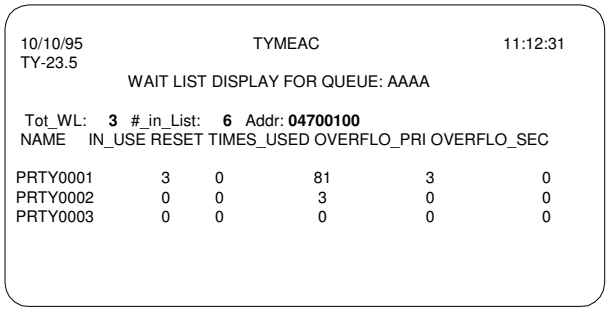
#_Posted:
The total times an ECB was handposted by a requesting task since system Start Up.

#_**Started:**
The total number of CICS® START commands issued for this task entry since system Start Up.

N.B. The number posted may far exceed the number of waits since another task may post this task at anytime in the interval between this task setting its status to 'available to post', and the WAIT time expiring.

Use Function Key 4 for the Wait List display.

3.4.5 WAIT LIST TABLE



10/10/95 TYMEAC 11:12:31
TY-23.5

WAIT LIST DISPLAY FOR QUEUE: AAAA

Tot_WL: **3** #_in_List: **6** Addr: **04700100**

NAME	IN_USE	RESET	TIMES_USED	OVERFLO_PRI	OVERFLO_SEC
PRTY0001	3	0	81	3	0
PRTY0002	0	0	3	0	0
PRTY0003	0	0	0	0	0

Figure 3.4.5 Wait List Display.

TOT WL:
The total number of Wait Lists for this Queue.

 # IN_LIST:
The maximum number of entries in each Wait List.

ADDR:
The main storage address of the first Wait List, in hex.

NAME:
The TYMEAC internal name for the Wait List entry.

IN_USE:
The total number of requests waiting for processing.

RESET:
Back-out processing and time-out processing mark entries as 'reset' rather than compress the list. The Queue Task is the only task that compresses and pops-up the Wait Lists.

TIMES USED:
The number of requests Processed.

OVERFLOWED
PRIMARY:

Requests go into the Wait List according to the priority requested. When the list is full, the request goes into the next higher Wait List. This is a primary overflow.

SECONDARY:

As above, however, if the next higher Wait List is full, the request goes into the next available Wait List. This is a secondary overflow.

APPENDIX 3.4.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN23 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that TYMEAC is not active.

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

025 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page of data. To refresh the display, use the Prior Key and start over.

3.5 DISABLED QUEUE UPDATE

Transaction id: NN24, Program: NN24PGM1

Task Queue Anchor Point entries can become disabled for a variety of reasons. (See Appendix 3.5.A). This is not a CICS® disable but a logical TYMEAC 'not available' status. Once the System Administrator knows the reason for this 'disable' and applies a solution, normal processing can continue after resetting the 'disabled' entries.

TYMEAC provides a second program, NN24PGM2, that is conversational and allows Queue names in lower case. The transaction id is NN2B, program, NN24PGM2.

Appendix 3.5.B contains a full list of screen messages.

3.5.1 DISABLED QUEUE MENU

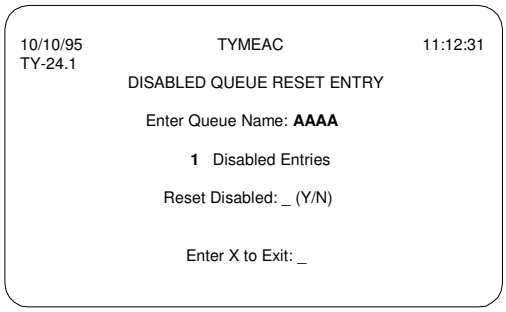


Figure 3.5.1 Disabled Queue Menu.

Name:
Enter the name of a Queue.

 # Disabled:
The total number of entries marked as disabled.

Reset:
When the number of disabled entries is greater than zero, enter a 'Y' to reset all disabled entries. In addition to resetting the disabled entries, when the Queue has Wait List entries and no Queue Task is processing the Queue, the transaction STARTs a task to process the Wait Lists.

Exit:
Enter any character to return to CICS® or use the Clear Key.

APPENDIX 3.5 A

How a Queue Task Table Entry becomes disabled.

The status of Queue Task Anchor Points is relevant to scheduling. Queue Tasks process a request and then look in the Wait Lists for the next request. When the status of any Queue Task Anchor Point entry is 'busy' or 'in-link', it is reasonable to assume that since Queue Tasks are processing, a Queue Task will process any request put into a Wait List. It is also reasonable to assume that Queue Task Anchor Point entries of 'posted' or 'started' will become active shortly and process the Wait Lists.

When a Queue Task remains in an active or about to be active state for an unreasonable amount of time then something may be wrong. Marking the entry 'canceled' and then 'disabled' after even more elapsed time is a way of eliminating the Anchor Point from consideration during scheduling. For a non performing Queue, rather than filling up all the Wait Lists with pending requests and returning a 'no Wait List available' message to the new request, the Scheduler rejects the request immediately with a 'no task available' message. The 'no Wait List available' message could mean, try again later. The 'no task available' message means something is wrong. See also Section 7.5, Scheduling failures.

The Monitor Task notifies system administrators when all Anchor Points are 'disabled'.

An entry becomes 'disabled' by:

1. An ABEND in an application program. Queue Tasks handle abends by marking the Anchor Point entry 'disabled'.

2. The Monitor task changes the status from Canceled to Disabled.

   TYMEAC keeps track of the time of five events:

   1. The time the Request Broker issues a CICS® START for a new Queue Task.
   2. The time the Queue Task issues a CICS® LINK to a Processing Application Program or the Scheduler, (for the Output Agent Queue).
   3. The time the Scheduler posts a waiting Queue Task's ECB.
   4. The time a Queue Task enters a 'busy' status, (executing TYMEAC code and commands.)
   5. The time the Request Broker or Queue Task issues a WAIT command.

   For 1, 2, 3, and 4 The maximum time allowable is one monitor task interval i.e., the interval the monitor task issues on a CICS® START for itself, or, one minute, if the interval is less than one minute. When the Queue Task exceeds the maximum time, the Monitor Task sets the entry 'canceled'. The next time the Monitor task runs, it sets all 'canceled' entries 'disabled'. If a Queue Task resumes processing, the Queue Tasks sets the entry 'busy', irrespective of any prior setting.

The Queue Task nulls the ‘start of busy’ time when the Queue Task leaves ‘busy’ status, (to ‘wait‘ status or ‘in-link’ status), and resets the ‘start of busy’ time when the Queue Task enters ‘busy’ status. Therefore, this is not a cumulative total or elapsed time, but a pure TYMEAC Queue Task busy time.

For 5. The maximum time allowable is 999 times the interval for the WAIT command issued by the Request Broker or the Queue Task Program.

N.B. The timing interval of the five events is the interval of the Monitor Task. Using variables for System, (and Queue), would establish an exclusive environment for a particular business need. When the business requirements change, the variables would need changing. When the platform changes, the variables would need changing. Each application on each platform would need re-engineering each time the business changes either internally by function or externally by porting.

TYMEAC event timing functions identically on all platforms. Adjusting the Monitor Task interval sets the interval across all TYMEAC timed events. Porting to any machine preserves the same application demeanor, (synergy); it acts the same only faster, (or slower). See also Section 6, Tuning, (Wait Lists, fixed number of entries within each Queue).

3. When the Queue Task detects Immediate Shut Down. See Section 5.2, Shut Down.

APPENDIX 3.5 B

SCREEN MESSAGES

MESSAGE STRUCTURE

NN24 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that TYMEAC is not active.

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I INVALID QUEUE NAME

Reasons and solutions:

The name is not a valid Queue name found in the Main (Queue) Table.

025 S QUEUE IN SHUT DOWN MODE

Reasons and solutions:

See 015 S, above.

030 I NO TASKS WERE DISABLED

Reasons and solutions:

This is list of disabled only task entries.

035 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

040 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page of data. To refresh the display, use the Prior Key and start over.

045 I ENTER A QUEUE NAME

Reasons and solutions:

Enter a valid Queue name.

050 I OK

Reasons and solutions:

The last request finished successfully.

3.6 STALL TABLE MAINTENANCE

Transaction id: NN25, Program: NN25PGM1

Asynchronous Requests can stall for three reasons.

1. A failure to schedule, (Section 4.2 return codes), a Queue and subsequent back-out failure during Request Broker processing. This failure is specific. It is recoverable in that the Request Broker informs the requesting task of the failure to schedule and the back-out failure is resolved over time.

Asynchronous Requests hold valuable system resources. The input area for all Queue Tasks is in shared storage. The request parameters for all Queue Tasks are in a temporary storage queue, item one. The Queue Tasks save the output data, if any, of each Processing Application Program in this temporary storage queue with item incremented.

The TYMEAC Request Broker initially writes the item one temporary storage record, adds an entry to the Temporary Storage Table, and then schedules each Queue in the Function. If a schedule fails, previously scheduled Queues are backed-out. This back-out procedure may fail. When it does, the Request Broker adds an entry to the Stall Table since it is in-doubt whether it can release all the request's resources.

The Monitor Task scans the Stall Table. When it finds an entry, added by the Request Broker during a failed back-out, that is no longer in-doubt, it deletes that entry.

2. A failure to schedule, (Section 4.2 return codes), the Output Agent Queue during phase two of Asynchronous Request processing. This failure is specific and recoverable by the re-schedule procedure, (Section 3.6.3).

The Output Agent Queue is the TYMEAC Queue that processes the combined output of all the previously executed asynchronous component processes of the Function.

The last Queue Task to finish processing schedules the Output Agent Queue. However, when scheduling fails, the Asynchronous Request stalls, (that is, it cannot complete). The Queue Task does not add an entry to the Stall Table. This is done by the Monitor Task, below.

3. A failure within an Asynchronous Request process during phase one, (prior to scheduling the Output Agent Queue), that results in a task purge. This failure is non-specific and may be non-recoverable.

There are two causes for this failure:
1. (a) The Transaction Monitor may be at maximum tasks, short on resources, or under stress and it purges tasks.
   (b) An agent of the Transaction Monitor purges tasks.
   (c) A Resource Manager or Adapter failed and it purged associated threads.
2. (a) The Processing Application Program issued a direct operating system call that

failed.

(b) The Processing Application Program failed in an abend exit routine.

The Asynchronous Request will never finish properly. The state of the resources used by this purged task may be 'inconsistent', (as defined by the ACID test).

The Monitor Task scans the Temporary Storage Table. If it finds an entry that it believes, 'Has been here much too long', it adds an entry to the Stall Table. TYMEAC is a process immediately system. This belief is sometimes wrong and the Monitor Task eventually cleans-up this false entry.

This transaction is primarily for system administrators, and therefore, Section 7.2, Stall Table for problem determination, fully describes this transaction.

The number of entries in this table is the sum of one half the number of Queue Task Table entries, (that which can support a CICS® task), plus one quarter the number of all Wait List entries. The table is dynamically expandable. When no entries are available, TYMEAC chains a next-table to the last and frees this next-table when no longer needed.

Appendix 3.6.A contains a full list of screen messages.

Copy code for this table is NNSTALL.

3.6.1 STALL TABLE MENU

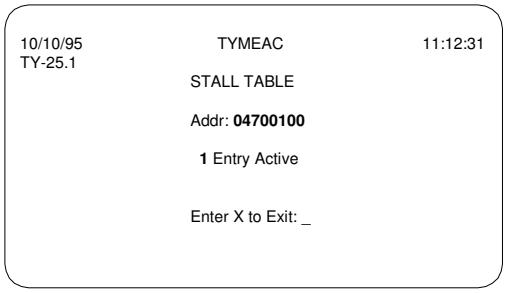


Figure 3.6.1 Stall Table Menu.

ADDR:
The main storage address of the base table, in hex. When the base table is full, TYMEAC chains subsequent tables from the base table and frees them when no longer needed.

Active:
The total entries active in all tables.

Exit:
Enter any character to return to CICS® or use the Clear Key.

Press the <ENTER> key to display a browse of active entries.

3.6.2 STALL TABLE BROWSE

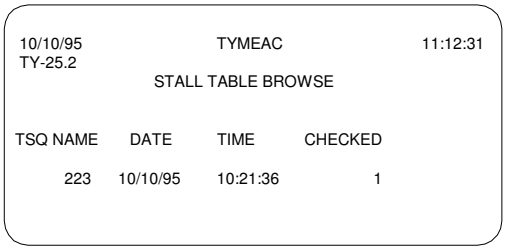


Figure 3.6.2 Stall Table Browse.

The Stall Table is a 'first available' type table. When adding entries to the table the system finds the first available entry and marks it 'in-use'. When purging entries, the system marks the entry 'available'.

The browse starts at the beginning of the table and picks up all 'in-use' entries until the end. Therefore, the display is not in chronological order. The transaction saves these entries in a temporary storage record.

Forward, backward paging is from the temporary storage queue and does not include any entry set 'in-use' after the queue's formation.

To rebuild the temporary storage queue, use the Prior Key, and start over.

TSQ Key:
The key is the name of the temporary storage queue for the Asynchronous Request, (a fifteen digit number). The transaction sorts all 'in-use' entries in ascending order and displays the first fourteen. This is the Req_Id.

Date/Time:
The date and time of entry into this table.

Checked:
The number of times the Monitor task checked this entry, To determine whether it is still valid. Place the cursor on a line and press <ENTER> for a detail display.

3.6.3 STALL TABLE DETAIL

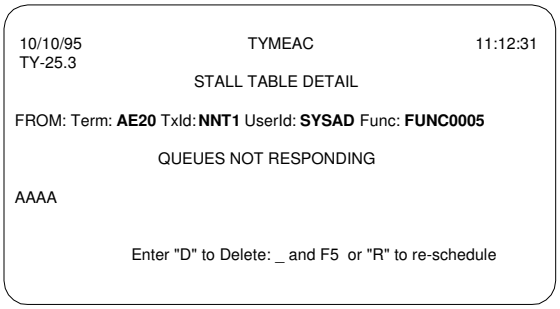


Figure 3.6.3 Stall Table Detail.

The requested data may no longer be available. The Monitor Task scans this table looking for entries that it can purge, (all Queue Tasks responded, etc.)

When the requested data is no longer available, the transaction displays the menu so that the user may get a current list of entries.

From Information:
Terminal, Transaction, Userid, and Function from the input source.

Queues:
The names of the TYMEAC Queues that have not finished processing. The Output Agent Queue, if part of the Function, is first.

The term 'not finished', for this display, means a Queue Task that did not update the temporary storage queue indicating that the Processing Application Program finished.

A task may abnormally terminate at any stage, (program). A program may abnormally terminate at any position. The key in this situation is whether the Processing Application Program, when a Queue Task, or, Client Application Program, when an Output Agent Task, successfully completed. This is important for the delete/re-schedule option, below.

When the Output Agent Queue is the only Queue on the display, then the Client Application Program did not begin. The Output Agent Task deletes the temporary storage queue before LINKing to the Client Application Program.

When the Output Agent Queue is **NOT** the only Queue on the display, then the stage of processing is unknown. The system administrator must use whatever means are available, including the 'From Information', above, to determine the stage of application processing.

Delete/Re-Schedule:
To delete this entry, place a “D” in the field and <F5>.
The delete frees the input area and deletes the temporary storage queue.

To re-schedule this entry, only when the re-schedule phrase is present on the screen, place an ”R” in this field and <ENTER>.

Re-scheduling is only for those Queues that did not process and only for those Asynchronous Requests that did not fail scheduling at the Request Broker level. The Request Broker encounters scheduling failures and informs the requester of the problem. However, the Request Broker adds a Stall Table entry during a failed back-out. The Monitor Task eventually cleans up this entry, (see Section 5.5 Monitoring). When the Asynchronous Request failed scheduling at the Request Broker level, the re-scheduling phrase is not present on the screen.

E.G. Function: ‘Re-value’ requires two Queues, ‘Que_A’ and ‘Que_B’, plus an Output Agent, ‘Que_final’. Both ‘Que_A’ and ‘Que_B’ finished processing. ‘Que_final’ failed scheduling at the Queue Task level and the Monitor Task added a Stall Table entry. The re-scheduling is for ‘Que_final’, only.

APPENDIX 3.6.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN25 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that the system was not started

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I VIEWED DATA NO LONGER VALID

Reasons and solutions:

The table entry no longer exists.

025 I ENTER “D” AND F5 TO DELETE

Reasons and solutions:

To delete the entry, place a “D” in the delete field and use the F5 key.

030 I NO FURTHER ENTRIES

Reasons and solutions:

There are no further entries in the table.

035 I NO VALID ENTRIES FOUND

Reasons and solutions:

No entries exist in the table.

040 I INVALID CURSOR POSITION FOR DETAIL

Reasons and solutions:

For detail information, the cursor must be on a data line.

045 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

050 I TEMP STORAGE READ FAILED, TRY AGAIN

Reasons and solutions:

The read for the temporary storage record failed.
Try again later.

055 S TEMP STORAGE READ FAILED, PGM RECORD

Reasons and solutions:

The read for temporary storage record used by the transaction to hold the browse data failed. Check for CICS® system damage.

060 S TEMP STORAGE WRITE FAILED

Reasons and solutions:

See 055 S, above.

065 I OK

Reasons and solutions:

The last command processed successfully.

070 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page for browsing.

080 S REQUEST WAS BACKED-OUT, RESCHEDULING NOT POSSIBLE

Reasons and solutions:

Re-scheduling is only for those Asynchronous Requests that the Request Broker successfully processed. A partial back-out by the Request Broker cannot re-schedule.

090 S GETMAIN FAILED

Reasons and solutions:

This System is short on storage.

100 S QUEUE=xx, SCHEDULING FAILED, RC=yyyy

Reasons and solutions:

The Queue (xx=name) failed scheduling. yyyy is the return code from the Scheduler.

010 S RESCHEDULING FAILED, SCHEDULER IN ERROR

Reasons and solutions:

The LINK to the Scheduler program failed. Make sure the Scheduler program name in the Configuration File is correct and that the program's status is correct.

3.7 TEMPORARY STORAGE TABLE MAINTENANCE

Transaction id: NN26, Program: NN26PGM1

This is the main storage display of the Temporary Storage Table. TYMEAC supplies this transaction for testing environments. Since TYMEAC uses, frees, and reuses entries in fractions of a second the usefulness of this transaction in a production environment is slim.

Asynchronous Requests hold valuable system resources. The input area for all Queue Tasks is in Shared Storage. The request parameters for all Queue tasks are in a temporary storage queue, item one, (copy code is NNTSREC). The Queue Task puts output data, if any, of each Processing Application Program into this temporary storage queue with item incremented. When all Queue Tasks finish, the Queue Task frees the input area and deletes the temporary storage queue.

For each temporary storage queue written, the Request Broker puts an entry into this table so that if something goes wrong, an abend, task purge, etc., there is a way of freeing resources.

The number of entries in this table is the sum of one half the number of Queue Task Table entries (that which can support a CICS® task), plus one quarter the number of all Wait List entries. The table is dynamically expandable. When no entries are available, TYMEAC chains a next-table to the last and frees this next-table when no longer needed.
.

Appendix 3.7.A contains a full list of screen messages.

Copy code for this table is NNTSTBL.

3.7.1 TEMPORARY STORAGE TABLE MENU

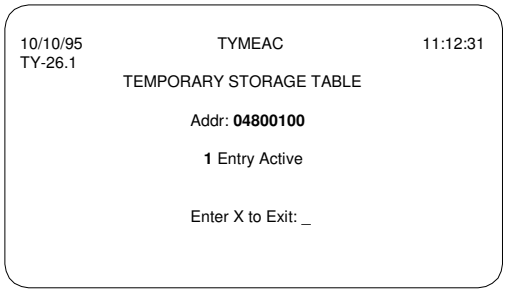


Figure 3.7.1 Temporary Storage Table Menu**.**

ADDR:
The main storage address of the base table, in hex. When the base table is full, TYMEAC chains subsequent tables from the base table and frees them when no longer needed.

Active:
The total entries active in all tables.

Exit:
Enter any character to return to CICS® or use the Clear Key.

Press the <ENTER> key to display a browse of active entries.

3.7.2 TEMPORARY STORAGE TABLE BROWSE

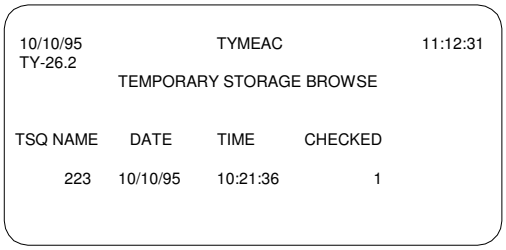


Figure 3.7.2 Temporary Storage Table Browse.

The Temporary Storage Table is a 'first available' type table. When adding entries to the table the system finds the first available entry and marks it 'in-use'. When purging entries, the system marks the entry 'available'.

The browse starts at the beginning of the table and picks up all 'in-use' entries until the end. Therefore, the display is not in chronological order. The transaction saves these entries in a temporary storage record.

Forward, backward paging is from the temporary storage queue and does not include any entry set 'in-use' after the queue's formation.

To rebuild the temporary storage queue, use the Prior Key, and start over.

TSQ Key:
The key is the name of the temporary storage queue for the Asynchronous Request, (a fifteen digit number). The transaction sorts all 'in-use' entries in ascending order and displays the first fourteen. This is the Req_Id.

Date/Time:
The date and time of entry into this table.

Checked:
The number of times the Monitor task checked this entry. When this number is two (2) the Monitor Task adds an entry to the Stall Table.

Place the cursor on a line and press <ENTER> for a detail display.

3.7.3 TEMPORARY STORAGE TABLE DETAIL

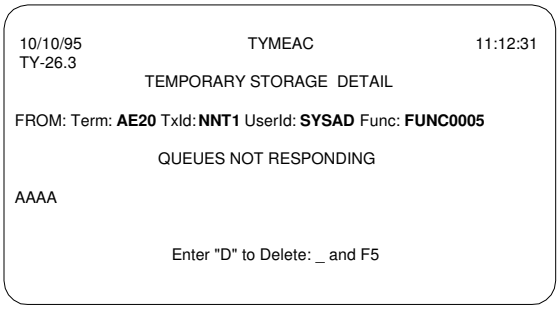


Figure 3.7.3 Temporary Storage Table Detail.

The requested data may no longer be available. The Monitor Task scans this table looking for entries that it can be purge, (all Queue Tasks responded, etc.)

When the requested data is no longer available, the transaction displays the menu so that the user may get a current list of entries.

From Information.:
Terminal, Transaction, Userid, and Function from the input source.

Queues:
The names of the TYMEAC Queues that have not finished processing.

Delete:
To delete this entry, place a “D” in the field and <F5>.

APPENDIX 3.7.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN26 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that the system was not started

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I VIEWED DATA NO LONGER VALID

Reasons and solutions:

The table entry no longer exists.

025 I ENTER “D” AND F5 TO DELETE

Reasons and solutions:

To delete the entry, place a “D” in the delete field and use the F5 key.

030 I NO FURTHER ENTRIES

Reasons and solutions:

There are no further entries in the table.

035 I NO VALID ENTRIES FOUND

Reasons and solutions:

No entries exist in the table.

040 I INVALID CURSOR POSITION FOR DETAIL

Reasons and solutions:

For detail information, the cursor must be on a data line.

045 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

050 I TEMP STORAGE READ FAILED, TRY AGAIN

Reasons and solutions:

The read for the temporary storage record failed.
Try again later.

055 S TEMP STORAGE READ FAILED, PGM RECORD

Reasons and solutions:

The read for the temporary storage record used by the transaction to hold the browse data failed. Check for CICS® system damage.

060 S TEMP STORAGE WRITE FAILED

Reasons and solutions:

See 055 S, above.

065 I OK

Reasons and solutions:

The last command successfully processed.

070 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page for browsing.

3.8 REQUEST TABLE MAINTENANCE

Transaction id: NN27, Program: NN27PGM1

This is the display of the main storage Request Table. TYMEAC supplies this transaction for testing environments. Since TYMEAC uses, frees, and reuses entries in fractions of a second the usefulness of this transaction in a production environment is slim.

No task touches the storage of another task. This includes read only storage. All TYMEAC tables are in shared storage. All input/output areas and request/response parameters are in shared storage.

Each component of the Synchronous Request generates an entry in this table. When the component completes, the Request Broker frees the entry. This table is the Request Parameter List for each component of the request.

The Request Broker moves the input area from the request to shared storage. Each Queue Task moves the output from the Processing Application Program, temporarily, to shared storage and then the Request Broker moves the output to its storage chain (task).

The ECB for posting the requester and other parameters are in this table.

The number of entries in this table is the sum of one half the number of Queue Task Table entries (that which can support a CICS® task), plus one quarter the number of all Wait List entries. The table is dynamically expandable. When no entries are available, TYMEAC chains a next-table to the last and frees this next-table when no longer needed.
.

Appendix 3.8.A contains a full list of screen messages.

Copy code for this table is NNREQCA.

3.8.1 REQUEST TABLE MENU

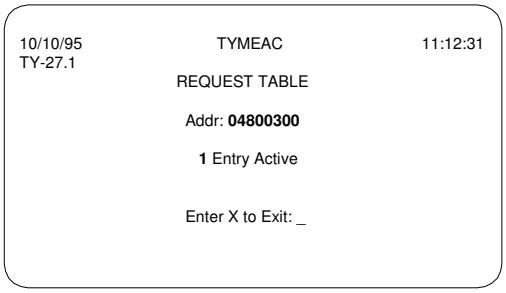


Figure 3.8.1 Request Table Menu.

ADDR:
The main storage address of the base table, in hex. When the base table is full, TYMEAC chains subsequent tables from the base table and frees them when no longer needed.

Active:
The total entries active in all tables.

Exit:
Enter any character to return to CICS® or use the Clear Key.

Press the <ENTER> key to display a browse of active entries.

3.8.2 REQUEST TABLE BROWSE

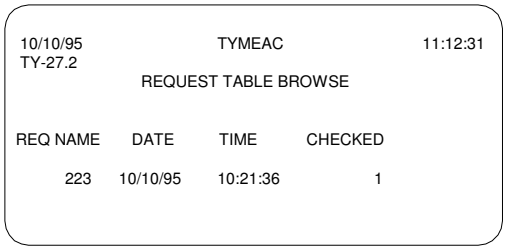


Figure 3.8.2 Request Table Browse.

The Table is a 'first available' table. When adding entries to the table the system finds the first available entry and marks it 'in-use'. When purging entries, the system marks the entry 'available'.

The browse starts at the beginning of the table and picks up all 'in-use' entries until the end. Therefore, the display is not in chronological order. The browse saves these entries in a temporary storage queue.

Forward, backward paging is from the temporary storage queue and does not include any entry made after the queue's formation.

To rebuild the temporary storage queue, use the Prior Key, and start over.

Req Name:
The key is a fifteen digit number from the Number Generation Table. The transaction sorts all 'in-use' entries in ascending order and displays the first fourteen.

Date/Time:
The date and time of entry into this table.

Checked:
The number of times the Monitor Task checked this entry. (Used internally by TYMEAC).

Place the cursor on a line and press <ENTER> for a detail display.

3.8.3 REQUEST TABLE DETAIL

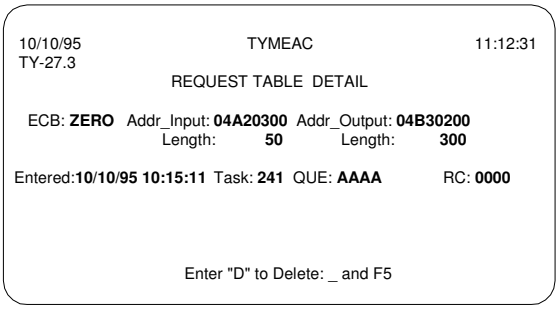


Figure 3.8.3 Request Table Detail.

The requested data may no longer be available. The Monitor Task scans this table looking for entries which it can be delete, (all Queue Tasks responded, etc.)

When the requested data is no longer available, the transaction displays the menu so that the user may get a current list of entries.

ECB:
Posted, or Zero (not posted).

Addr Input / Length:
The main storage address of the input data, (hex). / The length of the input data, (decimal).

Addr Output / Length:
The main storage address of the output data, (hex). / The length of the output data, (decimal).

Entered:
The date and time of entry into the table.

Task:
The CICS® task number of the task associated with this entry.

Que:
The TYMEAC Queue relevant to this entry.

RC:
The internal return code for TYMEAC processing:

Spaces - no work completed.
0000 The request processed normally.
9xxx The request timed-out or was backed-out by another Queue error.

DELETE:
To delete this entry, place a “D” in the field and <F5>.

APPENDIX 3.8.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN27 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that TYMEAC is not active.

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I VIEWED DATA NO LONGER VALID

Reasons and solutions:

The table entry no longer exists.

025 I ENTER “D” AND F5 TO DELETE

Reasons and solutions:

To delete the entry, place a “D” in the delete field and use the F5 key.

030 I NO FURTHER ENTRIES

Reasons and solutions:

There are no further entries in the table.

035 I NO VALID ENTRIES FOUND

Reasons and solutions:

No entries exist in the table.

040 I INVALID CURSOR POSITION FOR DETAIL

Reasons and solutions:

For detail information, the cursor must be on a data line.

045 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of data.

050 I TEMP STORAGE READ FAILED, TRY AGAIN

Reasons and solutions:

The read for the temporary storage record failed.
Try again later.

055 S TEMP STORAGE READ FAILED, PGM RECORD

Reasons and solutions:

The read for the temporary storage record used by the transaction to hold the browse data failed. Check for CICS® system damage.

060 S TEMP STORAGE WRITE FAILED

Reasons and solutions:

See 055 S, above.

065 I OK

Reasons and solutions:

The last command successfully processed.

070 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page for browsing.

3.9 REQUEST STATUS DISPLAY

Transaction id: NN28, Program: NN28PGM1

This is the display of the status of Asynchronous Requests in the TYMEAC system.

Enter the transaction id on a 3270 type terminal. Start the transaction with a look alike 3270 terminal, (FEPI, etc.), by sending a transaction id and request id. LINK to the program with a Common Area.

For standard 3270, screen processing, enter a transaction id. The transaction displays the MENU, Figure 3.9.1.

For non screen processing, enter a transaction id, space and request id.

(E.G. NN28b000000000000025 where NN28 is the transaction id.
b is a single character.
0...25 is the fifteen digit request id.

The transaction displays the reply by a non-formatted, CICS® SEND command, see Appendix 3.9.A.

Verification of input is that the transaction id must match the EIBTRNID, the request id must be fifteen digits and non zero. When the verification fails, the transaction assumes screen processing.

LINKing to this program is a way for programs outside the TYMEAC image to check the status of an Asynchronous Request previously sent. By using a front-end program to handle the communication, form a common-area as follows, (copy code, NN28):

Length is 130 bytes.

ID -- 12 characters, with a value of: “nnstatus_001”. (not case sensitive)
REQ_ID -- 15 digits of the right justified, zero filled request id
TSU-ABS -- 15 digits of the Absolute Time of TYMEAC Start Up.
APPLID-- 08 characters of the CICS® APPLID of the target TYMEAC system.
(case sensitive)
MSG -- 80 characters for the return message.

The TYMEAC Request Broker returns the REQ_ID, TSU-ABS, and APPLID when scheduling Asynchronous Requests. See Section 4.1 Usage.

3.9.1 REQUEST STATUS MENU

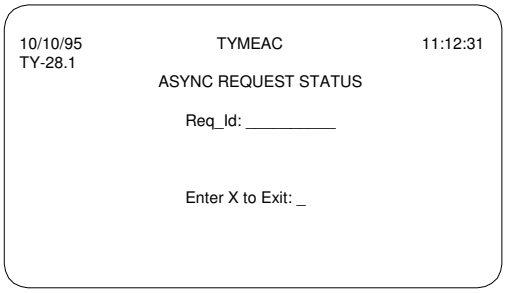


Figure 3.9.1 Request Status Menu.

REQ_ID:
The Request Broker returns the Request Id for Asynchronous Requests. This is a fifteen digit number used by TYMEAC for the temporary storage queue name to control Asynchronous Request processing. The highest generated number is visible on the Main Table Display transaction, (Section 3.4.1).

Enter up to fifteen digits, left justified.

Exit:
Enter any character to return to CICS® or use the Clear Key.

Press the <ENTER> key to receive the status of the request_id.

APPENDIX 3.9.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN28 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S INVREQ, TXID MISMATCH

Reasons and solutions:

The transaction id in the input data does not match the EIBTRNID. The transaction does not support forming a dummy terminal input area and transferring control to this program.

010 S INVREQ, APPLID MISMATCH.

Reasons and solutions:

The CICS® APPLID passed by the LINKing program does not match the APPLID of the image where this program is running.

015 S INVREQ, START UP ABS MISMATCH.

Reasons and solutions:

The Start Up Absolute Time passed by the LINKing program does not match the Start Up absolute time of the current TYMEAC system. This may result from TYMEAC Shut Down, followed by a Start Up.

020 S INVREQ, REQ_ID NOT NUMERIC OR ZERO

Reasons and solutions:

The Request Id must be fifteen digits and not all zero.

025 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed. Check to see that this file is open and enabled, or, not corrupted.

030 S TYMEAC NOT ACTIVE

Reasons and solutions:

The read for the main temporary storage queue record failed. This may simply mean that TYMEAC is not active.

035 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

040 I REQ_ID GREATER THAN MAXIMUM

Reasons and solutions:

The Request Id is greater than the highest used at this time. The highest generated number is visible on the Menu screen, Main Table Display transaction.

045 I COMMON AREA ID INVALID

Reasons and solutions:

For LINKed to processing with a Common Area, the identification, (first twelve bytes), must be as detailed at the beginning of this section.

050 I REQ_ID=0..X, IN STALL TABLE SINCE xxx yyy

Reasons and solutions:

The Request (named) was found in the Stall Table. xxx, yyy is the date and time of the Stall Table entry.

055 I REQ_ID=0..X, NOT IN SYSTEM

Reasons and solutions:

The Request (named) is not in any TYMEAC table.
TYMEAC processing completed.

060 I REQ_ID=0..X, AWAITING EXECUTION

Reasons and solutions:

The Request (named) was found in the Temporary Storage Table with no Queues finished processing.

065 I REQ_ID=0..X, EXECUTING xxx OF yyy QUEUES FINISHED

Reasons and solutions:

The Request (named) was found in the Temporary Storage Table with xxx Queue's out of yyy total Queue's finished processing.

3.10 EXPORT TYMEAC TABLES

The purpose of the Export and Import is to move TYMEAC tables from site to site smoothly.

The correlation between TYMEAC Functions and Queues is critical. A Function with missing Queues is a fatal Start Up error. The Configuration File contains header records for the Functions and Queues with counts for each. The Start Up program verifies the counts and terminates when in error.

The Configuration File Maintenance transaction is the means to add, change, and delete Functions and Queues. Import does not replace existing Functions and Queues.

The Export/Import transactions provide the means to flawlessly move groups of Functions, with related Queues, from one site to another. This generally means moving a test system, to QA, to production.

EXPORT GROUPS

Transaction Id: NN29, Program: NN29PGM1

Associated Functions, with related Queues, move to the Export/Import File, (copy code NNEXIM). The transaction copies Functions and Queues from the Configuration File **without** replacing that which already exists. That is: In GROUP_A, Func_1 uses QUE_X and QUE_Y, and, QUE_Y already exists on the Export/Import File, then the transaction does **not** replace QUE_Y. The Export function is a final step in porting from site to site. Functions may share many Queue's.

TYMEAC provides a second, conversational transaction for entering Group and Function names in lower case. Transaction id is NN2C, program, NN29PGM2.

Appendix 3.10 A contains a full list of the screen messages.

3.10.1 EXPORT VALUES

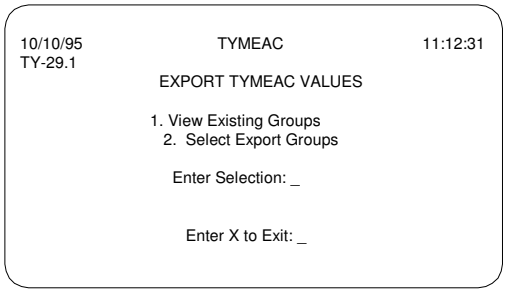


Figure 3.10.1 Export TYMEAC Values.

SELECTION:
1. View existing groups. Enter '1'. This displays the screen in Section 3.10.2; a view of the groups existing on the Export/Import File.

2. Select export groups. Enter '2'. This displays the screen in Section 3.10 4; the menu for exporting TYMEAC Functions and Queues.

Exit:
Enter any character to return to CICS® or use the Clear Key.

3.10.2 VIEW GROUPS

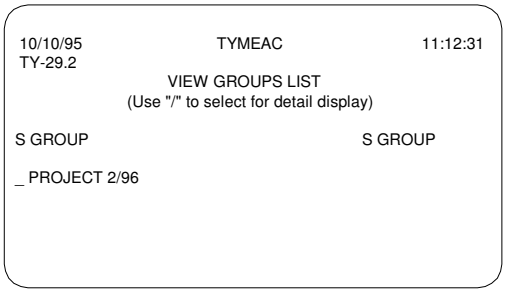


Figure 3.10.2 View Groups List.

“S” is Select:
Use a ‘/’ to select a group for detail viewing, i.e. associated Functions.

Group:
The name of the group.

3.10.3 VIEW GROUP DETAIL

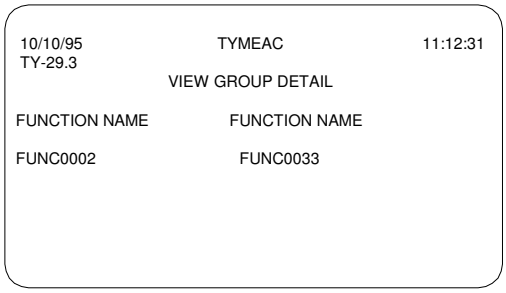


Figure 3.10.3 View Group Detail.

Function Name:
The names of the TYMEAC Functions associated with this group.

3.10.4 SELECT EXPORT GROUPS

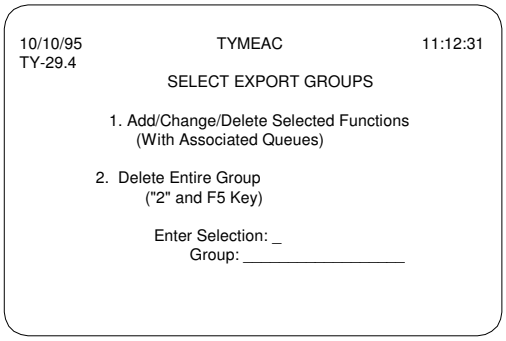


Figure 3.10.4 Select Export Groups.

TYPE:
1. To add, change, or delete selected Functions within a group. The transaction adds or deletes the Queues associated with these Functions.

2. Delete entire group. The transaction deletes all Functions with associated Queues. Requires Function Key '5'.

Group Name:
The twenty-four character name of the group to add, change, or delete. This is any character you wish to associate with your project.

3.10.5 EXPORT FUNCTIONS

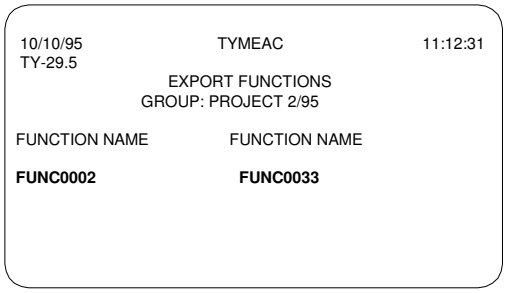


Figure 3.10.5 Export Functions.

Function Name:
The name of the Functions associated with this group for export. The transaction adds any Queues required by this Function, not already on the Export/Import File.

To delete a Function, space or erase the name. The transaction deletes any Queues used by this deleted Function and not used by any other Functions. When finished, use the F5 Key.

APPENDIX 3.10.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN29 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S NO GROUPS EXIST

Reasons and solutions:

The Export/Import file is empty.

010 S EXPORT/IMPORT FILE ERROR

Reasons and solutions:

A non-Normal code returned while accessing the file.
Check to see that the file is open and enabled.

015 S WRITE OF A TEMPORARY STORAGE RECORD FAILED

Reasons and solutions:

The program uses temporary storage to save intermediate results .
Check for out of space conditions or system damage.

020 S READ OF A TEMPORARY STORAGE RECORD FAILED

Reasons and solutions:

The program uses temporary storage to save intermediate results .
Check for out of space conditions or system damage.

025 I THIS IS THE FIRST PAGE

Reasons and solutions:

During back paging, this is the first page.

030 I THIS IS THE LAST PAGE

Reasons and solutions:

During forward paging, this is the last page.

035 S THE EXPORT/IMPORT FILE IS CORRUPTED

Reasons and solutions:

A NotFound condition occurred while browsing for Function records. The Group Header record contains the number of Functions associated with the group. The count therein does not match what was found on the file. Delete the file and rebuild.

040 S THE STATUS OF THE EXPORT/IMPORT FILE IS INVALID

Reasons and solutions:

Other than NotFound occurred while browsing for Function records. Delete the file and rebuild.

045 S SELECTION MUST BE “1” OR “2”

Reasons and solutions:

The only valid selection is a (1) or (2)

050 S INVALID OR MISSING GROUP NAME

Reasons and solutions:

Enter a Group name.

055 I USE FUNCTION KEY “5” TO PERFORM THE DELETE

Reasons and solutions:

Function Key “5” required in addition to the selection to delete.

060 I THE GROUP HAS BEEN DELETED

Reasons and solutions:

The delete of the group was successful.

065 S HIGHLIGHTED FUNCTIONS ARE INVALID

Reasons and solutions:

Functions must exist on the Configuration File.

070 S AN ERROR OCCURRED UPDATING THE EXPORT FILE

Reasons and solutions:

Other than Normal occurred while updating the file. Check to see that the file is open and enabled, and, that no other user is updating the file.

075 S A GETMAIN FAILED

Reasons and solutions:

The system is short of storage.

080 I OK

Reasons and solutions:

The process finished successfully.

085 I USE FUNCTION KEY “5” TO UPDATE THE FILE

Reasons and solutions:

See 055 I, above

090 I ENTER AT LEAST ONE FUNCTION NAME

Reasons and solutions:

The group must have at least one Function.

095 C FUNCTION EXISTS IN ANOTHER GROUP

Reasons and solutions:

The highlighted Function name exists in another group. The Functions in each group must be unique.

100 C FUNCTIONS EXIST IN OTHER GROUPS OR ARE INVALID

Reasons and solutions:

Multiple errors exist. Highlighted Functions either exist in other groups or are not present in the Configuration file.

3.11 IMPORT TYMEAC TABLES

Transaction Id: NN30, Program: NN30PGM1

The transaction adds Functions with related Queue's to the Configuration File. The transaction does **NOT** replace Functions and Queue's that already exist. Once imported, the Group remains on the Export/Import File. Use the Export, delete option, (Section 3.10.4), to delete the Group.

Appendix 3.11.A contains a full list of the screen messages.

3.11.1 IMPORT GROUPS

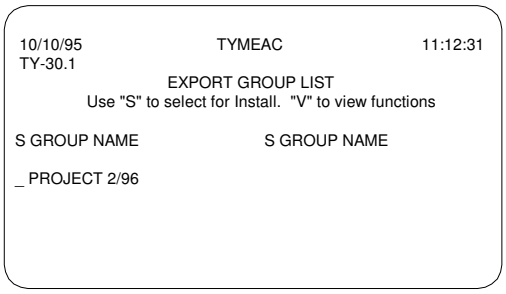


Figure 3.11.1 Import Groups List.

S: Selection:
'V' To view the Functions contained within the group. (Section 3.11.2).
'S' To import the group to the Configuration File.

Group Name:
The name of the Group.

3.11.2 IMPORT FUNCTIONS LIST

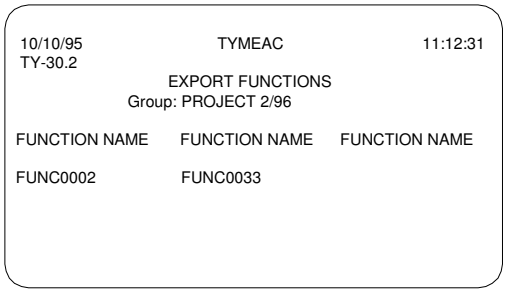


Figure 3.11.2 Import Functions.

Function Name:
The names of the TYMEAC Functions associated with the Group.

APPENDIX 3.11.A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN30 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S NO GROUPS EXIST

Reasons and solutions:

The Export/Import file is empty.

010 S EXPORT/IMPORT FILE ERROR

Reasons and solutions:

A non Normal return code occurred while accessing the file.
Check to see that the file is open and enabled.

015 S WRITE OF A TEMPORARY STORAGE RECORD FAILED

Reasons and solutions:

The program uses temporary storage to save intermediate results.
Check for out of space conditions or system damage.

020 S READ OF A TEMPORARY STORAGE RECORD FAILED

Reasons and solutions:

The program uses temporary storage to save intermediate results.
Check for out of space conditions or system damage.

025 I THIS IS THE FIRST PAGE

Reasons and solutions:

During back paging, this is the first page.

030 I THIS IS THE LAST PAGE

Reasons and solutions:

During forward paging, this is the last page.

035 S THE EXPORT/IMPORT FILE IS CORRUPTED

Reasons and solutions:

A NotFound condition occurred while browsing for Function records. The Group Header record contains the number of Functions associated with the group. The count therein does not match what was found on the file. Delete the file and rebuild.

040 S THE CONFIGURATION FILE IS CORRUPTED

Reasons and solutions:

The Function header record, count field does not match what was found on the file.
Delete the file and rebuild.

045 S AN ERROR OCCURRED UPDATING THE CONFIGURATION FILE

Reasons and solutions:

A non Normal return code occurred during the update process.
Delete the file and rebuild.

050 S A GETMAIN FAILED

Reasons and solutions:

The system is short of storage.

055 S xxx QUEUES AND yyy FUNCTIONS ADDED. zzz GROUP INSTALLED

Reasons and solutions:

xxx is the number of Queues added.
yyy is the number of Functions added.
zzz is the name of the Group installed.

3.12 LIVE QUEUE MODIFICATION

Transaction id: NN31, Program NN31PGM1

This is the Main Storage display/modification of a Queue Table.

TYMEAC stores the start-up values for the Queue Elements in the Configuration File. The Configuration File Maintenance Transaction is available to alter these values. However, these elements only load at TYMEAC Start Up.

This transaction provides a way to change Queue Element fields without a Shut Down / Start Up sequence. It also provides a way to change the status of Queue Tasks.

See the Configuration File Maintenance Transaction, Queue Maintenance, (Section 3.2.3), for a detailed description of the Queue Elements.

Altering values herein has an immediate effect on performance. Changing the New Task Thresholds while the Queue is under a heavy load is a way of tuning performance. The effect of the weighted factor, calculated on paper, may not perform as planned.

This transaction, in a test environment, gives designers an invaluable tool with which to test performance parameters under many conditions to establish the final values for a production system.

This transaction, in a production system, gives system personnel the tool necessary to respond to temporary, changing conditions.

TYMEAC provides a second program, NN31PGM2, that is conversational and allows Queue names in lower case. The transaction id is NN3A.

Appendix 3.12.A contains a full list of screen messages.

3.12.1 LIVE QUEUE MENU

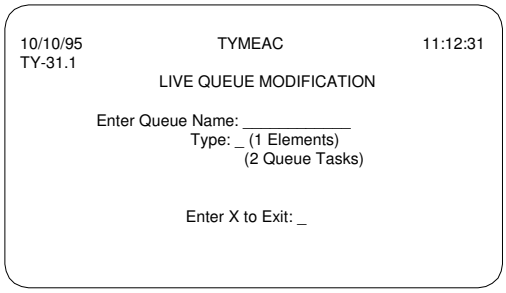


Figure 3.12.1 Live Queue Menu.

Queue Name:
Enter the name of the Queue for display / modification.

Type:
1. For a Queue Element. (Section 3.12.2).
2. For a Queue Task list. (Section 3.12.3).

Exit:
Enter any character to exit to or use the Clear Key.

3.12.2 LIVE QUEUE ELEMENTS

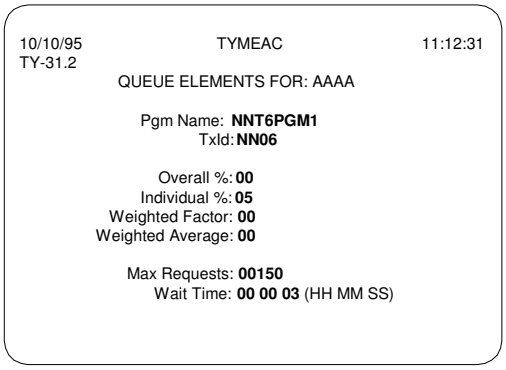


Figure 3.12.2 Live Queue Elements.

Pgm Name:
The name of the user-written Processing Application Program to which the TYMEAC Queue Task LINKs to process the user request. Over type to modify.

TXID:
The CICS® transaction identifier for this Queue. Over type to modify.

New Task Thresholds:

Overall **%:**	Values 00-99. Over type to modify.
Individual **%:**	Values 00-99. Over type to modify.
Weighted Factor**:**	Values 00-99. Over type to modify.
Weighted Average**:**	Values 00-99. Over type to modify.

Max Requests:
The maximum number of consecutive requests processed before restarting the Queue Task. Over type to modify.

Wait time:
The Hours, Minutes, Seconds for the interval on the WAIT Command issued when no work is pending. Over type to modify.

3.12.3 LIVE QUEUE TASKS

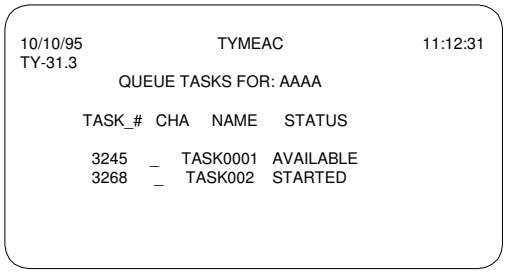


Figure 3.12.3 Live Queue Tasks.

Manually changing the status of any Queue Task may cause undesirable side effects.

Task #:
The CICS® task number associated with the Queue Task. The Queue Task stores the task number when it starts executing and sets it to nulls before returning to CICS®. If no task number appears, then no task is active for the Queue Task.

Task Name:
This is the constant 'TASK' plus a four digit sequence number.

Status:
The status of the Queue Task:

Available	Available for a new task. No CICS® task is processing this Anchor Point entry.
Processing	A CICS® task is actively processing this request.
In link to Appl	The Queue Task issued a CICS® LINK to a Processing Application Program.
In link to Schd	The Queue Task issued a CICS® LINK to the Scheduler program for the Output Agent.
Posted	Another task posted the Queue Task's ECB.
Started	Another task issued a CICS® START for a new task.
Canceled	The Monitor task determined that a problem may exist.

Disabled	The entry may not accept any new request. See Queue Task Status Update transaction, Appendix 3.5.A for a description of how an entry becomes disabled.

CHA:
Change the status of the Queue Task. Acceptable values are:

A - Make available for a new CICS® task.

C - Set to Canceled. The Monitor Task changes this status to 'disabled' when next it runs. However, if the Monitor Task is not available, then this status has no affect.

D - Set to disabled.

Full Shut Down cannot complete while Queue Tasks are active (waiting for posting, processing, In-link, In-schd), or about to be active (Posted, Started). The possibility of a storage violation is unacceptable. Setting a new status of 'A' or 'D' enables full Shut Down to complete.

A Queue Task Status cannot change to itself, i.e., Available may not become Available.

The CICS® task number is for a Master Terminal Purge or Force, if necessary.

The status may change at any time. This may flag an error condition.
The transaction supports paging for Queues longer than that which fit on one screen.

F5 is to update the current display only. To update multiple Queue Tasks on multiple screens, use F5 for the current display, page forward or backward and use F5 for that screen.

APPENDIX 3.12 A

SCREEN MESSAGES

MESSAGE STRUCTURE

NN31 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

005 S SYSTEM CONFIGURATION FILE ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S SYSTEM MAIN TS QUEUE ERROR

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that the system was not started

015 S SYSTEM IN SHUT DOWN MODE

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid.

020 I ENTER A QUEUE NAME

Reasons and solutions:

Enter a Queue Name.

025 I INVALID QUEUE NAME

Reasons and solutions:

The Queue Name is invalid.

030 S INVALID TYPE

Reasons and solutions:

Valid type is “1” or “2”.

035 C ERRORS DETECTED

Reasons and solutions:

The change is not valid. See above for valid values.

040 I NO ERRORS DETECTED

Reasons and solutions:

All information is correct. Use the F5 key to update.

045 I OK

Reasons and solutions:

The last request processed successfully.

050 C THE STATUS CHANGED

Reasons and solutions:

Before the requested change could be made, the status of the Queue Task changed. Try again, if necessary.

055 I THIS IS THE FIRST PAGE

Reasons and solutions:

This is the first page of Queue Tasks..

060 I THIS IS THE LAST PAGE

Reasons and solutions:

This is the last page of Queue Tasks..

CHAPTER 4 TYMEAC OPERATION

This chapter is the 'how to use' of the TYMEAC System. It explains the parameters passed to the Request Broker from your Server Application Program. It explains the parameters passed from the Queue Task Program to your Processing Application Program. It details the Request Broker and Scheduler return codes. Finally, it explains the sample programs loaded at installation.

4.1 TYMEAC USAGE

TYMEAC functions as a client/server pair. The client requests a Function and the server services the request. TYMEAC supplies two COBOL / C copy, (include), files, (members):

NNINCOMM is the Common Area, filled in by the client application program, for LINKing to the Request Broker.

NNTCOMMA is the Common Area, filled by the Queue Task program, for use by the Processing Application Program.

For the C language, substitute the underscore, (_) separator in place of the COBOL dash, (-), separator.

The name of the Request Broker program is NN04PGM1. Communication with the Request Broker is by a CICS® LINK with a Common Area (NNINCOMM).

The input parameters for the Request Broker, filled in by the application program follow the scenario:

Execute this Function, A,
using this optional input message, B,
for this type, C,
at this priority, D,
wait for completion no longer than this interval, E,
returning an optional result, F,
a completion code, G,
and, for non waiting requests, Return Data, H.

WHERE:

A: **NNIN-FUNC-NAME**
The name of the Function previously defined in the Configuration File Maintenance Transaction, (Section 3.2.4).

B: **NNIN-ADDR-INPUT**

NNIN-LENGTH-INPUT
Is the address of the input message area and the length of the input.

The address is a Pointer value. This area may come directly from the terminal input area, or, may be obtained by a GetMain.

The length is a 32 bit signed binary number.

N.B. The use of an input area, (output area as well), restricts access to a single affinity. However, front/back-ending the program that LINKs to TYMEAC is unrestricted. One may use a Distributed Program Link to a program which GetMains an input area, LINKs to the TYMEAC Request Broker and reverses the procedure upon return to the caller. See Section 6.1, Affinity. See also the note, below, on input areas.

C: **NNIN-TYPE**
Type is a one byte alphanumeric field reference by 88 level names in COBOL and #defines in C.

NNIN-TYPE-NON-QUEUE (Value '1') is for Non Queue processing. The Request Broker does not schedule this type but LINKs to the associated Processing Application Program, (Section 3.2.4). The Request Broker ignores priority and interval.

This type is for compatibility with existing systems, for use in a test region to monitor a new program before it is ready to execute in an asynchronous task environment, and for isolation, (see Section 7.1, Isolation).

NNIN-TYPE-QUEUE-WAIT (Value '2') is for Synchronous Request processing, (Queue, with a wait for completion). The Request Broker schedules all Queues in this Function with the same priority number and waits for all Queues to complete no longer than the interval amount, below.

NNIN-TYPE-QUEUE-NO-WAIT (Value '3') is for Asynchronous Request processing, (Queue without a wait for completion). The Request Broker schedules all Queues in this Function with the same priority number and immediately returns to the caller.

D: **NNIN-PRIORITY**
Priority is a 5 digit, unsigned decimal number. An invalid priority defaults to 1. A priority greater than that which is valid on a Queue, within the Function, defaults to the maximum priority on that Queue. A priority of zero bypasses the use of Wait Lists, therefore, when no Queue Task is waiting for an ECB post, then the Scheduler rejects the request.

Priority is for the Wait Lists. When no Queue Task is available to immediately process the request, the request goes into a Wait List corresponding to this priority. Priority 1 goes into the first Wait List. Priority 2 goes into the second Wait List, etc. The Queue Task processes Wait Lists from the first to the last, always exhausting all entries in the first before moving on to the next.

E: **NNIN-INTERVAL**
Interval is valid only for TYPE-QUEUE-WAIT, above. This is the maximum

time the Request Broker waits for Queue's to finish processing after scheduling all Queues. The Request Broker gives asynchronous tasks every opportunity to complete. If overall timing is important than do so in the requesting task, at the higher level. The higher function times the lower function. Lower functions do not time themselves.

Interval is a 6 digit, unsigned decimal number in the format Hours, Minutes, Seconds. An invalid (non numeric) interval defaults to 1 second. An otherwise invalid interval returns a non-zero Request Broker completion code, (Section 4.2).

F: **NNIN-ADDR-OUTPUT**
NNIN-LENGTH-OUTPUT
Address of the output message and length of the output.

The address is a Pointer value. The length is a 32 bit signed binary number.

For Functions not returning a reply, the address is null and the length zero.

See the note, above, (input area). See also the note, below, (output areas).

G: **NNIN-RC**
Section 4.2, at the end of this section, details completion codes. Zero is normal. However, this is the result of the Request Broker/Scheduler, not the result of application processing. The output message of each Processing Application Program must contain the completion codes for application processing.
See the note, below, (output areas).

H: **NNIN-ASYNC-RETURN-DATA**.
This is the information necessary to track the request in the TYMEAC system, only. This information is necessary when using the Request Status transaction, (Section 3.9). The information is only valid for a single affinity of TYMEAC and only for the time from TYMEAC Start Up and before TYMEAC Shut Down, (see Section 6.1, Affinity).

NNIN-REQID
The request id for Asynchronous Requests that successfully schedule is the Temporary Storage Queue name used to control processing in the asynchronous tasks. The name is a fifteen digit number, in character format, starting at one at TYMEAC Start Up, and incremented by one for every non waiting request.

The highest generated number, since Start Up, is viewable on the Main Storage Table Display (Section 3.4.1).

NNIN-TSU-ABS
This is the TYMEAC Start Up value from a CICS® ASKTIME, ABS Command at TYMEAC Start Up. The value stored here, for non-waiting requests, is a fifteen digit number in character format.

NNIN-APPLID
This is the CICS® APPLID from an ASSIGN Command, issued by the Request Broker in the CICS® Image where it runs. The value stored here, for non-waiting requests is eight characters.

The input parameters for the Processing Application Program, filled in by the Queue Task program, follow the scenario:

Execute your method,
using this optional input, A
returning an optional result, B
and, optionally, save your GetMained work area address here, C.

WHERE:

A: **TCOM-ADDR-INPUT**
TCOM-LENGTH-INPUT
Is the address of the input message area of the requesting task, (actually a copy thereof). The address is a Pointer value. The length is a 32 bit, signed binary number.

B: **TCOM-ADDR-OUTPUT**
TCOM-LENGTH-OUTPUT
Is the address of the result message. The address is a Pointer value. The length is a 32 bit, signed binary number.

This address must come from storage outside the program's dynamic storage. GetMain is an example of storage obtained outside of the program's dynamic storage, and is the preferable method. The Queue Task program takes this output area and either moves it to a shared storage area or a temporary storage item and then issues a FreeMain, NoHandle for the output area. It is highly recommended **NOT** to use a Pointer to any area outside the application programs' control, i.e., any locate mode file or database Pointer, or, any shared storage that might be in use by another task.

C: **TCOM-WORK-PTR**
Is an optional Pointer value for use by the Processing Application Program to save work areas between LINKs from the Queue Task program. Set this Pointer using a GetMain, **without** the Shared option.

The Queue Task program sets this Pointer to NULL the first time it LINKs to the Processing Application Program and does not alter this Pointer throughout the life of the task. The Configuration File Maintenance Transaction, (Section 3.2.3), has a field called Max_Number_Requests. This is the maximum number of times the Queue Task program LINKs to the Processing Application Program before restarting itself even with requests pending. This restart ends the task and frees all the accumulated storage obtained by CICS® on behalf of the transaction. It also syncpoints the transaction and frees any strings and threads.

MAX_NUMBER_REQUEST, together with the optional work area Pointer is a way to process multiple requests during the life of an asynchronous task.

However, another field in Queue Maintenance is, Max_Wait_Time. This is the maximum time the Queue Task WAITs when no work is pending.

With a value of zero, the Queue Task never issues a wait and when no work is pending, the task ends.

With a value greater than zero, with Max_Number_Requests not expired, and no work pending, the task suspends with a WAIT until an ECB posting makes the task dispatchable. Storage, strings, and threads held by the task remain during this period.

Input Areas:

There is a single input area and output area for all Queues within a Function. Each Processing Application Program must parse the input area for information relevant to its process.

The demonstration programs are a simple example.

The input area contains values for A1, A2, A3, B1, B2, B3, C1, C2, C3.

NNT6PGM1 only uses the A values and ignores B and C.
NNT6PGM2 only uses the B values and ignores A and C.
NNT6PGM3 only uses the C values and ignores A and B.

Output Areas:

The output message area returned to Synchronous Requests, or, passed to the Output Agent Task for Asynchronous Requests, contains the combined output from all Queue Tasks in the order in which each Queue Task finished processing. When an abnormal condition arises such as time-out, the output message contains only the output from the Queue Tasks' that finished.

The application program must parse this output area for information necessary to build a reply for the receiving party. This includes any completion codes required by the application. In a multitasking environment there is no overall completion code for all asynchronous threads. Each process should complete and not depend on another process to handle any clean up or bypass execution. The initiating application program may take further action depending on the completion codes present within the output area.

The Demonstration programs, (Section 2.2), are a simple example of an output message. Each program uses an eight character identification, (for program NNT6PGM1, the identifier is 'A---TEST'), followed by an internal return code, (RC=0000), followed by unique information concerning the processing of that program.
See also Section 4.3, Samples.

4.2 REQUEST BROKER/SCHEDULER RETURN CODES

These are the four digit codes, returned in field, NNIN-RC, from the LINK to the Request Broker. Numbers beginning with the digit four (4) relate to the Request Broker. Numbers beginning with the digit five (5) are from the Scheduler.

Common return codes:

0000 - Normal

4030 - Shut Down in progress.
4120 - Time-out occurred.
5020 - No task available to process request.
5025 - No Wait List available.
5030 - CICS® START for a new task failed.

RETURN CODES. REQUEST BROKER:

4005 Read for the Configuration File failed.
Check to see that the file is open and enabled.

4010 Read for the main temporary storage record failed with a 'not found' or 'queue id' error.
Check to see that the Start Up program has run.

4025 Read for the main temporary storage record failed with an error other than above.
Check for system damage.

4030 Shut Down in progress.
The system is in Shut Down mode.

4035 Invalid Function name.
The Function name passed, (NNIN-FUNC-NAME), is not in the Function Table.

4040 Invalid Type code.
The type of process requested, (NNIN-TYPE), is not 1, 2, or 3.

4045 For Non Queue type processing (Type 1), no Processing Application Program name was found in the Function Table.
Usually a type of processing error. The administrator set up the Queue for Queue type processing, (Type 2 or 3), and the user attempts to use Non-Queue.

4050 An internal GetMain failed.
Check for system damage or short on storage conditions.

4055 The LINK to the Non-Queue Processing Application Program failed.
Check to see that the program is available.

4060 For Queue type processing (2), either the Function Table contained a program name or the address of the queue list is null.
Usually a type of processing error. The administrator set up the Queue for Non-Queue type processing, (Type 1), and the user attempts to use Synchronous Request processing.

4100 The LINK to the Scheduler failed.
Check for system damage, that the Scheduler program name in the Configuration File is correct, (Section 3.2.2), and the program is ENABLED.

4110 The wait interval is invalid.

The interval passed to the Request Broker was invalid, (NNIN-INTERVAL). This is a six digit number in the format Hours, Minutes, Seconds.

4120 Time-out occurred.
For Synchronous Requests, the wait interval expired before all Queue Tasks finished. The output area is unpredictable, it contains only the output of those tasks that finished.

4135 For Asynchronous Request processing (3), either the Function Table contains a program name or the address of the queue list is null.
Usually a type of processing error. The administrator set up the Queue for Non-Queue type processing, (Type 1), and the user attempts to use Asynchronous Request processing.

4140 Number of Queues in the Function Table is invalid.
Check for system damage (storage overwrite on the Function Table).

4165 Write for a temporary storage queue failed.
For Asynchronous Request processing, associated data go into a temporary storage queue. The write for this queue failed. Check the Log File for the exact reason for the failure.

4170 The LINK to the Scheduler failed.
Same as 4100, above.

4185 An ABEND occurred.
Check the abend code and related storage for the reason.

RETURN CODES SCHEDULER:

5005 Read for the Main Temporary Storage Queue failed.
Check for system damage and that the system is available.

5010 The Queue Name associated with this Function is not in the main table.
Every Queue in the system is in the Main Table. Check for system damage.

5015 Shut Down in progress.
The system is being Shut Down.

5020 No task is available to process the request.
No tasks are processing the Queue, (busy), or are about to process the Queue, (started, posted), and no task is available to start or post. All tasks are probably 'disabled'. Determine the reason for the disable, correct the problem, and reset the disabled tasks with the Disabled Queue Update transaction, (Section 3.5).

The reason for the disable may be because the system is in Shut Down mode. Queue Tasks, when shutting down Immediately, set their status to disabled before ending the task.

5025 No Wait List entry is available .
For non-priority Wait Lists: There is no available Wait List entry in any Wait List.
For priority Wait Lists: There is no available Wait List entry in the requested Wait List, nor in any Wait List subsequent to the requested Wait List.

5030 Similar to 5020, above, however, an attempt was made to start a new task and the start failed. Check for system damage.

5055 Reset processing failed.
Reset processing attempted during a back-out failed. This is a very small window that results from the reset attempt coming just as the Queue Task finishes, but before the Queue Task can post the requester's ECB. This message only appears in the Log File. See Section 7.2, Stall Table for a detailed discussion of back-out processing.

5060 No Output Agent task is available to process the request.
This is the same as 5020, above. However, the Queue is the Output Agent. Even though this Queue may become available at the time it requires scheduling, the possibility exists, now, that the Asynchronous Request will stall.

5065 No Output Agent Wait List entry is available.
This is the same as 5025, above. However, the Queue is the Output Agent. Even though a Wait List entry may become available at the time it requires scheduling, the possibility exists, now, that the Asynchronous Request will stall.

4.3 SAMPLES

TYMEAC provides source code for the demonstration system for customer use.

Program NNT7PGM1 is the Output Agent Processing Application Program for Queue 'DDDD'. The code writes the combined output of associated asynchronous tasks to a file (NNTEST), or passes it on to a communication type application (NNT8PGM1). This is a general format of how one may do it. The code provides a working example that is available for use as a skeleton for application development.

Program NNT6PGM1, 2, 3 are Processing Application Programs for Queue's 'AAAA', 'BBBB', and 'CCCC' respectively. The programs form a simple output message using selected data from the input message.

Program NNT6PGM4 is the sample nested Processing Application Program example. This program forms a simple output message using selected data from the input message. It LINKs to the Request Broker for further asynchronous processing and appends that output to the current output message.

Program NNT5PGM1 is the sample Processing Application Program for Non-Queue processing. There is no logic difference between this program and the other application programs. With TYMEAC, application programs do not concern themselves with their execution environment. They perform their function. Input to, and output from, are specific processing parameters.

Program NNT1PGM1 is the sample single event menu. This program is an example of setting up the parameters for the LINK to the TYMEAC Request Broker. Program NNT2PGM1 is the conversational version of this program.

Programs NNT3PGM1, NNT4PGM1 are the sample multiple event menu/driver. The menu accepts the number of drivers to start, starts that many driver tasks, displays the status of the driver tasks and shuts down the process. The drivers simulate the single event process for multiple TYMEAC Functions. This set is available to place a load on the system. Consequences of placing too heavy a load on the system are time-outs in the driver tasks, Wait Lists full, and no task available in Queue Tasks. System wide consequences include short on storage, stall purge, and driver task START's discarded due to maximum task constraints.

A flooded TYMEAC system simply rejects requests where resources are not available or excessive time elapses. A flooded CICS® system may purge driver tasks before they can complete. This set uses a temporary storage queue, name "TMTMTMTM". Purging this temporary storage queue, (e.g. CEBR), stops driver processing.

TYMEAC provides source code for the notification program for customer use. (NN61PGM1). This program reads the transient data queue for records written by the Monitor Task. Processing of that record is specific, and is the customer's option.

TYMEAC provides source code for the front-end Start Up, (NN01PGM1) and Shut Down, (NN99PGM1) programs for customer use.

TYMEAC provides copy code for all the sample programs and for TYMEAC Tables.

CHAPTER 5 RUN TIME

This chapter details the TYMEAC transactions employed mainly outside application development. These transactions are a complement to the application development transactions in Chapter 3.

5.1 TYMEAC START UP

The Start Up program is NN01PGM1. This program reads the Configuration File 'system' record for values used across all TYMEAC processing. It browses the file loading the Function Table, with related queue lists, and, the Main Table (of supported Queues). It builds the Queue Table for each Queue, with related Wait Lists. It builds the Stall Table, Temporary Storage Table and Request Table. It writes the main temporary storage record, starts the Monitor Task, and starts MIN_ACTIVE_TASKS, (see Configuration File, Queue Maintenance, Section 3.2.3), for each Queue.

The program writes fatal errors to the Log File and aborts the Start Up.

Enter the transaction id, NN01, on a terminal supporting terminal transaction initiation. It may be part of a PLTPI stage* where files are open and temporary storage is available. LINK to the program with or without a Common Area, (see NN00, below), or start in any other way.

Using transaction NN00.

TYMEAC supplies this transaction, (source code as well), for use in a test environment. Although, with simple modification, use in a production environment is desirable, (e.g., at PLTPI time it LINKs to the Start Up program and if the return code is unacceptable, it writes an operator message).

Enter the transaction id on a 3270 type terminal. The program LINKs to the Start Up program with a two byte Common Area. Upon completion of the LINK, it writes a status message the terminal:

START UP SUCCESSFUL
START UP NO GOOD.

For the former, the system is fully functional. For the latter, check the Log File for the exact reason for the failure.

* PLTPI programs run as CICS® tasks. For those systems that separate CICS® and USER storage, TYMEAC may not run as a PLTPI program. TYMEAC does not use the option, UserDataKey, on the GetMain for TYMEAC shared storage.

Since TYMEAC functions as a pure application, all tables must be available for modification by TYMEAC programs.

5.2 TYMEAC SHUT DOWN

The Shut Down program is NN02PGM1. This program reads the Configuration File, 'system' record. It reads the main temporary storage record and updates the record with 'in shut down mode'. It marks each Queue 'in shut down mode', writes statistics to the statistics transient data queue, and, if no transactions are active, frees all shared storage and deletes the main temporary storage record.

Enter the transaction id, NN02, on a terminal supporting terminal transaction initiation. It may be part of a PLTSD stage. LINK to this program with or without a Common Area, (see NN99, below), or, start in any other manner. The program may run several times until all Queue Tasks finish. This is the normal mode. The first time the transaction runs it marks each Queue, 'shut down when you are finished with all pending work'. Subsequent executions of the transaction mark each Queue, 'shut down immediately'.

Using transaction NN99.

TYMEAC supplies this transaction, (source code as well), for use in a testing environment, although, with simple modification, use in a production environment is desirable.

Enter the transaction on a 3270 type terminal. The program LINKs to the Shut Down program with a two byte Common Area. Upon completion of the LINK it writes a status message to the terminal:

INVALID READ FOR CFG FILE, SHUTDOWN ABORTED

Check to see that the Configuration File is open and enabled.

INVALID READQ TS FOR MAIN TSQ, SHUTDOWN ABORTED

If not an I/O error, than the TYMEAC system is not active.

INVALID REWRITE TS FOR MAIN TSQ, SHUTDOWN ABORTED

If a temporary I/O error, then rerun the transaction.

INVALID MAIN STORAGE ID, SHUTDOWN ABORTED

All TYMEAC shared storage starts with a character marker (Id).
The marker for the main storage area is invalid.

1. TYMEAC was not Shut Down before CICS® was Shut Down leaving the main temporary storage record. As a result of a warm or emergency restart the addresses within the main temporary storage record are now invalid.

2. TYMEAC storage is in error.

The program deletes the main temporary storage record and aborts further processing.

Depending on the operating system environment and invalid address may result in an ASRA abend or a TRAP. Manually purge the main temporary storage record with the CEBR transaction, if necessary.

SHUTDOWN STARTED, TASKS ARE STILL ACTIVE, TRY LATER

This is not an error. The Shut Down request is coming while TYMEAC processing is active. Queue Tasks are still active. Although these tasks check for Shut Down mode at every opportunity, this may require manual intervention. While tasks are active, TYMEAC cannot free storage.

N.B. The Live Queue Update Transaction, (Section 3.2.3), is available for just such a situation but care is advisable in using it. Do not mark a task 'disabled' while it is in LINK to a Processing Application Program that is not responding. If that program becomes active after the stall, it may cause a storage violation. Use the master terminal Purge or Force to kill the transaction first.

Once Shut Down starts, the Request Broker does not handle any new Functions. The Queue Tasks continue to process pending work until they exhaust the Wait Lists. The Scheduler may initiate new Output Agent Tasks. When Shut Down comes a second time the Queue Tasks end immediately regardless of pending requests.

SHUT DOWN COMPLETED SUCCESSFULLY, STORAGE FREED, TSQ DELETED

The TYMEAC system is fully Shut Down.

5.3 TYMEAC NOTIFICATION

The notification transaction is NN61, program, NN61PGM1, (entered in System Setup, Section 3.2.2).

The TYMEAC Monitor Task examines the internal tables looking for actual or potential problems. An actual problem is that all Anchor Points for CICS® tasks within a Queue are 'disabled'. A potential problem is that a task's processing or wait time is excessive. The Monitor Task writes a message describing the problem to this Transient Data Queue. The supplied skeleton program reads the TD Queue. Further processing is the customer's option.

Appendix 5.3.A contains the messages written to the transient data queue.

The layout of the Transient Data record is copy code NNTDQ.

Appendix 5.3.A

NOTIFICATION MESSAGES

MESSAGE STRUCTURE

The message identifier is that of the Monitor Transaction

NN03 NNN S T Q/S D Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

T is the date and time of the message.

Q/S is either the Queue Name or the Stall Table Entry Name.

D is a fifty-six character description of the error.

---

010 I LONG STALL TABLE ENTRY

Explanation:

The Stall Table Entry remains for more than thirty cycles of the Monitor Task. Entries in this table are for those Asynchronous Requests that exceed two cycles of the Monitor Task. The Stall Table is for manual intervention. This entry remains without intervention.

070 I NEW STALL TABLE ENTRY

Explanation:

Entries in the Stall Table are for those Asynchronous Requests that exceed two cycles of the Monitor Task. When the Monitor Task adds a new entry to the Stall Table, it writes this message.

020 I DISABLED QUEUE ANCHOR POINT

Explanation:

The Monitor Task set the Queue Task Anchor Point to ‘disabled’. The Log File contains the reason for the disablement.

030 I ALL TASKS ARE DISABLED

Explanation:

All Queue Task Anchor Points are ‘disabled’. No new CICS® task may process this Queue. The Scheduler rejects all requests for this Queue. The Disabled Queue Update transaction, (Section 3.5), is the means to reset these entries, after correcting the cause of the disablement.

5.4 TYMEAC STATISTICS

Statistics are available by three methods. At shutdown, on request, and remotely.

SHUTDOWN

Transaction: NN62
Program: NN62PGM1
Destination: NN62
File: NNSTAT

This is a transient data transaction, (entered in System Setup, Section 3.2.2). It reformats transient data queue records, written by the Shut Down program, (copy code NNSHUTDW), and writes print image records to the Statistics File, (copy code NNSHUTFL). A user program may physically process these records.

ON REQUEST

The On Request statistics transaction is NN63, program NN63PGM1. Enter this transaction on a 3270 type terminal or start it in any other way. It writes print image records to the Statistics File, (copy code NNSHUTFL), and may be run at any time.

When started from a terminal, it displays the following unformatted messages.

MESSAGE STRUCTURE

NN63 NNN S Where:

NNN is a unique number within the program.

S is the severity,

I - Information only
C - Conditional, Possible problem
S - Severe

---

000 I STATISTICS SUCCESSFULLY WRITTEN TO STATISTICS FILE

Reasons and solutions:

The request processed normally.

005 S CONFIGURATION FILE READ ERROR

Reasons and solutions:

A read for the System Record on the Configuration File failed.
Check to see that this file is open and enabled, or, not corrupted.

010 S TYMEAC NOT ACTIVE

Reasons and solutions:

The read for the main temporary storage queue record failed.
This may simply mean that the system was not started

015 S TYMEAC IN SHUT DOWN

Reasons and solutions:

The system is shutting down. TYMEAC may free main storage tables at any time, therefore, addresses may become invalid

020 S STATISTICS FILE WRITE ERROR

Reasons and solutions:

A write to the Statistics File failed.
Check to see that this file is open and enabled, or, not corrupted.

REMOTELY

The remote program is NN64PGM1. This program is available as a means for other systems, (outside the image where TYMEAC is running), to get a picture of the status of a TYMEAC image. An example of another system is a GUI processor that formats the return data into a chart. The communication between the GUI and this program may be a front-end program. Section 6.1 Affinity, discusses the use of front-end programs.

LINK to this program with a Common Area, (copy code NN64), as follows:

The length is forty (40) bytes.

Function: 02 bytes (see below)

Name: 24 bytes. For those Functions that require a name, this is the name of the TYMEAC Function or Queue. For generic Functions, the left justified characters to identify the desired group. (that is -- To retrieve statistics on all TYMEAC Functions beginning with ‘FUNC’, then the first four characters of name is ‘FUNC’, the remaining twenty characters are spaces.)

Return Code: 04 bytes (see below).

Filler: 02 bytes, unused.

Data length: 04 bytes, The 32 bit, signed binary length of the data area returned.

Data address: 04 bytes, The Pointer to the returned data.

Function Codes:

01 All statistics This includes the Number Generation Table, the Stall Table, the Temporary Storage Table, the Request Table, all Functions, the Main Table, and all Queues with Wait Lists.

02 Number Generation Table.

03 Stall Table.

04 Temporary Storage Table.

05 Request Table.

06 Function, single, name required.

07 Function, generic, name required.

08 Function, all.

09 Main Table, (list of Queue's).

10 Queue, single, without Wait List, name required.

11 Queue, single, with Wait List, name required.

12 Queue, generic, without Wait List, name required.

13 Queue, generic, with Wait List, name required.

14 Queue, all, without Wait Lists.

15 Queue, all, with Wait Lists.

16 Wait List, single, Queue name required.

17 Wait List, generic, Queue name required.

18 Wait List, all.

Return Codes:

0000 Normal.

0005 Configuration File Error.

0010 TYMEAC System is not active.

0015 TYMEAC System is in Shut Down mode.

0020 Invalid request Function.
0030 Invalid TYMEAC Function name.

0040 Invalid TYMEAC Queue name.

The data length is the total number of bytes, (lines times 80), of the data area.

The data area is a Pointer to task storage containing the requested information in the format of copy code, NNRS.

5.5 TYMEAC MONITORING

TYMEAC is a pure application. There are no hooks, no exits, nor system programs. This is in order that TYMEAC may run without interfering with any existing exit routine and may run on any future release of CICS® without modification.

Without accessing CICS® tables and using exit routines a monitor facility is necessary for the use of asynchronous tasks and shared storage.

The Monitor Task is NN03, program NN03PGM1. The Start Up program starts this task. The test environment, below, is the only other way to start this transaction.

The Monitor Task uses five locks, (CICS® ENQ).

GLOBAL_NAME ('TYMTYMSS')

This lock prevents multiple threads of this program as well as locking out the Start Up and Shut Down programs.

The Monitor issues this lock at task start and the lock ends with task end.

AREA_ENQ_NAME (Found within the fixed storage of each Queue).

This lock forces single thread processing of a Queue Wait List and the status indicator of each Queue Task within a Queue.

The Monitor issues this lock at the beginning of processing of each Queue and releases the lock at the end of processing of each Queue.

STALL_TABLE_ENQ (Found within the fixed storage of the Stall Table).

This lock forces single thread processing of the Stall Table.

The Monitor issues this lock at the beginning of processing of the Stall Table and releases the lock at the end of processing of the Stall Table.

TEMPORARY STORAGE_TABLE_ENQ (Found within the fixed storage of the Temporary Storage Table).

This lock forces single thread processing of the Temporary Storage Table.

The Monitor issues this lock at the beginning of processing of the Temporary Storage Table and releases the lock at the end of processing of the Temporary Storage Table.

REQUEST_TABLE_ENQ (Found within the fixed storage of the Request Table).

This lock forces single thread processing of the Request Table.

The Monitor issues this lock at the beginning of processing of the Request Table and releases the lock at the end of processing of the Request Table.

TSQ_NAME (The name of the temporary storage queue for non waiting tasks).

This lock is forces single thread processing of the temporary storage queue for Asynchronous Requests.

The Request Broker, Monitor, and Queue Task programs issue this lock before reading the temporary storage queue, item one record, and release the lock after rewriting the record. Also, the Queue Task programs issue this lock before adding item 2-n records, and release the lock when complete.

TYMEAC avoids the deadly embrace by all TYMEAC programs enqueuing in the same order and only one thread of this program running at any one time. TYMEAC supplies transactions for viewing/updating all TYMEAC tables

The Monitor Task examines the Stall Table, Temporary Storage Table and Request Table. The Monitor deletes each 'in-use' entry that is no longer valid, and, frees extensions to each table acquired during a heavy load.

The Monitor Task examines each Queue (called AREA internally). The program uses the Monitor Interval, (Configuration File, System record), or, one minute if no interval is present, as the time interval, (see Disabled Queue Update Transaction, Section 3.5).

The Monitor marks 'reset' for each Wait List entry for a non performing Queue i.e., no Queue Task busy, posted, started, or waiting, and, when an Asynchronous Request, it adds an entry the Stall Table.
The Monitor Task issues a CICS® START for itself using the Monitor Interval and ends the task.

The TESTING environment:

It is desirable not to use the automatic initiation of the Monitor Task in a testing environment. When employing real time program monitors with single stepping, TYMEAC monitoring may believe that a stall is present. However, using the clean-up facility of the Monitor Task is desirable.

On a 3270 type terminal, type the transaction id, a space, and the upper case word TESTING, (i.e., NN03 TESTING). The Monitor does not issue a CICS® START for itself.

CHAPTER 6 MISCELLANEOUS

This chapter deals with subjects of concern to the application development process.

6.1 AFFINITY CONSIDERATIONS

TYMEAC uses shared storage for communication between TYMEAC modules. Therefore, all TYMEAC processing must remain in the image in which TYMEAC starts. Generally this is an AOR. However, since TYMEAC is a pure application, there is no restriction to any type of image.

There is no restriction on application processing. The TYMEAC Queue Task Program LINKs to the user-written Processing Application Program passing a Common Area. This application program may use a distributed LINK or any other means to effect processing in another environment.

There is no restriction on the number of images in which TYMEAC may execute. TYMEAC tables load from a Configuration File. Using multiple files, one for each unique environment, or one for several environments, or any combination thereof is without restriction.

TYMEAC may reside in multiple application images. One may use a distributed LINK to access the Request Broker, by way of a front-end program, (see Section 4.1, setting up parameters). The Request Broker returns the CICS® Applid of the TYMEAC image for Asynchronous Requests. Request Status Transaction inquiries, (Section 3.9, Request Status Display, LINKing option), require this field.

The following is a brief example of using a front-end program:

> The application program forms a Common Area containing the parameters needed by the Request Broker, the input needed by the Queue Task Processing Application Programs, and the maximum output space allowed. The program LINKs to the front-end program.
>
> The front-end program forms a TYMEAC Common Area, GetMains an input area, moves the passed input to that area, and LINKs to the TYMEAC Request Broker.
>
> Upon return from TYMEAC, the front-end program moves the Return Data, or, output area back to the original Common Area.
>
> If this is a short running task, FreeMaining the explicitly GetMained input and output areas is not necessary.

TYMEAC is a means to separate complex requests into component processes, the object of which may ultimately execute anywhere along a distributed network.

6.2 TUNING

Tuning is a full time job for professionals and involves every parameter of the operating system, transaction server, applications, and network. This section is not a how to tune manual, but a what does TYMEAC do, and therefore, what should I look for in tuning. Four parameters, applicable to TYMEAC, are:

- Temporary Storage,
- Shared Memory,
- Maximum Tasks, and
- Wait Lists.

TYMEAC runs on all processors that support the transaction server. However, not all transaction servers work identically on all platforms, for all releases of the transaction server or operating system.

TEMPORARY STORAGE

TYMEAC uses temporary storage queues to pass the request to, and among, the Queue Tasks for processing Asynchronous Requests. TYMEAC endeavors to keep the temporary storage queue record lengths to no more than 4,000 bytes.

The Item 1 record contains the information necessary for each Queue Task to process the request. This record consists of an 80 byte fixed portion and a variable portion of 34 bytes for each Queue Task participating in the request. With 115 active Queue Tasks the length of the Item 1 record is under 4,000 bytes, (80 + (34 * 115) = 3990). This release supports up to 165 active Queue Tasks, therefore, the Item 1 record may reach a limit of 5,690 bytes. Considering the enormous load on the system having Functions with more than 115 Queue Tasks, this is not a normal situation.

The Item 2-n records contain the output of each Queue Task and no single Item record exceeds 4,000 bytes. For output greater than 4,000 bytes, the Queue Task segments the output into multiple Item records.

For systems with a high usage of Asynchronous Requests, increasing the strings for temporary storage is desirable, or such other means (File Manager threads) where applicable.
The Stall Table and Temporary Storage Table Display / Update transactions, delete function, purge the temporary storage queue associated with stalled requests. (Sections 3.6.3 and 3.7.3).

SHARED MEMORY

TYMEAC uses shared memory. All GetMains use the FLENGTH option and the request should come from the extended dynamic storage area. TYMEAC does not specify, UserDataKey, and PLTPI restrictions apply for Start Up.

TYMEAC's principle is that no task may touch the storage of another task.

The Request Broker copies input areas, for use by individual Queue Tasks, to shared storage. Each Queue Task copies the input area (shared) to its storage chain (task).

For Synchronous Requests, the Request Broker frees the input area (shared) and the Monitor Task, when running, frees this input area (shared), when all Queue Tasks finish copying the area. The Request Table Update transaction, (Section 3.8.3), delete function, also frees this area.

For Asynchronous Requests, the Queue Task frees the input area (shared) when the last Processing Application Program finishes. The Temporary Storage Update, (Section 3.6.3), and Stall Table Update (Section 3.7.3), transactions, delete function, also free this area.

The Queue Task program places output areas for Asynchronous Requests into temporary storage queues.

The Queue Task copies output areas, for Synchronous Requests, into shared storage from the Processing Application Program. The Request Broker copies individual output areas (shared) to its storage chain (task) and frees the shared storage. The Monitor Task, when running, frees this area when an excessive time interval elapses. The Request Table Update (Section 3.8.3), transaction, delete function, also frees this area.

The Stall, Temporary Storage, and Request Tables are dynamically expandable. Start Up initializes these tables for an average load. The algorithm for locating an available entry is 'first available'. When the executing program no longer needs this entry, the executing program marks it 'available'. When no entry is available in the current table, then the executing program initializes a new table and chains it to the previous. The Monitor Task frees totally unused tables beyond the first.

The Shut Down Program, when no tasks are active, frees all TYMEAC shared storage.

MAXIMUM TASK, MAXIMUM ACTIVE TASK

The New Task Thresholds, (see Configuration File Queue Maintenance, Section 3.2.3), describes the algorithms used in determining when TYMEAC may start a new task. The START command may fail when exceeding Maximum Tasks. Hence, there is a very close correlation between the number of tasks for each Queue and the maximum supported by the system.

Maximum Active Tasks, or, Maximum Non Terminal Tasks, gives little warning of a problem. Real time monitors with alarms are very useful. One problem with many tasks waiting (for an ECB post) is that even when an ECB post takes place, if at maximum active tasks, the task does not resume execution. This excessive suspend time may result in a time-out in the Request Broker or a warning message from the Monitor Task written to the Log File. Hence, there is a very close correlation between the number of tasks for each Queue, the maximum time for waiting for new work for these Queue Tasks, and the maximum active tasks for the system.

Another area of concern is balancing the task/load parameter. Total_#_Tasks in Queue Maintenance, (Section 3.2.3), is the total number of Anchor Points for CICS® tasks supported by the Queue. This is relevant to the system wide MAX and AMAX, above. The total Anchor Points for all Queues must be less than the system wide maximums. When a new TYMEAC task fails to start, the TYMEAC Monitor Task marks the Anchor Point 'disabled' precluding any further use of this Anchor Point pending manual intervention, (see Disabled Queue Update transaction, Section 3.5).

The Shut Down, (Section 5.2), and On Request, (Section 5.4), programs write statistics to the Statistics File. The Detail Queue Display, (Section 3.4.4), screen under the Main Table Display, displays Queue details, (pictured below).

TASK	#_PROCESSED	#_WAITS	#POSTED	#_STARTED
1	5,000	15	4,900	10
2	4,000	10	3,900	8
3	3,000	5	2,900	4
4	100	1	30	2
5	0	1	0	1
6	0	0	0	0

This indicates that 12,100 requests processed by 4 tasks. Anchor Point 5 had 1 task started but the request processed by another task before it became active. Anchor Point 5 issued 1 wait but no new work came in before its wait interval expired. Anchor Point 6 processed nothing.

The algorithm, for task posting, is 'first available'. The usage, above, is usually heaviest at the top. However, what may appear as a lopsided usage in no way precludes that something is wrong. The status of higher order tasks may be 'in use' so that lower order tasks 'post' more often. The application process of some tasks may take longer than others. The general usage by task number in itself is of little importance.

The number 'started' of zero is significant. A prudent reserve is always wise and business rules change daily. However, after observing a zero usage over time, eliminating some

never used entries makes the display easier to read, reduces statistics and slightly reduces storage usage (128 bytes for each entry).

The number of 'waits' and number 'posted' are significant. The number of 'waits' is the number of time-outs i.e., wait interval expired with no handposted ECB. The Queue Task sets its status to 'available for posting', issues a DEQ, POST, ASKTIME, and WAIT. If the Request Broker handposts its ECB at anytime before the wait interval expires than there is no time-out.

Max_Nbr_Requests, (in Queue Maintenance, Section 3.2.3), is the maximum number of requests a Queue Task may process before restarting itself. The restart is simply a START command with protect, followed by a RETURN. This frees accumulated storage, strings, and threads. The correlation of 'processed', 'posted', and 'started' is dependent on this factor. For non string/thread holding applications, that use a wait interval, the number 'posted' should closely follow the number 'processed', and, the number of 'waits' should be low. However, an observation over time is necessary before making permanent changes.

The Live Queue Update transaction, Element Modification screen, (Section 3.12.2), is a way to experiment without making permanent (DASD) changes. The New Task Thresholds, Maximum Number of Requests and Wait Interval are alterable and effect an immediate change.

WAIT LISTS

Each Queue must have at least one Wait List. The Configuration File, Queue Maintenance, (Section 3.2.3), defines the number of Wait Lists and number of Wait List entries available for each. The Scheduler, when starting a new task, always puts the request into a Wait List. This is so that the first task looking for work may process the request, rather than making the request wait for the new task to become active.

Wait Lists may be priority lists. The request, (the Common Area passed to the Request Broker, field, NNIN-PRIORITY, in Section 4.1), specifies the priority of the request and when no task is immediately available to process the request, the Scheduler puts it into its respective Wait List. Priority 3 goes into Wait List 3, Priority 1 goes into Wait List 1.

Wait Lists may not be priority lists. Using two or more Wait Lists when the request always specifies priority 1 is an overflow technique.

The Scheduler attempts to put the request into the Wait List corresponding to its priority. However, when that Wait List is full, the Scheduler puts the request into the next higher Wait List. This is a primary overflow. When that Wait List is also full, the Scheduler puts the request into the next higher Wait List. This is a secondary overflow.

The statistics record primary and secondary overflows at Shut Down, (Section 5.2), and On Request, (Section 5.4). The Wait List screen under the Main Table Display transaction, (Section 3.4.5) displays these overflows.

For Non-Priority Wait Lists:

WL	Times_used	Overflow_pri	Overflow_sec

1	10,000	100	0
2	90	0	10
3	10	0	0

This indicates that 10,100 requests processed. The Scheduler tried to put all 10,100 requests into Wait List 1. This succeeded 10,000 times. 100 requests rejected because the Wait List was full. The Scheduler tried to put the 100 overflows into Wait List 2. 90 were successful but 10 times Wait List 2 was also full and those requests went into Wait List 3.

For Priority Wait Lists:

WL	Times_used	Overflow_pri	Overflow_sec
1	25	0	0
2	10,000	75	0
3	75	0	0

This indicates that 10,100 requests processed. 25 requests were priority 1 and all processed as priority 1. The Scheduler attempted to put 10,075 requests into priority 2. This was successful 10,000 times. 75 requests overflowed from Wait List 2 and successfully processed from Wait List 3.

One may use the statistics to observe a trend before it becomes a problem. One may increase of decrease the number of Wait Lists and/or entries available for Wait Lists. Each Wait List has a fixed portion of 96 bytes and 32 bytes for each entry.

All Wait Lists for a Queue have a fixed number of entries taken from the Configuration File. Using a variable number for each Wait List makes tuning extremely difficult.

Business rules change day to day. What was yesterday is not today will not be tomorrow. In reality, nobody monitors any system with extreme diligence. Using a variable number of entries for each Wait List means that similar Queues may not reflect similar behavior. When conditions change, the algorithm for determining what is a problem must be unique for the number of entry's factor. The fixed number makes the calculation simple, affords an 'at a glance' picture with numbers and charts, and a meaningful assumption about similar Queues is possible.

See also the note in Section 3.5, Disabled Queue Update, Appendix 3.5.A.

CHAPTER 7 PROBLEM DETERMINATION

This is not a chapter on how to fix everything that could ever go wrong. This chapter is about how to narrow the focus.

7.1 ISOLATION

The Non-Queue Function option, (see Section 3.2.4), eases testing and isolating erratic asynchronous tasks. This option is for those programs that may not execute as asynchronous tasks. This option is also available for testing and isolation.

Set up the Function with the program name as the Non-Queue Program. The Request Broker LINKs to this program directly without scheduling an asynchronous task using the same parameters, (Common Area), as for an asynchronous task. Therefore, debugging tools such as EDF are available.

Trapping asynchronous tasks is more complex. Each debugging tool, provided by different vendors, traps in unique ways. Therefore, the method depends on the tool. Generally, the trap is at the program name. The task id is too general and may be part of another user's work. The task number requires an attached task.

Non-Queue programs and asynchronous programs use the same Common Area layout. Therefore, isolating, as above, is the recommended method for suspect application programs.

7.2 STALL TABLE

The Stall Table, (Section 3.6), is an in-doubt mechanism for Asynchronous Requests.

The first in-doubt situation arises during initial scheduling.

The Request Broker adds an entry to this table during back-out processing, when in-doubt. Back-out occurs when a Queue, within a Function, fails scheduling. The Request Broker informs all previously scheduled Queue's to ignore processing. If the request is in a Wait List, then the Scheduler marks the Wait List entry 'reset'. If the request is currently executing, then the Scheduler informs the Queue Task Program to ignore output processing. In the small window between the time the application program finishes processing but before the Queue Task Program updates the temporary storage record, the Request Broker is in doubt whether it can safely release resources. The Request Broker places an entry in the Stall Table. The Monitor Task cleans-up such an entry when the Queue Task updates the temporary storage record. See Section 5.5, Monitoring.

An example is helpful. Function, FUCN0006, in the Demonstration System, (Section 2.2), uses two Queue's, 'AAAA' and 'BBBB', intended for Asynchronous Request processing, and an Output Agent Queue, 'DDDD'.

A request for FUNC0006 comes into the system. The user-written Server Application Program forms the parameters needed by the TYMEAC Request Broker, (Section 4.1), and LINKs to the Request Broker. When all TYMEAC Queues successfully schedule, the Request Broker returns to the user-written Server Application Program. The application program may inform the requester that the request scheduled and free the connection for other work.

However, the following may occur: The Request Broker determines that the Function consists of two TYMEAC Queues. The Request Broker saves the input area in shared storage and writes a temporary storage queue, item one record, containing parameters for the TYMEAC Queue Tasks. The Request Broker LINKs to the TYMEAC Scheduler for Queue 'AAAA'; scheduling complete. The Request Broker LINKs to the TYMEAC Scheduler for Queue 'BBBB'; scheduling fails, (see Section 4.2, Scheduler Return Codes).

This Function cannot complete successfully. A back-out of all previously scheduled Queues is desirable either to avoid unnecessary processing when the request is in a Queue's Wait List pending execution, or, to limit the amount of post-application processing done by the Queue Task.

In this example, Queue 'AAAA' is the only previously scheduled Queue in need of back-out processing. The Request Broker LINKs to the TYMEAC Scheduler to back-out Queue 'AAAA' with the 'request name' equal to the temporary storage queue name. If the Scheduler finds the name in a Wait List, that Wait List entry is 'reset' and the back-out is successful. If the Scheduler finds the name in a Queue Task Anchor Point, (copy code, NNAREA, see also Section 1.4, Figure 1.4.4), the Scheduler nulls the Anchor Point temporary storage 'request name', to ignore further processing by the Queue Task, and the back-out is successful.

When a successful back-out occurs, the Request Broker deletes the temporary storage queue, frees the input area from shared storage, and informs the requesting Server Application Program of the reason for the failure.

When an unsuccessful back-out occurs, the Request Broker cannot free the request's resources, (temporary storage queue and shared storage for the input area), without the possibility of jeopardizing system integrity. That is, a Queue Task may be reading the shared storage input area or updating the temporary storage queue. The stage of processing for the Queue is unknown.

TYMEAC uses shared storage for tables and ENQ's to preserve integrity. TYMEAC ENQs on the temporary storage queue name for Asynchronous Requests before updating or adding output areas, (items 2-n), thereto. This may simply be a case where one task is locking a resource and another task is waiting on that lock before it updates the resource. This is only a problem in that the Request Broker cannot free valuable system resources at this time. Therefore, the Request Broker adds an entry to the Stall Table.

When the Queue Task for Queue 'AAAA' finishes processing the request, it determines that Queue 'BBBB' finished, no further Queues remain for the request, and the request is 'in error'. Therefore, the Queue Task frees the input area and deletes the temporary storage queue.

The entry in the Stall Table remains. However, no resources for the request remain. When the Monitor Task runs, it deletes the Stall Table Entry.

The second in-doubt situation arises during the Monitor Task scan of the Temporary Storage Table.

The Monitor Task places an entry in the Stall Table when in-doubt. TYMEAC is a process now system. The Monitor Task examines the Temporary Storage Table (copy code, NNTSTBL, Section 1.4, Figure 1.4.5), for entries remaining for a lengthy period and adds an entry to the Stall Table when "this has been here much too long" (The field, 'TimesChecked' determines this lengthy period, (see Section 3.7.2, Temporary Storage Table Browse)). An example of an entry remaining for a lengthy period is a situation caused by an abend, a CICS® task purge, or a stall in a thread outside the CICS® image.

When the Asynchronous Request is flagged due to lengthy processing time in applications, (that is, the Request is not stalled, it is just taking a long time), then this Stall Table Entry should be ignored. The Monitor Task eventually cleans-up this entry.

When the Asynchronous Request is flagged due to a failure to schedule the Output Agent Queue, then recovery and re-scheduling are possible.

The Output Agent Queue is the TYMEAC Queue that processes the combined output of all the previously executed asynchronous component processes of the Function.

The last Queue Task to finish processing schedules the Output Agent Queue. However, when scheduling fails, (Section 4.2, Scheduler Return Codes), the Asynchronous Request stalls. The last Queue Task to finish processing frees the input area for the Function. However, this Queue Task cannot free the temporary storage queue, containing the output of the Function's component processes.

The Monitor Task, after a lengthy period, puts the stalled request into the Stall Table and writes messages to the Log File and transient data Notification queue.

After the system administrator resolves the problem with the Output Agent Queue, the Stall Table re-schedule option, (Section 3.6.3) is available to re-schedule the Output Agent Queue.

Part of the processing of the Request Broker is to check the status of the Output Agent Queue for Asynchronous Requests.

After scheduling each component process of the Function, the Request Broker LINKs to the Scheduler for the Output Agent Queue, not to schedule this Queue, but to make sure that the Queue's Anchor Points are available for Output Agent Tasks and at least one Wait List entry is available. The Request Broker does this to avoid a stall when the Queue Task attempts to schedule the Output Agent Queue.

Refer to the **example**, above, for Function, FUNC0006. Assume Output Agent Queue, 'DDDD', has one Anchor Point and twenty-five Wait List entries.

Unknown at this time, the PPT target program, NN07PGM1, for the Queue 'DDDD' Output Agent Task, NN07, is incapable of execution.

One hundred requests for Function, FUNC0006, come into the system. The Request Broker successfully schedules the first thirty requests. The first Queue Task to finish processing the component processes, (Queue's 'AAAA' and 'BBBB'), LINKs to the Scheduler for the Output Agent Queue, 'DDDD'. The Scheduler places the request into the first Wait List entry and a issues a CICS® START for task, NN07, on Anchor Point, 0001. The Scheduler marks the status of this Anchor Point, 'started', however, the PPT target program, NN07PGM1, cannot execute.

The next twenty-four Queue Tasks to finish processing LINK to the Scheduler for the Output Agent Queue, 'DDDD'. The scheduler places each request into a Wait List entry. Since the status of the only Anchor Point is 'started', meaning that a CICS® task is about to process the Queue, the Scheduler does no more. All twenty-five Wait List entries for Queue 'DDDD' are 'in-use' and no processing is taking place.

Since no Wait List entry is available, the Request Broker, having successfully scheduled the first thirty requests, now rejects all following requests with a return code of "No Wait List Available for Output Agent Queue", (Section 4.2).

The Request Broker scheduled thirty requests.

Twenty-five requests are in the Wait Lists of Queue 'DDDD'. The next five Queue Tasks to finish processing the component processes of the Function fail scheduling for the Output Agent Queue, 'DDDD'.

The Monitor Task finds all thirty requests in the Temporary Storage Table and determines that these requests "have been here much too long". The Monitor Task adds thirty entries to the Stall Table and sends messages to the Log File and transient data Notification queue.

The Monitor Task determines that Anchor Point, 0001, on Queue 'DDDD' has been in a 'started' status "for much too long". The Monitor Task changes that status to 'canceled', and writes a message to the Log File. The next time the Monitor Task runs, it changes the status from 'canceled' to 'disabled', and writes messages to the Log File and transient data Notification queue.

The system administrator resolves the problem with program NN07PGM1.

The system administrator uses the Disabled Queue Update transaction, (Section 3.5.1), to reset the status of Anchor Point 0001 in Queue ‘DDDD’ from ‘disabled’ to ‘available’. This transaction also determines that Queue ‘DDDD’ has twenty-five pending requests in Wait Lists and STARTs a new CICS® task, NN07, to process these requests.

The system administrator uses the Stall Table re-schedule option, (Section 3.6.3), to manually re-schedule the five remaining Queue ‘DDDD’ requests.

When the Asynchronous Request is flagged due to a processing, (abend), or task, (flush), failure, then the system administrator must use every available tool to determine the stage of application processing.
The transactions, Stall Table Display, (Section 3.6), and Temporary Storage Table Display, (Section 3.7), are available to view and selectively delete, (and re-schedule for the Stall Table), entries in these tables.

7.3 ABENDS

The Queue Task Program handles abends in that it marks the task Anchor Point entry in the Queue 'disabled' and writes a message to the Log File. The Queue Task Program LINKs to the user-written Processing Application Program. Handling abends on this level is the user's responsibility.

The Request Broker handles abends for Queue processing to free shared storage and delete the temporary storage queue for Asynchronous Requests.

7.4 SHARED STORAGE and LONG RUNNING TASKS

Section 6.2, Tuning, details TYMEAC's use of shared storage. Application programs use of shared storage, especially for long running tasks, is the user's responsibility.

The long running task, in a transaction processing system, requires consideration.
For Queue Task applications, the variable Max_Nbr_Requests, in Queue Maintenance, (Section 3.2), is the total number of requests the Queue Task may process before ending the task. The Queue Task Program LINKs to the user-written Processing Application Program Max_Nbr_Requests times, issues a CICS® START for itself, and ends the task even with requests pending in the Wait Lists. The task-end frees strings/threads and accumulated task storage over which the application programs have no control. This variable controls the long running Queue Task in Queue Maintenance and is dynamically alterable in the Live Queue Update transaction, (Section 3.12).

The TYMEAC Queue Task, Request Broker, and Scheduler free all explicitly GetMained storage. Variables do not control the Request Broker/Scheduler like the Queue Task program, above. It is the user's responsibility to control the length of the task that invokes TYMEAC.

7.5 SCHEDULING FAILURES

Scheduling failures come under four categories:

- Time-out,
- No available Wait List,
- No available task, and
- Resource failure.

Time-out, for Synchronous Requests, means that all components (Queues) of the request must finish within the interval requested. If, at the moment of expired time, the last Queue finishes, then there is no time-out. The time-out interval is unique for each request and is the user's responsibility to adjust. For peak processing periods, such as those experienced with seasonal business, an increase in the wait time may be sufficient. However, the time-out may be due to problems related to excessive time in application programs, i.e., waiting for strings/threads, locks, or other resources. The Queue Detail and Wait List Displays, in the Main Table Display Transaction, (Section 3.4), give a real time picture of the processing state. The On Request Statistics Transaction, (Section 5.4), writes the entire TYMEAC system picture to a file.

No Available Wait List may occur for several reasons. Section 6.2, Tuning, details TYMEAC's Wait List processing. The number of entries in Wait Lists may be inadequate for which the solution is to increase the number of entries. The number of Wait Lists, when used as an overflow mechanism, may be inadequate during peak requirements, for which the solution is to increase the number of Wait Lists. The overflow mechanism also is true for prioritized Wait Lists. For priority 3, the request goes into Wait List 3. If Wait List 3 is full and there is no Wait List 4, the Scheduler rejects the request regardless of the status of Wait Lists 1 and 2. An analysis of priority as well as number of, and number in, Wait Lists is necessary.

No Available Task is the condition when no task is available for posting, (waiting), or is actively processing a request, (busy), or is about to process a request, (started or posted), or no Anchor Point is available to start a new task, (available). The status of all tasks is 'canceled' or 'disabled'. The Disabled Queue Update Transaction, (Section 3.5), describes how task Anchor Points become 'disabled' and is the means to reset the disabled status. The Live Queue Update transaction is also available to alter the status of Queue Tasks, (Section 3.12).

Resource Failure is that a CICS® resource, usually a GetMain, is not available. The exact reason for the failure is in the Log File.

TYMEAC uses temporary storage to pass the Asynchronous Request parameters to, and among, asynchronous tasks. The name of this queue comes from the Generation Table and is a fifteen digit number beginning with one (1). When CICS® starts with warm or emergency, the remaining temporary storage file may cause a conflict.

Index

—O—

—P—

—Q—

—R—

—S—

—T—

—U—

—W—