# TIER3

# Client/Server Development

**January 2009**

This manual describes how to develop and fine-tune Tier3 Client/Server applications. Tier3 provides the infrastructure necessary to develop high performance, mission critical, applications while affording developers maximum latitude in determining the functionality that each application is to provide.

| | |
|---|---|
| **Revision Update Information:** | This manual supersedes the Tier3 Client/Server Development manual V3.0 |
| **Operating Systems:** | VAX/VMS, Alpha/VMS, IA64/VMS |
| **Software Version:** | Tier3 Version 3.1 |

**Tier3 Software Ltd**

**January 2009**

The information in this document is subject to change without notice and should not be construed as a commitment by Tier3 Software Ltd. Tier3 Software Ltd assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.



(Many thanks to Franco Cravero for providing the discussion and coding examples in section 7.3)

# CONTENTS

# 1 OVERVIEW

Tier3 is a client/server system that enables remote client access to your server system resources. As the middleware residing between the client and server components of your application, Tier3 regulates and schedules client requests, allocates the resources necessary to satisfy those requests, and channels the resulting output back to the requesting client.

## 1.1 Introduction

The ability of a remote client to communicate over the network with a server on another node is the minimum requirement of every client/server application, but unfortunately it is not the only requirement. There are certain technical issues, peculiar to client/server development, that must be resolved before you can even begin to design a system to meet your user requirements.

Some of the questions that need to be broached are as follows:

- How will your server application manage the incoming network connections and what mechanism will be used to monitor the status of these connections?

- How do you restrict access to your server application to only authorized clients? Furthermore, how can you establish what rights and privileges each client is entitled to on the server node?

- How do you provide the multi-threading capability needed to allow your server application to deal with multiple client requests simultaneously?

- Is there a way to prevent distributed data corruption and inconsistency by encompassing all of the modifications to your distributed data sources under the ACID-umbrella of a true Two-Phase Commit?

Tier3 expedites client/server development by relieving your server programmers of the responsibility for performing these arduous tasks. Thus permitting them to concentrate exclusively on the application specific requirements.

## 1.2 System Components

This section discusses the main components of a Tier3 application and how these components interact with each other. To better understand this interaction you should appreciate that a logical division or demarcation line exists between the client and server components of any Tier3 application. This natural division affords a level of autonomy to both client and server developers.

For example, if the server application that you are developing will provide a calendar lookup facility then it is highly probable that there will be many and varied client applications that will need to avail themselves of these services. On the list of potential client applications could be: -

- A Diary application that, due to a unilateral decision made by the client development team, will execute on a PC platform so that the latest GUI development tools can be used. When scheduling monthly meetings, this application needs to obtain "First Working Day" information from the calendar server.

- An Archive facility currently executing as a Batch Job on a remote VMS node. This application needs to obtain "End of Period" information so that it can re-submit itself for the next archive run.

Regardless of the prospective client applications, your server developers need only be concerned with establishing the message formatting convention for both client request and server response messages, and ensuring that all client applications observe this convention.

### 1.2.1 Network Components

Both the DECnet and TCP/IP network transport protocols are supported in this version of Tier3. Therefore, the nodes that will host the client and server components of your application can be linked via either a DECnet or TCP/IP network.

### 1.2.2 Client Components

The developers of your client application programs are not constrained by any Tier3 specific Application Programming Interface, and communicate with Tier3 server applications using standard C/Java Socket or System Service APIs. It is the responsibility of your server developers to publish the interface requirements of their application, detailing the format and content of the messages that can be exchanged. (See Chapter 3 for more information on developing client programs).

#### 1.2.2.1 DECnet Non-transparent Task-To-Task Communication

Depending on your client system environment, the DECnet API can range from VMS System Services to the Pathworks Socket Dynamic Link Library. The DECnet programming documentation for your client operating system describes the API that must be employed by your client developers when communicating with Tier3 server applications.

#### 1.2.2.2 TCP/IP Sockets

Standard C or Java Socket Library calls are available on most platforms, and there is also a System Service interface provided with VMS. Other platform specific implementations such as Windows' ActiveX Controls may also be deployed. Please consult your client system documentation for details.

### 1.2.3 Server Components

The server component of each Tier3 application consists of a communication server process, a variable number of execution server processes, and a configuration file entry identifying the application's resource and security requirements on a particular node. The application specific functionality is provided by your server development team in the form of a shareable image containing the User Action Routines that Tier3 will call in response to significant events.

#### 1.2.3.1 Configuration File

You provide Tier3 with the information necessary to configure your application on a particular node, by inserting a record into the Tier3 Configuration File. This file is maintained by your system manager, and should reside in a common system directory so that it is accessible to all cluster nodes that will be required to host Tier3 server applications. (See Chapter 2 for more information on maintaining the Tier3 Configuration File).

#### 1.2.3.2 Communication Server

When you start a Tier3 server application, a detached communication server process is created. Each communication server is responsible for performing the following functions: -

- Advertising the availability of your server application by declaring itself as a Network Object (If using DECnet) and/or listening at the requested Port Number (If using TCP/IP).
- Ensuring that only authorized clients are permitted to access your server application.
- Monitoring the status of all client network connections.

- Providing a multi-threaded message exchange facility so that multiple incoming requests and outgoing response messages can be handled simultaneously.

- Allocating execution servers to action client requests.

- Dynamically adjusting the size of the execution server processing pool to meet client workload requirements.

Tier3 creates a separate communication server process for each server application that will execute on a given node thereby reducing inter-application resource contention and increasing request throughput.

#### 1.2.3.3 Execution Server

Execution servers are themselves detached processes that are created and governed by communication servers. All of the execution servers that are able to receive requests for an application at any given time are referred to collectively as the Execution Server Processing Pool.

Execution servers are responsible for accepting client requests, actioning those requests and directing their communication server to send the resulting response messages back to the requesting client. Tier3 provides the generic mainline shell that controls the processing cycle of all execution servers, regardless of the application. Your server developers are responsible for creating the User Action Routines that provide the application specific functionality.

Tier3 execution server processes are reusable on a transactional basis. Once a communication server has formed an association between client and execution server, it is completely at the discretion of your server developers as to how long the association will persist. For some applications server affinity may be maintained only long enough to satisfy a single client request, while for others you may choose to dedicate an execution server to an individual client until that client disconnects from the server application. (See Chapter 4 for more information on server development).

## 1.3 Preparing Programmers to Write Tier3 Applications

This section describes what your client and server programming teams must know in order to be able to write Tier3 applications. Included in this section is a suggested reading path that may assist programmers in navigating through this manual.

### 1.3.1 What Programmers Must Know

The knowledge requirements vary depending on whether a programmer will be involved in client or server development.

#### 1.3.1.1 Client Programmers

To be able to write client programs or subroutines, that are capable of accessing Tier3 server applications, programmers must have knowledge of the following:

- The client system application development environment.

- The DECnet or TCP/IP API for their client operating system and programming language.

- The message hand-shaking sequence that occurs when a client initially connects to a Tier3 server application.

- The format of the request and response messages that can be exchanged with the target server application.

Although it is not a requirement of Tier3, you may choose to hide the interaction with the underlying network protocol by encasing the DECnet or TCP/IP API calls in a series of subroutines that can be called from any language or development tool available on the client system.

#### 1.3.1.2 Server Programmers

To be able to write Tier3 server applications, programmers must have knowledge of the following:

- The VMS application development environment.
- A programming language that adheres to the VMS Procedure Calling Standard.
- The Tier3 execution server processing cycle.
- The Tier3 interface requirements of the User Actions Routines that they will be required to develop.
- The Tier3 System Services that can be called from your User Action Routines.

### 1.3.2 Suggested Reading Path for Programmers

After reading this chapter in full, programmers may find it useful to refer to the discussions in Chapter 9 on the Tier3 and hotTIP example programs, before returning to the chapters pertaining to application development.

Chapter 2 is intended primarily for use by system managers, but contains information that both client and server programmers should be aware of.

Chapter 3 describes what functions must be performed by client programmers in order to communicate with a remote Tier3 server application. Server programmers need not concern themselves with this chapter.

Chapters 4, 5 and 6 deal specifically with server development, although may be of some interest to client programmers.

Chapter 7 deals with hotTIP and is essential reading for all developers, database administrators and system managers who have an interest in distributed-database integrity. This chapter discusses how to enlist your VMS server application, and the data sources that it controls, into a distributed Two-Phase Commit transaction.

Chapter 8 discusses Tier3 Applet Uploaders including the benefits and drawbacks of using them to achieve a lightweight and extremely fast HTTP-server capability.

# 2 SYSTEM MANAGEMENT FUNCTIONS

This chapter describes the functions that need to be performed by the person responsible for managing Tier3 Application Servers at your site.

## 2.1 Registering Applications in the Tier3 Configuration File

Once your development team has created the User Action Routines that will respond to client requests for your application, you are ready to identify the application to Tier3 by registering it in the Tier3 configuration file.

To register a new Tier3 application, or modify the parameters of an existing application, you invoke the configuration file maintenance program by entering the following command: -

```
$RUN SYS$SYSTEM:T3$CONFIG
```

SYSPRV privilege is required to maintain the Tier3 Configuration File and only one person can be executing the maintenance program at any given time.

Tier3 locates the configuration file via the logical name T3$CONFIG_FILE, which by default translates to T3$MANAGER:T3$CONFIG.DAT. If this file does not exist when you run the configuration maintenance program, an empty file is automatically created.

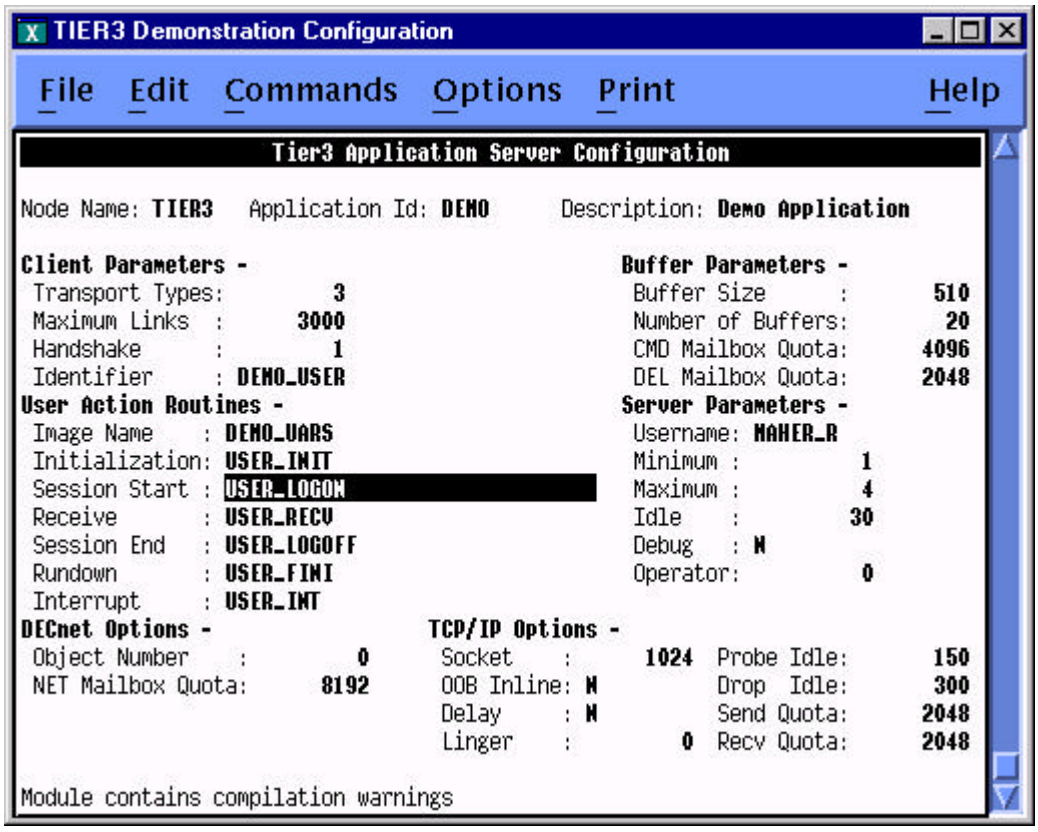
T3$CONFIG_FILE is an RMS indexed-sequential file with a combination of Node Name and Application Id as its primary key. Therefore, to locate the configuration record that you wish to modify or delete, you must enter both the name of the node on which the application will execute and the eight character string that uniquely identifies the application on that node. If the application does not exist in the configuration file, you will be asked whether you wish to create it. If the application does exist, you will be asked if you wish to modify or delete it.

The following table describes the keyboard function keys that are active during Tier3 configuration file maintenance.

| Function Key | Description |
|---|---|
| RETURN | Move to next field. If this key is pressed when the cursor is positioned on the last active field, your changes will be committed to disk |
| DOWNARROW | Move to next field. If this key is pressed when the cursor is positioned on the last active field, your changes will be committed to disk |
| CTRL/Z | Cancel processing. If this key is pressed when the cursor is positioned on a non-key field, any changes you may have made will be ignored any you will be re-prompted for the Application Id. If this key is pressed when the cursor is positioned on a key field, you will be returned to DCL. |
| UPARROW | Move to previous field. If this key is pressed when the cursor is positioned on either the Node Name or Description fields, you will be re-prompted for the same field. Press CTRL/Z if you wish to exit. |

Note: If you modify the configuration parameters for an application, they will not take affect until you stop and re-start that application's communication server.

### 2.1.1 Tier3 Application Configuration Parameters




#### 2.1.1.1 Application Identification

The first three fields on the configuration maintenance screen constitute Tier3 application identification.

##### 2.1.1.1.1 Node Name

Enter the SCSNODE name of the node on which the server application will run.

Tier3 allows you to take full advantage of your VMS Cluster environment by facilitating the fine-tuning of the server components of your application on a node-by-node basis. Therefore, you can adjust the Tier3 parameters to best suit the processing power and available memory of each host node. If you require all client connections for an application to be processed by a single node then create a configuration record for that node alone.

The Node Name parameter defaults to the SCSNODE name of the node on which the T3$CONFIG utility was activated.

##### 2.1.1.1.2 Application Id

The Application Id parameter uniquely identifies an application on a given node and must be specified as 1 to 8 alphanumeric characters, without leading or embedded spaces.

Each record in the Tier3 configuration file represents a separate instance of a communication server and execution server processing pool. As each Tier3 server instance acts independently of any other it can therefore be tuned independently of any other.

To illustrate, assume that node PERTH is a database server located at the Perth regional office of an Australian banking organization and has to run two Tier3 applications: TELLER and DISTRIB. The TELLER application must be designed and tuned to handle the high-volume transaction-processing environment of hundreds of bank teller PC clients requesting account debits, credits and mortgage repayments. Whereas the DISTRIB application need only deal with the periodic distribution of foreign exchange rate, and credit card lost/stolen information from Head Office.

Clearly one configuration could not adequately service the requirements of two such diverse applications. Therefore, Tier3 allows the system manager to specify configuration parameters that best represent the requirements of each individual application.

#### 2.1.1.1.3 Description

The Description parameter is a free format text field that you can use to store a more meaningful name or brief description of the application.

### 2.1.1.2 Client Parameters

#### 2.1.1.2.1 Transport Type

Transport Type is a bit-map field with each bit set indicating that the corresponding network transport is supported for this Tier3 application. Bit 0 is for DECnet and Bit 1 is for TCP/IP. Therefore, in this version of Tier3, the allowable values for Transport Type are 1-DECnet 2-TCP/IP or 3-Both.

#### 2.1.1.2.2 Maximum Links

The Maximum Links parameter limits the number of client tasks that can be simultaneously connected to the application's communication server on the given node. Specify a number in the range 1 to 8191.

When using DECnet you should be aware that the sum of the Maximum Links parameters for all applications on a given node should not exceed that node's executor MAXIMUM LINKS parameter specified via NCP.

#### 2.1.1.2.3 Handshake

The handshake protocol to use when accepting client connection requests. In this version of Tier3, the number 1 is the only valid value for the Handshake parameter.

This field is protected against user input.

#### 2.1.1.2.4 Identifier

Must be a valid VMS rights list identifier.

The identifier is used by the communication server to verify that a client task is authorized to access the application. All of the following conditions must be met for the communication server to authorize client access: -

- The Access Control Information buffer supplied by the client is 80 bytes long.
- The username does not exceed 12 bytes in length.
- The username appears in UAF.
- The account has not expired.

- The account has no secondary password set.
- The username has not been disabled using the AUTHORIZE command: *MODIFY username/FLAGS=DISUSER*
- The password has not expired.
- The password does not exceed 32 bytes in length.
- The password matches the password stored in the UAF file.
- The username has been granted the identifier required to access the application.
- No matching Intruder record exists for this client/username

If any of these conditions are not met then Tier3 will deny the client access to the application and disconnect the client's network link. The reason for each authorization failure is recorded in the communication server's log file.

You can use the Identifier parameter to limit a client's access to an application to a specific node. For example, to connect to the DEMO application on node BIGVAX a client may need to hold the SUPERVISOR identifier but other clients could access the same application on either the MEDVAX or SMLVAX nodes by holding only the PEDESTRIAN identifier.

#### 2.1.1.3 Buffer Parameters

### 2.1.1.3.1 Buffer Size

The Buffer Size parameter determines the maximum size of a message, in bytes, that can be passed between the client task and your User Action Routines. If your application will normally transfer 1000 byte messages but occasionally needs to transfer a message of 2000 bytes then you should set the Buffer Size parameter to 1000 and employ message fragmentation techniques when transmitting or receiving the larger messages. For more information on sending and receiving fragmented messages, refer to the RECEIVE User Action Routine, the T3$SEND system service, and the DECnet or Pathworks programming documentation for your client operating system.

Tier3 currently requires a minimum buffer size of 65 bytes for internal messaging. Therefore, you must specify a value in the range 65 to 60000 for the Buffer Size parameter.

Note: Messages transferred between the client task and your User Action Routines do not have to be padded or rounded up to the value specified in Buffer Size. If the client task has sent you a 20-byte message then Tier3 will present you with only those 20 bytes.

### 2.1.1.3.2 Number of Buffers

During initialization, the communication server uses the Number of Buffers and Buffer Size parameters to configure a buffer pool that is used, during processing, as a temporary storage area for all messages that are sent to client tasks. For performance reasons, the communication server transmits these messages asynchronously, that is, it does not wait for a remote node to accept a message before continuing to process additional messages. Therefore, for each message sent to a remote node, a buffer is removed from the buffer pool until the target node is able to acknowledge receipt of the message.

When specifying the value of the Number of Buffers parameter for a given application, you must therefore estimate the maximum number of simultaneous network write operations that the application may generate. If the communication server is experiencing delays in receiving notification that a remote End Communication Layer has received and acknowledged a message sent, then the value of Number of Buffers should be set accordingly.

If too low a value is specified then the communication server will experience buffer starvation and be unable to continue processing until at least one of the outstanding network write requests has completed.

#### 2.1.1.3.3 CMD Mailbox Quota

For each application, all messages generated by the execution server processing pool are placed in a single out-tray or queue, which is read by the communication server. The queue takes the form of a VMS Mailbox and you use this parameter to specify the amount of memory available to the mailbox. Specify the value in bytes.

When you call T3$SEND from your User Action Routines, two bytes of internal control information are attached to the message before it is queued for processing by the communication server. Therefore the minimum value that you can specify for CMD Mailbox Quota is the value of the Buffer Size parameter plus two. For example, if you specified 100 as the value for Buffer Size then the minimum value permitted is 102. The maximum value that you can specify for this parameter is 60000.

A large CMD Mailbox Quota can improve performance in some applications where there is a one-to-many relationship between messages received from, and messages sent to a client. By specifying the appropriate flag bits in the call to T3$SEND, you can direct Tier3 to return to your User Action Routine as soon as the message has been queued in the communications server's Command Mailbox and not to wait for the message to be sent.

If there is no room to buffer the message in the communication server's Command Mailbox when T3$SEND has been called, then the calling execution server will be forced to wait until the communication server has liberated enough space to hold the message.

Note: Command Mailbox space can also be occupied by Tier3 internal messages.

#### 2.1.1.3.4 DEL Mailbox Quota

VMS notifies each communication server of execution server process deletion via a separate mailbox called a Process Deletion Mailbox. The current length of the message is acc$k_termlen, or 84, bytes. Therefore to arrive at a value for this parameter you should multiply 84 by the number of execution servers that you expect to die simultaneously in peek load. 2000 bytes should cover most eventualities.

### 2.1.1.4 User Action Routines

#### 2.1.1.4.1 Image Name

The Image Name parameter specifies the filename of a shareable image that contains the User Action Routines that will be called by Tier3 during the life of an execution server. Do not specify a file type, or any file specification punctuation or wildcard characters.

If the shareable image does not exist in the SYS$SHARE directory then the name you specify for the Image Name parameter must be an Executive-Mode logical name in the LNM$SYSTEM logical name table. The shareable image must have been created before an application can be registered in the Tier3 configuration file.

#### 2.1.1.4.2 Routine Names

The next six User Action Routines parameters are all routine names that must exist in the shareable image specified in the Image Name parameter, and be declared as universal symbols.

Your server developers are responsible for providing the UAR names.

### 2.1.1.5 Server Parameters

#### 2.1.1.5.1 Username

Must exist in the VMS User Authorization File and have both read and execute access to the shareable image specified in the User Action Routine parameter Image Name.

Each execution server is created as a detached process running under this username via the sys$system:loginout.exe program. It is the responsibility of the system manager to ensure that all process quotas, rights and privileges specified in the UAF file for this username are sufficient to meet the requirements of the work to be performed in the User Action Routines. You should also be aware that the login command file for this username will be executed when each execution server starts up.

NB: Tier3 does not require this username to have any privileges.

#### 2.1.1.5.2 Minimum

The Minimum parameter specifies the minimum number of processes that the communication server must maintain in the execution server processing pool. Accordingly, it is also the number of execution servers that are created during application startup.

The Minimum server parameter cannot be greater than the Maximum Links client parameter.

#### 2.1.1.5.3 Maximum

The Maximum parameter specifies the maximum number of processes that can exist simultaneously in the execution server processing pool.

If no execution server is available when the communication server receives a request from a client, that is not already associated with an execution server, then that client is placed on the end of the server wait queue. At this time the communication server checks the size of the processing pool and if the value specified in the Maximum parameter has not been reached, it creates an additional execution server process.

If your application requires associations between client and execution server to persist for extended periods, you should consider specifying the same value for the Maximum server parameter as was specified for the Maximum Links client parameter. This configuration would effectively dedicate an execution server process to each client task and prevent clients from experiencing lengthy delays in the servicing of requests.

The Maximum parameter value must be greater than zero, cannot be less than the Minimum parameter value and must not be greater than the value specified for the Maximum Links client parameter.

#### 2.1.1.5.4 Idle

The Idle parameter specifies the number of seconds that an execution server can remain inactive, before being culled from the processing pool by the communication server.

After an execution server has terminated an association with a client task and called the SESSION END User Action Routine, it reports its availability to the communication server. The following table describes what subsequent actions are performed by the communication server depending on the value specified for the Idle parameter.

| Idle Value | Description |
|---|---|
| -1 | No further action is taken. The execution server is permitted to wait indefinitely for another association |
| ZERO | The execution server is culled from the processing pool, triggering the RUNDOWN User Action Routine to be called. |
| N | All positive values result in the communication server waiting "N" seconds before deeming the execution server to be superfluous, and culling it from the processing pool. If the execution server is needed to service an additional client request before "N" seconds have elapsed, then the communication server no longer considers it a target for culling. |

The Idle parameter has no further effect on execution servers once the size of the processing pool has been reduced to the value specified in the Minimum parameter.

#### 2.1.1.5.5 Debug

The Debug parameter is a boolean “Y” or “N” flag that you use if you wish to invoke the VMS Debugger from your User Action Routines. If this value is set to “Y” then Tier3 will activate a special version of the execution server code that has been linked with the /TRACEBACK qualifier. For security reasons you *should not* run Tier3 applications that have the Debug flag set to “Y”, in production.

#### 2.1.1.5.6 Operator

Specify the Operator parameter as a number in the range 0 - 12.

All abnormal terminations of execution servers or communication servers are automatically reported to any terminal enabled as a NETWORK operator terminal. The Operator parameter allows you to specify an additional, system manager defined, operator terminal type that should also be notified of abnormal terminations.

### 2.1.1.6 DECnet Options

If you have not chosen DECnet as one of possible network transports for your server application then the DECnet Options parameters will be protected against user input.

#### 2.1.1.6.1 Object Number

By default, if you leave this parameter as zero, Tier3 will construct a DECnet Object Name for your application by appending your Application Id parameter to the literal “T3_”. This is the Object Name that your client programs would use to connect to your server application when using the DECnet transport protocol.

If you wish Tier3 to advertise your application via a DECnet Object Number rather than an Object Name then you must specify this parameter as a non-zero value. Valid values are in the range 128 to 255.

You can therefore use the NCP command SHOW KNOWN OBJECTS, to verify whether or not an application is actively servicing DECnet connection requests on a given node.

#### 2.1.1.6.2 NET Mailbox Quota

DECnet communicates with Tier3 communication servers by sending network event messages to a VMS Mailbox. Use this parameter to specify the size of this mailbox in bytes. The larger the number of DECnet network connections that your application will manage the larger the value of this parameter should be.

Enter a value between 128 and 60000.

### 2.1.1.7 TCP/IP Options

If you have not chosen TCP/IP as one of possible network transports for your server application then the TCP/IP Options parameters will be protected against user input

#### 2.1.1.7.1 Socket

Enter the TCP/IP Port Number that you wish Tier3 to listen on for connection requests to your server application. Valid port numbers are in the range 1 to 65535.

#### 2.1.1.7.2 OOB Inline

When this parameter is set to “Y”, out-of-band data is placed in the normal input queue. No User interrupt data will be delivered to your INTERRUPT User Action Routine. This option corresponds to the Socket option OOBINLINE.

#### 2.1.1.7.3 Delay

This parameter corresponds to the TCP option NODELAY. When set to “N” the Delay parameter instructs TCP/IP not to delay sending any data in order to be able to merge packets.

#### 2.1.1.7.4 Linger

The Linger parameter specifies the number of seconds to delay closure of a socket if there are unsent messages in the queue. Corresponds to the Socket option LINGER.

#### 2.1.1.7.5 Probe Idle

Probe Idle specifies the time interval, in seconds, between Keepalive probes. Corresponds to the TCP option PROBE_IDLE

#### 2.1.1.7.6 Drop Idle

Drop Idle specifies the time interval, in seconds, after which an idle connection will be dropped. Corresponds to the TCP option DROP_IDLE

#### 2.1.1.7.7 Send Quota

The default for Send Quota is the larger of Buffer Size and 4096. This parameter corresponds to the socket option SNDBUF

#### 2.1.1.7.8 Recv Quota

The default for Recv Quota is the larger of Buffer Size and 4096. This parameter corresponds to the socket option RCVBUF.

## 2.2 Making Your Shareable Image Known to VMS

Tier3 can only access your User Action Routines if they have been made known to VMS by using the VMS INSTALL utility. For example, if you were running the Tier3 DEMO example at your site you may wish to enter the following commands into your site-specific startup command procedure.

```
$INSTALL ADD DEMO_UARS /OPEN /HEADER /SHARE
```

The /SHARE qualifier is mandatory; the /HEADER and /OPEN qualifiers are optional.

Note: This restriction does not apply if the Debug flag has been set to “Y” for the application.

### 2.2.1 Shareable Image Location

If the shareable image containing the User Action Routines for your application will not reside in the SYS$SHARE directory, you must define a trusted logical name that Tier3 can use to locate the image. For example, if you require your shareable image to reside in the DKA400:[DEMO] directory, you must enter the following command before attempting to register or start the DEMO application.

```
$DEFINE/SYSTEM/EXEC DEMO_UARS DKA400:[DEMO]DEMO_UARS
```

This logical name would need to be defined on every node that would host the DEMO application.

## 2.3 Starting an Application

To start a Tier3 application you first define a symbol for the T3$STRTSRV image, then run the utility using the appropriate Application Id as the argument. The following commands would start the DEMO application.

```
$T3_START :== $T3$STRTSRV
$T3_START DEMO
```

In order to be able to successfully start a Tier3 server application your process must currently have SYSPRV and DETACH privileges enabled.

The T3$STRTSRV utility can only start applications that have been registered in the Tier3 configuration file, as applications that execute on the node from which T3$STRTSRV utility was activated.

The T3$STRTSRV utility permits one trailing asterisk "*" to be appended to the Application Id. If this asterisk is present, then the Tier3 configuration file is searched for all applications with an Application Id beginning with the characters that were entered prior to the asterisk. For every matching record found, the corresponding application is started. If you do not specify an Application Id when you run the T3$STRTSRV utility than all applications that have been configured to execute on the issuing node, will be started.

Tier3 considers an application to be started, once it has successfully created a detached communication server process for that application. T3$STRTSRV outputs a message for each communication server it attempted to create with an indication as to whether or not the attempt was successful. If the attempt was successful then the message will contain the Process Id of the communication server. If the attempt failed, then a secondary message indicating the reason for the failure will also be output.

All communication servers execute under the user profile TIER3$SERVER. This account is added to UAF file when you install Tier3 on your system, and can be modified using the VMS AUTHORIZE utility. By default, TIER3$SERVER forces communication servers to execute at a process base priority of 6. Tier3 execution servers, on the other hand, are governed by the UAF entries for their respective usernames.

Tier3 constructs process names for communication servers by appending the Application Id parameter to the literal "T3_". For example, the communication server process for the DEMO application would have a process name of T3_DEMO. You can use the DCL command SHOW SYSTEM at any time to easily determine which Tier3 server applications are currently running on a given node.

Execution servers continue this process naming convention by taking the process name of their governing communication server and appending their processing pool index number. For example, if the DEMO application were configured for a maximum of 10 execution servers, each execution server process would have a process name of T3_DEMO_nnn where nnn would range from 001 to 010.

The following is an example of output generated by the T3$TSRTSRV utility.

```
$T3_START :== $T3$STRTSRV
$T3_START D*
%TIER3-W-CREFAIL, error creating communication server for DEBTORS
-SYSTEM-F-DUPLNAM, duplicate name
%TIER3-S-CRESRV, communication server created for DEMO, process id 000000B4
%TIER3-I-STRTSRV, successfully started 1 communication server
```

In this example, the system manager chose to start all applications beginning with the letter D. T3$STRTSRV found two applications matching the selection criteria, but failed in its attempt to start DEBTORS because a communication server process was already active for that application.

All network transports chosen to provide access to your servers must be running before you can start a Tier3 application. Therefore, if you wish to automate the activation of your Tier3 applications, you must ensure that your network startup procedures precede your Tier3 application startup procedures.

## 2.4 Stopping an Application

To stop a Tier3 application you first define a symbol for the T3$STOPSRV image, then run the utility using the appropriate Application Id as the argument. The following commands would stop the DEMO application.

```
$T3_STOP :== $T3$STOPSRV
$T3_STOP DEMO
```

In order to be able to successfully stop a Tier3 server application your process must currently have SYSPRV and DETACH privileges enabled.

The T3$STOPSRV utility permits one trailing asterisk "*" to be appended to the Application Id. If this asterisk is present, then the Tier3 configuration file is searched for all applications with an Application Id beginning with the characters that were entered prior to the asterisk. For every matching record found, the corresponding application is stopped. If you do not specify an Application Id when you run the T3$STOPSRV utility than all applications that are currently executing on the issuing node, will be stopped. (Tier3 will request confirmation before stopping any application.)

Tier3 instructs each communication server that has been chosen to shutdown by delivering a forced exit AST via the $FORCEX system service. Therefore an indefinite amount of time may elapse between issuing the request to shutdown, and the communication server actually dying. If necessary, and your application code permits such action, it is safe to issue immediate shutdown requests via the DCL command STOP/ID.

All Tier3 applications will terminate automatically when network shutdown notification is received. Individual applications will also terminate if a communication server detects a severe error.

## 2.5 Server Log Files

Two types of log file are created during the life of any Tier3 application. Both of these log files equate to the SYS$OUTPUT file of the detached process that created it, and can be scrutinized by the system manager while the process is still active.

The periodic deletion of historical log files should be incorporated into the normal housekeeping procedures of all Tier3 applications.

### 2.5.1 Communication Server Log Files

Each communication server process keeps a log of significant events, such as all network connections that it has had to deal with, in the following file:

```
T3$MANAGER:nodename_T3_appn.LOG
```

| Field Name | Description |
|---|---|
| nodename | The SCSNODE name of the node on which the communication server was created. |
| appn | The Application Id, identifying the application on whose behalf the communication server was created. |

Communication servers create a log entry in response to following events:

1. Initialization commencement
2. Initialization completion
3. Execution server creation
4. Execution server initialization completion
5. Execution server termination
6. Network connection acceptance
7. Network connection rejection
8. Client access authorization
9. Client access denial
10. Network link disconnections
11. Network shutdown notification
12. Image exit

You can interrogate a communication server log file by using the DCL commands TYPE or SEARCH to obtain such information as, the reason a certain user has been unable to access an application.

### 2.5.2 Execution Server Log Files

The filename for execution server log files is as follows:

```
userdef:nodename_T3_appn_nnn.LOG
```

| Field Name | Description |
|---|---|
| userdef | The default device and directory specified in the UAF file for the username specified in the Tier3 configuration file for the application on whose behalf the execution server was created. |
| nodename | The SCSNODE name of the node on which the execution server was created. |
| appn | The Application Id, identifying the application on whose behalf the execution server was created. |
| nnn | The processing pool index number of the execution server that created the file. |

Unlike other types of log file, what is written to an execution server log file is at the discretion of the developers of an application's User Action Routines. Any output that these routines direct to the process permanent file SYS$OUTPUT will be recorded in an execution server log file.

If necessary, Tier3 may also output error messages to an execution server log file.

# 3 CLIENT DEVELOPMENT

Client tasks communicate with Tier3 Application Servers using standard DECnet or TCP/IP Application Programming Interfaces (APIs). Consequently, absolutely no Tier3 specific software needs to be installed on the client node. This chapter provides a brief description of the functions that must be preformed by your client task when accessing a remote Tier3 application. For a more detailed explanation, you should refer to the TCP/IP, DECnet or Pathworks programming documentation for your client operating system.

For an example of a VMS client task that communicates with the DEMO application using DECnet, please refer to DEMO_CLIENT_DECNET.COB.

For an example of a VMS client task that communicates with the DEMO application using TCP/IP, please refer to DEMO_CLIENT_TCP_IP.COB.

## 3.1 Assigning a Network Channel

Before you can access a Tier3 server application you must first assign a network channel that can be used in subsequent calls to refer to the logical link between client and server.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$ASSIGN or LIB$ASN_WTH_MBX to assign a network channel to the _NET: device.<br>If it is a requirement of your application that the User Action Routines be able to send interrupt messages back to your client task, then you must supply the **mbxnam** parameter in the call SYS$ASSIGN. You create a mailbox by calling the SYS$CREMBX system service or you can call the LIB$ASN_WTH_MBX Run-Time Library routine to create a mailbox and associate it with the network channel in a single call. |
| | Socket | Call the SOCKET routine specifying the **domain** as AF_DECnet, the **type** as SOCK_SEQPACKET and the **protocol** as DNPROTO_NSP.<br>At this point, you may also wish to call the SETSOCKOPT and SIOCTL routines to alter any of the default options associated with the socket. |
| TCP/IP | System Services | Use SYS$ASSIGN to assign a network channel to the _BG: device followed by a call to SYS$QIO to create the socket and optionally alter any of the default options associated with the socket. |
| | Socket | Call the SOCKET routine specifying the **domain** as AF_INET, the **type** as SOCK_STREAM and the **protocol** as TCP.<br>At this point, you may also wish to call the SETSOCKOPT and SIOCTL routines to alter any of the default options associated with the socket. |

## 3.2 Requesting a Logical Link Connection

Once created, the network channel/socket is used to launch a connection request to the Tier3 server application at the remote node. If the connection request is rejected, then the reason for the rejection will be recorded in the target communication server's log file. (Note: For TCP/IP connection requests, a rejection status means that the Application is not currently running on the remote node. You are notified of an Application connection rejection by the failure of a subsequent SEND request.)

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$QIO (IO$_ACCESS). During initialization, your communication server will have declared itself as a network object, capable of receiving multiple inbound DECnet connection requests. If you entered a value other than zero for the Object Number in the Tier3 Configuration file, then that is the DECnet object number the communication server will have declared. If you entered zero for Object Number then the communication server declares a DECnet object name by appending the Application Id to the literal “T3_”. The task specification string that you supply in the Network Connect Block must contain the DECnet object name, or number, that was declared by the communication server for your target application. For example, 'PERTH::"TASK=T3_DEMO"' would be a valid NCB that could be used to connect to the DEMO application on node PERTH.<br>Tier3 currently ignores any optional user data that you may have specified when issuing the connection request. This may change in future versions. |
| | Socket | Use the CONNECT routine. The **sdn_objname** that you specify in the **destblk** argument must contain the DECnet object name, or **sdn_objnum** must contain the DECnet object number, that was declared by the communication server for your target application.<br>To prevent the DECnet address of the target node from having to be hard-coded into your client program, you can call the GETNODEBYNAME routine to convert the ASCII nodename of the target node into its binary network node address equivalent. |
| TCP/IP | System Services | Use SYS$QIO (IO$_ACCESS). In the remote socket declaration you specify the remote IP address of the node that is hosting the Tier3 server application, and the Socket (or port number) that was entered for the application in the Tier3 Configuration File. |
| | Socket | Use the CONNECT routine. In the remote socket name you specify the remote IP address of the node that is hosting the Tier3 server application, and the Socket (or port number) that was entered for the application in the Tier3 Configuration File. |

### 3.3 Providing Access Control Information

Once you have established a logical link connection with the target communication server, you need to provide Tier3 with the information necessary to verify that the client is authorized to access the chosen application. To do this you must ensure that the Access Control Information buffer is the first message that is transmitted to the communication server. (See section 3.4 for more information on sending data messages).

The **aci_buffer** is a fixed length 80-byte message consisting of two 40-byte fields. The first field being the username that identifies the user profile that the client has chosen to execute under on the target node, and the second being the password associated with that username. Each field must contain ASCII-coded text and may be null-terminated or space-filled. As VMS is the only supported server platform for this version of Tier3, you should limit the size of the username field to 12 bytes and the password field to 32 bytes.

To complete the hand shaking sequence the communication server will reply to a successful access attempt with the Tier3 identification buffer.

### 3.3.1 Tier3 Identification Buffer

If your application's communication server has accepted your logon request then it will respond with the **t3_id_buffer**. This 48-byte communication server acceptance message is comprised of the following information: -

| Name | Offset | Size | Description |
|---|---|---|---|
| T3_ID | 0 | 3 | Buffer Identification. Literal constant "T3$" |
| MAJ_VERS | 3 | 1 | Major version of Tier3 at the server node. Byte integer. |
| MIN_VERS | 4 | 1 | Minor version of Tier3 at the server node. Byte integer. |
| SCSNODE | 5 | 6 | SCSNODE name of the server hosting your Tier3 application. |
| LOGFAILS | 11 | 5 | Number of user login failures since last successful login attempt. Five-byte decimal integer. Tier3 increments this value for every unsuccessful client login attempt and clears it after a successful login attempt. |
| LAST_LOGIN_I | 16 | 16 | Timestamp of last interactive login. Decimal integer. The format for this field is Year 4 bytes, Month 2 bytes, Day 2 bytes, Hours 2 bytes, Mins 2 bytes, Secs 2 bytes and Hundredths of Secs 2 bytes. |
| LAST_LOGIN_N | 32 | 16 | Timestamp of last non-interactive login. This value will be reset after any successful login to a Tier3 application. The format for this field is the same as that for LAST_LOGIN_I. |

To obtain this information you must issue a receive request on your network channel. (See section 3.5 for more information on receiving data messages).

If the access control information provided by the client is invalid at the remote node, then the communication server will break the network connection. For VMS clients, your receive request will terminate with the error SS$_LINKDISCON. For other clients, your receive request may complete successfully with zero bytes transferred. The reason that the client was denied access to the application will be recorded in the target communication server's log file. (See section 2.1.1.2.4 for more information on authorizing client access to Tier3 applications).

Because the logical link between client and server is automatically broken when invalid access control information has been supplied, it is necessary to re-connect to the target application before attempting to transmit an amended **aci_buffer**.

## 3.4 Sending Data Messages

Tier3 places the following restrictions on the format and content of data messages that can be sent to your server application.

- The first message that you send must be the **aci_buffer**. This buffer is used internally by Tier3 to authorize client access to an application and will not be delivered to your RECEIVE User Action Routine.

- Zero byte messages are ignored by Tier3 and will not be delivered to you RECEIVE User Action Routine.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$QIO (IO$_WRITEVBLK) or (IO$_WRITEVBLK! IO$M_MULTIPLE). |
| | Socket | Use the SEND routine. |
| TCP/IP | System Services | Use SYS$QIO (IO$_WRITEVBLK) |
| | Socket | Use the SEND routine. |

### 3.5 Receiving Data Messages

Tier3 places the following restrictions on the format and content of data messages that can be received by your client application.

- The first message that your client task will receive will be the **t3_id_buffer.**
- Tier3 will not permit your User Action Routines to transmit zero byte messages.

Furthermore, it is the responsibility of the developers of the client and server components of your application to ensure that for every message generated by your User Action Routines, a corresponding receive request will be issued by the client task.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$QIO (IO$_READVBLK) or (IO$_READVBLK! IO$M_MULTIPLE). |
| | Socket | Use the RECV routine. |
| TCP/IP | System Services | Use SYS$QIO (IO$_READVBLK) |
| | Socket | Use the RECV routine. |

If your call to the RECV routine returns zero than the logical link has been disconnected. You can use the GETSOCKOPT routine at any time to determine the state of the logical link.

### 3.6 Sending Interrupt Messages

When you send an interrupt message to your server application, you are effectively requesting Tier3 to deliver an AST to the execution server process that you are currently associated with. Where possible, Tier3 honours that request by calling the INTERRUPT User Action Routine, in the appropriate execution server, at AST level.

Note that if the client that is sending the interrupt message is not currently associated with an execution server then the interrupt message will simply be ignored.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$QIO (IO$_WRITEVBLK! IO$M_INTERRUPT). |
| | Socket | Use the SEND routine with (MSG_OOB). |
| TCP/IP | System Services | Use SYS$QIO (IO$_WRITEVBLK!IO$M_INTERRUPT) |
| | Socket | Use the SEND routine with (MSG_OOB). |

### 3.7 Receiving Interrupt Messages

In this version of Tier3, the only interrupts that will be delivered to the client task will be those that were generated by your User Action Routines. Note that failure to accept interrupt messages that were generated by your User Action Routines can result in the communication server experiencing buffer starvation.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$QIO (IO$_READVBLK) – Mailbox Read. Your client application should have an asynchronous read pending on the channel to the VMS mailbox that was associated with your DECnet channel when it was initially assigned. When the mailbox read completes and the AST is fired, you can examine the **message-type** field to determine if the message contains an interrupt or link status information. Interrupt messages that DECnet places in the mailbox associated with your logical link have **message-type** of MSG$_INTMSG. |
| | Socket | Use the RECV routine with (MSG_OOB). |
| TCP/IP | System Services | Use SYS$QIO (IO$_READVBLK!IO$M_INTERRUPT) |
| | Socket | Use the RECV routine with (MSG_OOB). You can use the SELECT routine at any time to determine when an out-of-band message has arrived. |

### 3.8 Terminating a Logical Link

Although the Tier3 system service T3$SEND provides your User Action Routines with a mechanism for synchronously disconnecting a logical link, it is usually the client that decides when the link should be terminated.

If a client task terminates a logical link while associated with an execution server, then that execution server will be notified of the termination by receiving a system interrupt via the INTERRUPT User Action Routine.

If an execution server aborts while servicing a client request, then the application's communication server will automatically terminate the logical link to the client task.

Tier3 currently ignores any optional user data that the client may have specified when disconnecting the logical link. This may change in future versions.

| Network | API | Description |
|---|---|---|
| DECnet | System Services | Use SYS$DASSGN. Depending on the requirements of your application, you may choose to terminate a logical link by calling the SYS$QIO system service with the (IO$_DEACCESS ! IO$M_SYNCH) or (IO$_DACCESS ! IO$M_ABORT) function codes before calling SYS$DASSGN. |
| | Socket | Use the SCLOSE routine. |
| TCP/IP | System Services | Use SYS$DASSGN. |
| | Socket | Use the SCLOSE routine. |

When using the Socket interface on some operating systems, you must be sure to terminate any active logical links before your program terminates. If you do not, the links will remain active. All VMS User-Mode channels are automatically deassigned at image exit.

# 4 SERVER DEVELOPMENT

The development of the server component of your application involves creating a shareable image that contains the six User Action Routines that Tier3 will execute on your behalf, during the life of an execution server. Although Tier3 provides the generic mainline shell that controls when and how your User Action Routines are to be executed, the functions that each routine is to perform are purely at your discretion. Once the shareable image has been created, your system manager can register the application in the Tier3 configuration file. (See section 2.1 for more information on registering Tier3 applications.)

Tier3 does not impose a procedure naming convention on your User Action Routines, but for the purposes of this document the following names will be used when referring to each individual routine.

**INITIALIZE**
**SESSION START**
**RECEIVE**
**SESSION END**
**RUNDOWN**
**INTERRUPT**

Chapter 5 describes each of these User Action Routines in detail, while Chapter 9 discusses three examples of coding and building User Action Routines into a shareable image that can be registered in the Tier3 Configuration File.

Chapter 6 describes the Tier3 system services that are made available to your User Action Routines during execution server processing. Of the five Tier3 system services available to your UARs, T3$SEND is the only mandatory service call. This is because it is the mechanism used to inform Tier3 whether or not your code has completed the unit of work that it was performing on behalf of a client.

If you wish your UARs, and any resource managers they invoke, to participate in a distributed transaction then you should also consult Chapter 7 for complete details of how your UARs should interact with hotTIP.

## 4.1 Tier3 System Libraries

Tier3 moves three libraries to the SYS$LIBRARY directory at installation time: -

| Library | Description |
|---|---|
| T3$USER | Shareable image library containing links to the Tier3 system services. |
| T3$OBJECT | Object library containing Tier3 VMS message codes and external symbols. |
| T3$LIB | Macro library containing Tier3 symbol definitions. Useful for compile-time bit-flag manipulation. See example in BUILD_TIP_DEMO.COM. |

In order for your User Action Routines to be able to access the Tier3 system services, you must link against the T3$USER library when you are creating your shareable image. If you wish to have access to Tier3 external message codes and external symbol definitions then you should also link against the T3$OBJECT library. For example: -

```
$LINK/SHARE=UARS my_rtns, T3$USER/LIB, T3$OBJECT/LIB
```

The remainder of this chapter deals with the processing cycle of an execution server and describes the execution environment and restrictions that the developers of your User Action Routines need to be aware of.

## 4.2 Execution Server Processing Cycle

The following pseudo-code illustrates the processing cycle of an execution server and the order in which your User Action Routines will be called.

```
perform initialization procedures

call INITIALIZE User Action Routine

tell communication server we are available

get first client association or inactivity timeout

loop until we have been culled due to inactivity timeout

    call SESSION START User Action Routine

    open channel to client

    get first message for association

    loop until channel is closed

        call RECEIVE User Action Routine

        if channel is still open
            get next message for association
        end if

    end loop

    call SESSION END User Action Routine

    tell communication server we are available

    get next client association or inactivity timeout

end loop

call RUNDOWN User Action Routine

perform clean-up procedures

server exit
```

Due to the asynchronous nature of the **INTERRUPT** User Action Routine, it has been excluded from the previous pseudo-code. Although it is impossible to predict exactly when Tier3 will call this routine, you should be aware that interrupt messages are only sent to an execution server if that server is currently associated with a client task.

The T3$SETCTX service can be called from any User Action Routine, but a call to the T3$SEND service can only complete successfully during RECEIVE or INTERRUPT User Action Routine processing.

## 4.3 Execution Environment

This section describes the execution environment that will exist when Tier3 invokes your User Action Routines.

### 4.3.1 Process Initialization

Each execution server process is created under the user profile of the VMS username that was specified for the application in the Tier3 configuration file. For all intents and purposes you can assume that your User Action Routines are executing in a process that was created when this username logged on to the system.

Please be aware therefore, that the LGICMD file specified in the UAF for the application's username *will* be executed *before* Tier3 gains control of the execution server. This facility can be useful if you wish to change the output rate for your log files or set up an Xwindows display for a debugging session. But don't forget that the longer you take to yield control of the execution server to Tier3 the longer this execution server will be unavailable to service client requests.

### 4.3.2 Rights and Privileges

You must ensure that the username chosen to sponsor the execution servers for your application has been bestowed with sufficient rights and privileges to meet the requirements of the work to be performed in the User Action Routines. For example, if part of the functionality that your application is to provide involves taking out system wide locks via the $ENQ system service, and SMITH_J is the username that your application will run under, then SMITH_J would need to be given the SYSLCK privilege.

Although Tier3 verifies that a client is authorized to access an application before forwarding any requests from that client to your User Actions Routines, you may still need to supply additional application specific security checking. For example, assume that the application that you are developing is a file server that will accept a filename from a client, open that file and transmit the contents back to the requesting client. A problem arises in that although the client is authorized to access the file server application, this does not necessarily mean that he has read access to every file on the system.

One option for removing this potential for a security breach is to modify your User Action Routines to include a call to the VMS system service SYS$CHECK_ACCESS before attempting to open the requested file. If the error SS$_NOPRIV is returned, you could transmit an error message informing the client that you were unable to process the request. A far more elegant, robust and simpler, solution would be to have VMS change the persona of the server process to that of the requesting client before attempting to open the file. The VMS persona system services provide the mechanism to manage and switch personae within a process. There is one slight drawback with these services however, and that is that you would normally need to give the username that your servers are running under both SYSPRV and DETACH privileges. To eliminate the need for your servers to have to run with these heavy-duty privileges enabled Tier3 provides you with the T3$PERSONA_ASSUME (Alpha and IA64) system service. By calling T3$PERSONA_ASSUME your application developers can have the server process switch persona to that of the calling client without any additional privileges being required.

### 4.3.3 Execution Server Output

Any output that your User Action Routines direct to SYS$OUTPUT will be recorded in the execution server's log file. (See section 2.5 for more information on the location and content of execution server log files).

## 4.4 General Rules and Technical Notes

- Your User Action Routines can be written in any programming language that adheres to the VMS Procedure Calling Standard.

- Each User Action Routine is called with a CALLG instruction; must exit with a RET instruction and return a condition value in R0. If the condition value returned is other than SS$_NORMAL, then the execution server will terminate abnormally. Furthermore, if the low-order bit is clear in the condition value returned, then the communication server for the application will abort and trigger a forced exit of any remaining execution servers.

- All Tier3 arguments passed to your User Action Routines are maintained in User-Mode ready-only storage. Any attempt to modify their contents will result in an access violation. Copy any argument that needs to be modified into local working-storage and perform the modification on the local copy instead.

- If you are running on a VAX and any of your User Action Routines invoke User Written System Services, you must ensure that these services do not use any Executive-Mode dispatching codes in the range -3000 to -2900. These codes are reserved to Tier3.

- Your User Action Routines must not call a Tier3 system service from an access mode other than User-Mode.

- Tier3 reserves the global PSECT name T3$COMMON for internal use.

- If you need to disable AST delivery you must re-enable it before your User Action Routine returns control to Tier3. Tier3 relies on AST delivery to inform you of the arrival of user and system interrupts, and to force an execution server to exit when necessary. Disabling AST delivery interferes with the timely performance of these functions. Therefore, do not disable AST delivery unless it is absolutely necessary.

- Tier3 obtains any local event flags that it needs by calling the Run-Time Library routine LIB$GET_EF. To avoid potential conflict with Tier3, your User Action Routines should also use LIB$GET_EF if you need to allocate local event flags.

- If an execution server terminates abnormally while it is associated with a client task then the network link to the client will be broken. All network links to client tasks are broken automatically when the communication server terminates.

- Your User Action Routines should not attempt to accept input from SYS$INPUT.

- Communication server processes execute at the base priority that is specified for the TIER3$SERVER username in the UAF file, which by default is set to 6. As execution server processes execute at the base priority that is specified for the application's username in the UAF file, it is your responsibility to ensure that the execution servers for a given application will never have a higher base priority value than their governing communication server.

# 5 USER ACTION ROUTINE DESCRIPTIONS

## 5.1 INITIALIZE

The INITIALIZE User Action Routine is called only once by Tier3 as part of an execution server's initialization procedures. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** application_id ,buffer_size

---

### Arguments

#### application_id

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by reference**

The **application_id** is the address of an eight-byte string that contains the application identifier of the application on whose behalf the execution server was created. If the application identifier is less than eight bytes long, then the **application_id** string will be left justified and space filled.

#### buffer_size

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **buffer_size** argument is the address of an unsigned longword that contains the maximum Buffer Size for the application, as specified in the Tier3 configuration file.

---

### Description

The communication server creates execution servers, as the need arises, in accordance with the Maximum and Minimum server parameters supplied by the system manager in the Tier3 configuration file. Whenever an execution server is created, the INITIALIZE User Action Routine is called to perform application specific image initialization.

In this routine you would normally open file(s) or connect to any database(s) required by your application. It is strongly recommended that you also call the Tier3 system service T3$SETCTX from this routine in order to establish a context variable for subsequent User Action Routines.

As a CLI has been loaded for the execution server process, you are free to spawn a sub-process via the Run-Time library routine LIB$SPAWN, however the process permanent files SYS$INPUT and SYS$OUTPUT should be specified so as to avoid potential conflict with the execution server process.

There is no restriction on an execution server becoming a client of a different Tier3 Application(s) on another node(s).

## 5.2 SESSION START

The SESSION START User Action Routine is called when a client request has been allocated to the execution server for processing. This routine is the means by which Tier3 identifies the associated client to an execution server. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** user_context ,transport_type ,remote_node ,remote_user **,**local_user ,local_persona

---

### Arguments

#### user_context

VMS usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A 32-bit value passed to the T3$SETCTX service. The **user_context** argument is passed without interpretation to all User Action Routines, with the exception of INITIALIZE. If the T3$SETCTX service has not been called, then a zero value is passed.

#### transport_type

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **read only**
mechanism: **by reference**

The **transport_type** argument is the address of an unsigned word that contains the underlying network transport protocol employed by the client in accessing the communication server. Valid transport types are as follows: -

| Transport Type | Symbolic Name | Value |
|---|---|---|
| DECnet | t3$k_decnet | 1 |
| TCP/IP | t3$k_tcp_ip | 2 |

#### remote_node

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor**

The **remote_node** argument contains the address of a descriptor pointing to a 1 to 31 character string that is the network node name of the node on which the client task is executing.

**remote_user**

VMS usage: **char_string**

type: **character-coded text string**

access: **read only**

mechanism: **by descriptor**

The **remote_user** argument contains the address of a descriptor pointing to a system specific structure that identifies the job or task executing on the remote node. The **remote_node** and **remote_user** arguments together constitute client identification.

The format of the **remote_user** structure varies depending on the transport type that was employed by the client when accessing the Tier3 application. For DECnet connections a 1 to 12 byte ASCII string is supplied. For TCP/IP connections Tier3 supplies a structure consisting of a four-byte host address followed by a four-byte port number. The following is a COBOL example of a **remote_user** structure declaration: -

```
01  remote_user_name      pic x(12).
01  remote_user_socket    redefines remote_user_name.
    03  remote_host_addr.
        05  rha_1          pic x.
        05  rha_2          pic x.
        05  rha_3          pic x.
        05  rha_4          pic x.
    03  remote_port_num    pic 9(9)  comp.
```

**local_user**

VMS usage: **char_string**

type: **character-coded text string**

access: **read only**

mechanism: **by descriptor**

The **local_user** argument contains the address of a descriptor pointing to a string that is the VMS username that the client is authorized to execute under, on the local node. You should use the **local_user** argument to determine what access rights and privileges the client is entitled to, when you are performing functions on the client's behalf.

**local_persona (Alpha and IA64)**

VMS usage: **longword_unsigned**

type: **longword (unsigned)**

access: **read only**

mechanism: **by reference**

Within the context of an execution server process, Tier3 maintains a list of personae, one for each client that it has formed an association with. The **local_persona** argument is the VMS persona id of this Tier3 persona. Tier3 personae are maintained at Executive-Mode and are therefore not available to your User-Mode code for use with the standard VMS persona system services.

See the system service T3$PERSONA_ASSUME (Alpha and IA64) for more information on switching a persona.

Please note that the **local_persona** argument is not available on VAX systems.

---

**Description**

Tier3 informs an execution server that an association with a client has been formed by calling this routine. The association persists until you terminate it by specifying either the T3$M_CLOSE or the T3$M_DISCONNECT flag in a call to the Tier3 system service T3$SEND.

Note: The output channel to the client will not be opened until SESSION START has returned control to Tier3. Any attempt to call the T3$SEND system service from this routine will result in the error T3$_CHANCLOSE being returned.

This routine could be used to start a database transaction that would encompass work to be performed in the RECEIVE User Action Routine, or you may merely wish to (re) initialize any session specific working-storage.

## 5.3 RECEIVE

The RECEIVE User Action Routine is called when a message, or fragment of a message, is received from the client with which the execution server is associated. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** user_context ,buffer ,buflen ,flags

---

### Arguments

#### user_context

VMS usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A 32-bit value passed to the T3$SETCTX service. The **user_context** argument is passed without interpretation to all User Action Routines, with the exception of INITIALIZE. If the T3$SETCTX service has not been called, then a zero value is passed.

#### buffer

VMS usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The starting virtual address of the buffer containing the message that was received from the client.

#### buflen

VMS usage: **longword_signed**
type: **longword integer (signed)**
access: **read only**
mechanism: **by reference**

The **buflen** argument is the address of a signed longword that is the length of the buffer, or buffer fragment, in bytes.

The maximum size of any message that can be received by an execution server is limited by the Buffer Size parameter for the application, specified in the Tier3 configuration file. If the client transmits a message that is larger than this maximum value, then the message is broken up into separate fragments by Tier3 and delivered to the RECEIVE routine in successive calls.

**flags**

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **flags** argument is the address of a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option.

The T3$DEF macro defines a symbolic name for each flag bit. The following table describes each flag:

| Flag | Description |
|---|---|
| T3$M_MORE | Tier3 uses this flag to indicate to your RECEIVE routine that it has been presented with only part of a client message and that more data will be provided. This flag will be clear when the final fragment of the message has been delivered. |
| T3$M_OOB | This flag is only ever set when you have chosen TCP/IP as your network transport protocol and you have also set the OOB Inline option to "Y" in the Tier3 configuration file. Tier3 sets this flag to indicate to your RECEIVE routine that it has been presented with Out-of-Band data. |

---

**Description**

You can think of the RECEIVE User Action Routine as a remote procedure, called by the client task with a full-duplex pipe as its only parameter. By exiting the procedure while the pipe, or channel, is still open, you are requesting Tier3 to retrieve more information from the client. When another message is available, Tier3 will once again return control of the execution server to your RECEIVE User Action Routine.

All output from your RECEIVE routine can be channelled back to the client by calling the Tier3 service T3$SEND. When you specify either the T3$M_CLOSE or the T3$M_DISCONNECT flag in a call to T3$SEND, you are informing Tier3 that the logical unit of work that the execution server was performing on behalf of the client has been completed, and that your association with the client should be discontinued. When the RECEIVE routine returns to Tier3 after the channel has been closed, control of the execution server is passed to the SESSION END User Action Routine.

It is possible for the RECEIVE routine to be completely bypassed during a client association cycle. If the network link to the client was lost after an association with an execution server was formed but before the communication server accepted the first message for that association, then the association will be terminated and the processing cycle for the execution server will proceed directly from the SESSION START to the SESSION END User Action Routine.

**Data-Type Incompatibility in a Heterogeneous Network**

Tier3 does not attempt to interpret the contents of any data message received from a client or sent by a User Action Routine, therefore cannot provide any data compression or conversion facilities. If the client task is executing under an operating system other than VMS, you may need to convert data such as integer and floating-point numbers to their appropriate VMS data-types before attempting to operate on them.

For performance reasons, you may decide to require each client task to convert all messages to a VMS readable format before transmitting them to the communication server. If this is not possible or practical for your application, you may decide to use the RECEIVE routine as a clearing-house to interpret all client messages and convert any incompatible data-types to their VMS equivalent formats. The RECEIVE routine could then, based on the type of client message, transfer control to an appropriate subroutine to action the client request. When this subroutine has returned, RECEIVE would convert the results to a client readable format before transmitting them back to the client.

For example, if the client is executing under a PC operating system, it could transmit the ASCII representation of all numeric data rather than its operating system specific integer or floating point representation. Your RECEIVE routine could then use the standard syntax of the language it was written in to convert the data to VMS format, eg:

```
MOVE INTEGER_TEXT IN CLIENT_MSG TO WS_VMS_INTEGER.
```

If the language that you have chosen for your User Action Routines does not support automatic data conversion, or you wish to have more control over the conversion process, you can use the OTS$ Run-Time Library conversion routines to perform the conversion for you. eg:

```
CALL "OTS$CVT_T_F"
     USING  BY DESCRIPTOR FFLOAT_TEXT IN CLIENT_MSG
            BY REFERENCE  WS_VMS_FFLOAT
            BY VALUE      0, 0, 1, 0
     GIVING SYS_STATUS
```

**Fragmented Messages**

If you terminate the association with the client before all the fragments of client message have been received, then the remaining fragments of the message are not lost and will be offered to the first available execution server.

Note: If you request the logical link to the client be broken by specifying the T3$M_DISCONNECT flag in a call to the T3$SEND service, then the remaining fragments of the message are lost.

## 5.4 SESSION END

The SESSION END User Action Routine is called after an association with a client has been terminated. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** user_context

---

### Arguments

**user_context**

VMS usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A 32-bit value passed to the T3$SETCTX service. The **user_context** argument is passed without interpretation to all User Action Routines, with the exception of INITIALIZE. If the T3$SETCTX service has not been called, then a zero value is passed.

---

### Description

After an association with a client has been terminated, the SESSION END User Action Routine receives control of the execution server process. You use the SESSION END routine to perform any session specific clean up procedures that are required by your application.

When the SESSION END routine returns control to Tier3, the communication server is informed that the execution server is available to form an association with another client and process additional requests. The execution server then waits until a new association is formed, in which case Tier3 will once again pass control to the SESSION START routine, or until shutdown notification has been received, in which case Tier3 will pass control to the RUNDOWN User Action Routine.

## 5.5 RUNDOWN

The RUNDOWN User Action Routine is called when the execution server has been culled from the processing pool, due to inactivity timeout. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** user_context

---

### Arguments

**user_context**

VMS usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A 32-bit value passed to the T3$SETCTX service. The **user_context** argument is passed without interpretation to all User Action Routines, with the exception of INITIALIZE. If the T3$SETCTX service has not been called, then a zero value is passed.

---

### Description

You use the RUNDOWN User Action Routine to perform any image rundown procedures required by your application. The RUNDOWN routine is called only once by Tier3 as a result of the execution server not being associated with a client task within the required time interval. The Idle and Minimum server parameters in the Tier3 configuration file govern the culling of execution servers.

Note: This routine is only called when the execution server is exiting due to inactivity timeout. If your clean-up procedures must be invoked regardless of the reason for image exit then you should consider declaring an exit handler in the INITIALIZE User Action Routine.

## 5.6 INTERRUPT

The INTERRUPT User Action Routine is called when either a user interrupt, sent by the client task, or a system interrupt, sent by the communication server, has been received by Tier3. All Tier3 User Action Routines must return by immediate value a condition value in R0.

---

**Format** **user-routine** user_context ,intmsg ,msg_type

---

### Arguments

#### user_context

VMS usage: **user_arg**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

A 32-bit value passed to the T3$SETCTX service. The **user_context** argument is passed without interpretation to all User Action Routines, with the exception of INITIALIZE. If the T3$SETCTX service has not been called, then a zero value is passed.

#### intmsg

VMS usage: **char_string**
type: **character-code text string**
access: **read only**
mechanism: **by reference**

The **intmsg** argument is the address of a 16-byte string that contains the interrupt message that was sent by either the client or the communication server. If the interrupt message is less than 16 bytes long then the **intmsg** string will be left justified and space filled.

#### msg_type

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by reference**

The **msg_type** argument is the address of a longword integer describing the type of interrupt message that has been received.

The T3$DEF macro defines symbolic names for all interrupt types. The following table describes each type:

| Flag | Description |
|---|---|
| T3$K_SYSTEM | Interrupt originated from the communication server. |
| T3$K_USER | Interrupt originated from the client task. |

---

**Description**

The INTERRUPT routine differs from all other User Action Routines in that it is invoked from AST level. For user interrupts, the contents of the **intmsg** argument are user-defined and at the discretion of your client and server developers. For system interrupts, Tier3 uses the first byte of the **intmsg** argument for interrupt identification. Currently “Link Disconnection Notification” is the only interrupt that is sent by Tier3, signifying that the communication server lost the network link to the client with whom the execution server was associated. Therefore, in this version of Tier3, the first byte of all system interrupts will be a byte-integer with a value of 1. Bytes 2 through 16 are reserved for future use.

If the network link to a client has been lost, then Tier3 will automatically terminate any association that client may have had with an execution server. As the execution server's output channel to the client will have been closed for you, your INTERRUPT routine need do to more than return control to Tier3 when Link Disconnection Notification has been received.

Note: If the communication server receives a user interrupt message from a client when that client is not associated with an execution server, the interrupt message is simply ignored.

# 6 TIER3 SYSTEM SERVICE DESCRIPTIONS

## 6.1 T3$PERSONA_ASSUME (Alpha and IA64)

Assume Client Persona

The Assume Client Persona service allows your User Action Routines to direct Tier3 to change the current VMS persona of the execution server process to that of the requesting client.

---

**Format T3$PERSONA_ASSUME**

---

### Arguments

None.

---

### Description

Once Tier3 has created an association between a client and an execution server you can use the T3$PERSONA_ASSUME service to change the persona of the VMS process in which your User Action Routines are being executed, to that of the requesting client. This functionality can be extremely useful for ensuring that the execution server is endowed with only those rights and privileges that the client username is legitimately entitled to hold on the server node.

To switch the execution server persona back to that of the original Username specified in the Tier3 Configuration file, you would use the SYS$PERSONA_ASSUME system service specifying ISS$C_ID_NATURAL as the persona identification.

NB: After calling T3$PERSONA_ASSUME, the ACCOUNT information of the execution server process *will not* be set to that of the client's username but rather the literal "T3$ACC". This restriction is necessary for the internal management of Tier3 personae. Please also note that it is *not* a requirement of Tier3 that your UARs call this system service, and you are free to perform your own persona management should you so desire. Tier3 provides this routine merely as a powerful mechanism for switching persona to that of the client without the need for the heavy duty privileges SYSPRV and DETACH.

### Related Services

SYS$PERSONA_ASSUME

**Condition Values Returned**

| | |
|---|---|
| SS$_NORMAL | The service completed successfully. |
| SS$_PERSONANONGRATA | The calling execution server is not currently associated with a client task. |

Any status value returned from SYS$PERSONA_CREATE, SYS$PERSONA_FIND or SYS$PERSONA_ASSUME.

## 6.2 T3$SEND

Send data or Interrupt Message to Client

The Send Data or Interrupt Message to Client service requests the communication server to transmit either a data message or an interrupt message to the client with which the issuing execution server is currently associated.

---

**Format** **T3$SEND** buffer ,buflen ,flags

---

### Arguments

#### buffer

VMS usage: **address**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The starting virtual address of the buffer that contains the message to be sent.

#### buflen

VMS usage: **longword_unsigned**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The length of the buffer in bytes. If an interrupt message is being sent, then 16 is the maximum value for **buflen** if using DECnet and 1 is the maximum value if using TCP/IP, otherwise it is limited by the Buffer Size parameter for the application, specified in the Tier3 configuration file. Zero-length messages are not permitted.

#### flags

VMS usage: **mask_longword**
type: **longword (unsigned)**
access: **read only**
mechanism: **by value**

The **flags** argument is the address of a longword bit mask that is the logical OR of each bit set, where each bit corresponds to an option. Note: If no flags have been specified, the flags argument must still be supplied as zero.

The T3$DEF macro defines a symbolic name for each flag bit. The following table describes each flag:

| Flag | Description |
|---|---|
| T3$M_INTERRUPT | Send the message to the client outside the normal flow of data messages. Do not specify this flag unless the client is able to receive interrupt messages. Improper use of the T3$M_INTERRUPT flag can result in buffer starvation. |
| T3$M_CLOSE | By specifying this flag, you are informing the communication server that this is the final message that the execution server will be sending to the client, and that the association should be discontinued.<br>If T3$SEND completes successfully when this flag has been specified, the output channel to the client is said to be closed.<br>A subsequent message received from the client will be allocated to the first available execution server and a new association formed. |
| T3$M_DISCONNECT | By specifying this flag, you are requesting the communication server to synchronously disconnect the logical link to the client after the message has been sent.<br>If T3$SEND completes successfully when this flag has been specified, the output channel to the client is said to be closed. |
| T3$M_MULTIPLE | Use this flag to send data messages that are larger than the Buffer Size parameter for the application. When you supply this flag, it indicates that the **buffer** contains only part of a message and that more data will be supplied. To indicate the last fragment of the message being sent, you should call T3$SEND without the T3$M_MULTIPLE flag. This flag is intended for use with network transport protocols that support record boundaries, such as DECnet.<br>This flag is ignored for interrupt messages or if channel closure has been requested. |
| T3$M_NOW | Return to the caller as soon as the message has been placed in the communication server's out tray. If this flag is not specified, then T3$SEND will not return to the caller until the communication server has received the message.<br>Use of this flag can improve performance in some applications where a one-to-many relationship exists between messages received and messages sent, or where multiple messages will be received from a client during a single association.<br>If the out tray is full, then T3$SEND will not return until the communication server has freed enough space to hold the message.<br>This flag is ignored if channel closure has been requested. |

## Description

The T3$SEND service is designed to be called during RECEIVE or INTERRUPT User Action Routine processing, and will only succeed if the issuing execution server is currently associated with a client.

You use the T3$SEND service to transmit data to the client. For each message sent by an execution server, the client must issue a corresponding call to receive the message.

**Terminating an association with a client**

Once the communication server has formed an association between a client and an execution server, Tier3 will continue to request messages from the client on your behalf until you close the channel by specifying either T3$M_CLOSE or T3$M_DISCONNECT in the flags argument to the T3$SEND service. If the channel to the client is open when the RECEIVE User Action Routine returns control to Tier3, then another message, or fragment of a message, is requested. If the channel has been closed, then Tier3 will pass control to the SESSION END User Action Routine.

The communication server may also terminate an association with a client at any time, if the network link to the client has been lost. Tier3 will notify you of such an event by calling your INTERRUPT User Action Routine.

Note: Successful completion of the T3$SEND service *does not* guarantee delivery of the message to the client task, only that the message has been scheduled for processing by the communication server.

Messages will be lost or ignored by the communication server under the following circumstances: -

- The network link to the client was lost during or prior to an attempt to transmit the message to the client. If the association is persisting with an execution server, then the communication server will terminate it, and your INTERRUPT User Action Routine will be called.

  If the link to the client was lost while an attempt was being made to transmit the final message for a given association, then the execution server process *will not* be notified of the disconnection.

- The communication server terminated abnormally or the system manager shut down the network. A forced exit of all execution servers for the application will proceed. The client will be notified by receiving a link abort error on the next network access attempt for the connection.

- The execution server that issued the call to T3$SEND has, itself, terminated abnormally before the communication server was able to process the message. The client will be notified by receiving a link abort error on the next network access attempt for the connection.

It is the responsibility of the developers of the client and User Action Routine components of any application to agree on a message passing protocol that would satisfy both parties as to the success or failure of any atomic unit of work, or association.

**Condition Values Returned**

| | |
|---|---|
| SS$_NORMAL | The service completed successfully. The message has been scheduled for processing by the communication server. If the T3$M_CLOSE or the T3$M_DISCONNECT flag was specified, then the association with the client has been terminated. |
| SS$_ABORT | The T3$SEND request has been cancelled by Tier3 and a forced exit of the execution server is pending. This status will only be returned if T3$SEND was called at AST level. Your routine must return from the AST to allow the forced exit to proceed.<br>This status will be returned if contact with the communication server has been lost while the T3$SEND was in progress. |
| SS$_ACCVIO | Either the argument list or the contents of the **buffer** cannot be read by the caller. |
| SS$_BADPARAM | One of the following conditions was true:<br>➢ The **buflen** argument was specified as zero.<br>➢ An interrupt message was requested, and the **buflen** argument was greater than 16 (DECnet) or greater than 1 (TCP/IP).<br>➢ A data message was requested and the **buflen** argument was greater than the Buffer Size parameter for the application, specified in the Tier3 configuration file. |
| SS$_CANCEL | The T3$SEND request has been cancelled by Tier3 and a forced exit of the execution server is pending. This status will only be returned if T3$SEND was called at AST level. Your routine must return from the AST to allow the forced exit to proceed.<br>This status will be returned if contact with the communication server has been lost while the T3$SEND was in progress. |
| SS$_EXQUOTA | The execution server process has exceeded its BIOLM quota. |
| SS$_INSFARG | Insufficient call arguments were provided. All three arguments must be specified. If no flags are being set, the **flags** argument must be specified as zero. |
| SS$_INSFMEM | The system dynamic memory is insufficient for completing the service. |
| T3$_CHANCLOSE | The execution server is not currently associated with any client; therefore there is no open channel on which data can be sent. |
| T3$_RECVPEN | You attempted to terminate an association with a client after the RECEIVE User Action Routine had returned to Tier3 requesting additional messages.<br>This status will only be returned if T3$SEND was called at AST level and either the T3$M_CLOSE or the T3$M_DISCONNECT flag was specified. |

### 6.3 T3$SETCTX

Set User Context

The Set User Context service establishes a context variable that Tier3 will pass to subsequent User Action Routines as the first parameter in each call.

---

**Format** **T3$SETCTX** user_context

---

**Arguments**

**user_context**

VMS usage: **context**
type: **longword (unsigned)**
access: **read only**
Mechanism: **by value**

Unsigned longword that can be used by the User Action Routines to maintain position during the life of an execution server.

---

**Description**

You use the T3$SETCTX service to identify a user defined context variable that can be used to pass execution context information between subsequent User Action Routines.

All User Action Routines, with the exception of INITIALIZE, receive by value the **user_context** argument as the first parameter in each call. If the T3$SETCTX service has not been called, the value of **user_context** is zero.

The T3$SETCTX service can be called at any time during the life of an execution server.

---

**Condition Values Returned**

| | |
|---|---|
| SS$_NORMAL | The service completed successfully. |
| SS$_ACCVIO | The argument list cannot be read by the caller. |
| SS$_INSFARG | Required argument is missing. |

### 6.4 T3$TIP_GET_TM_URL

Get TIP Transaction Manager URL

The T3$TIP_GET_TM_URL service returns the Transaction Internet Protocol URL for the local hotTIP Transaction Manager.

---

**Format** **T3$TIP_GET_TM_URL** tm_url ,urllen

---

**Arguments**

**tm_url**

VMS usage: **char_string**
type: **character-coded text string**
access: **write only**
Mechanism: **by descriptor**

The **tm_url** argument contains the address of a descriptor pointing to a string that is to receive the URL of the local hotTIP Transaction Manager.

**urllen**

VMS usage: **word_unsigned**
type: **word (unsigned)**
access: **write only**
Mechanism: **by reference**

The **urllen** argument is the address of a word integer into which hotTIP writes the length of the string returned in **tm_url**.

---

**Description**

In order for a coordinating Transaction Manager on a remote node to be able to push a distributed TIP transaction to your local node, it must first be able to locate the hotTIP Transaction Manager running on that node. The T3$TIP_GET_TM_URL service is one way of obtaining the local Transaction Manager's URL. This URL could subsequently be made available to your cooperating client task on the remote node, which in turn would present it to a transaction's coordinating Transaction Manager.

hotTIP translates the following System/Executive-Mode logical names when constructing the value for **tm_url**: -

| Logical Name | Default |
|---|---|
| T3$TIP_TM_NODE | UCX$INET_HOSTADDR |
| T3$TIP_TM_PORT | 3372 |

Please note that during the life of a given executable image, successive calls to the T3$TIP_GET_TM_URL service will not change the results being written to the output arguments. If any of the above logical names have changed and you wish the results to be reflected in the values returned in **tm_url** then you must first stop and then re-run the image that will invoke T3$TIP_GET_TM_URL.

NB: In order to guarantee the ACID properties of your distributed transaction, it is essential that your application code, which will be invoking SYS$START_BRANCH, is executing on the same VMS Cluster node as that of the hotTIP Transaction Manager to which the transaction was initially pushed. In other words, you must ensure that the Application-Pipe and the Transaction-Pipe both terminate at the same SCSNODE in a cluster.

Therefore, when choosing a value for T3$TIP_TM_NODE, do *not* select a cluster alias name or other Round-Robin or Load-Balancing DNS name that will not uniquely resolve to the same SCSNODE as the one that is hosting your application code. This restriction will be hardened in a future release of hotTIP.

---

**Condition Values Returned**

| | |
|---|---|
| SS$_NORMAL | The service completed successfully. |
| SS$_INSFARG | Required argument is missing. |

Any status value returned from LIB$SCOPY_R_DX.

### 6.5 T3$TIP_URL_TO_TID

Convert hotTIP Transaction URL to DECdtm Transaction Identifier

The T3$TIP_URL_TO_TID service converts a hotTIP transaction URL into a Transaction Id and Branch Id format that is actionable by DECdtm.

---

**Format T3$TIP_URL_TO_TID** txn_url ,tid ,bid

---

**Arguments**

**txn_url**

VMS usage: **char_string**
type: **character-coded text string**
access: **read only**
mechanism: **by descriptor**

The **txn_url** argument contains the address of a descriptor pointing to a string that is the URL of the local hotTIP transaction. This string was returned to your cooperating client task on the remote node when it pushed its distributed TIP compliant transaction to the local node.

**tid**

VMS usage: **trans_id**
type: **octaword (unsigned)**
access: **write only**
mechanism: **by reference**

The **tid** argument is the equivalent DECdtm Transaction Id (TID) on the local node, of the hotTIP distributed transaction identified by **txn_url**.

**bid**

VMS usage: **branch_id**
type: **octaword (unsigned)**
access: **write only**
mechanism: **by reference**

The **bid** argument is the equivalent DECdtm Branch Id (BID) on the local node, of the hotTIP distributed transaction identified by **txn_url**.

## Description

Once your cooperating client task on the remote node has successfully pushed its TIP compliant transaction to hotTIP on the local node, it is necessary to export the resultant TIP transaction identifier to the local node via the application-pipe, or middleware, that you are using. The T3$TIP_URL_TO_TID service is available to translate the TIP transaction identifier into a format that is understood and actionable by the DECdtm system service API.

Your server application is now free to enlist any DECdtm compliant resource managers that it may invoke, into the distributed transaction that is being coordinated by the client task. You may also want to consider using the DECdtm XA Veneer if you have XA compliant resource managers that you wish to enlist in the scope of your distributed TIP transaction.

NB: When presenting the **tid** and **bid** arguments in a call to the SYS$START_BRANCH service, you must also specify the **tm_name** argument as that of the local node.

### Related Services

SYS$START_BRANCH, SYS$END_BRANCH, SYS$ABORT_TRANS

## Condition Values Returned

| | |
|---|---|
| SS$_NORMAL | The service completed successfully. |
| SS$_INSFARG | Required argument is missing. |
| SS$_BADPARAM | The **txn_url** argument is not a valid hotTIP transaction Identifier. |

Any status value returned from STR$ELEMENT.

# 7 hotTIP

hotTIP is a Transaction Internet Protocol (TIP) compliant Transaction Management package that enables your VMS server application, and any DECdtm controlled Resource Managers that it invokes, to enlist and participate in a distributed Two-Phase Commit (2PC) transaction with a remote coordinating Transaction Manager. It is the ACID properties of a true 2PC transaction that provide the guarantee of bulletproof data integrity for your distributed heterogeneous applications.

Although hotTIP is bundled with this version of Tier3, it should be stressed that it is by no means a requirement of hotTIP that Tier3 be used as the application middleware product of choice. Would you like to talk to VMS with COM or DCE/RPC? BEA MessageQ, IBM MQSeries, HTML or Tier3 Sockets? The choice is yours and yours alone.

The beauty of TIP's Two-Pipe strategy is it's application-pipe, or middleware, neutrality. Whereas most XA implementations mandate homogenous Transaction Monitor deployments, such as Tuxedo everywhere, Encina everywhere, MQSeries everywhere, ACMSxp everywhere and so on, hotTIP from Tier3 Software gives you complete freedom to choose the middleware product(s) that best suite your particular application and heterogeneous network needs. Another drawback of traditional One-Pipe strategies is that they preclude the run-time determination of transaction participants. Functionality which may be advantageous in a wide-area or Internet based application.

## 7.1 Restrictions

There are certain restrictions with this version of hotTIP that you should be aware of: -

- hotTIP can only enlist and participate in a TIP transaction if that transaction is being coordinated from a remote node.

- Microsoft Transaction Manager / Distributed Transaction Coordinator (MTS/DTC) on Windows is currently the only supported transaction coordinator.

- Transactions cannot be “pulled” by your VMS application from Windows but must rather be “pushed” from your Windows application via MTS/DTC to hotTIP/DECdtm on VMS.

- No cluster-wide recovery. If a hotTIP Transaction Manager on a given VMS cluster node had been managing a transaction that was in a Prepared state when that node crashed, then the transaction would be stalled until that particular node became available again. Unfortunately, in this version of hotTIP, none of the other cluster nodes are able to resolve the transaction on behalf of the failed node. If the server in question is to be unavailable for an extended period then the transaction(s) can be manually resolved by using the LMCP utility.

It is hoped that these restrictions may be lifted in a future release.

## 7.2 Transaction Manager

By default, the hotTIP Transaction Manager is not started as part of the standard Tier3 system startup procedures. If you wish to start hotTIP on your system then you must first uncomment the server startup command contained in sys$startup:t3$startup.com. Once this has been done, then the hotTIP Transaction Manager process will automatically start at system boot-time under the username TIER3$SERVER. The VMS process name for the transaction manager is T3$TIP_MANAGER and it runs at a base priority that is one higher than that registered in the UAF for the TIER3$SERVER username.

If this process ever fails for any reason, such as network shutdown, then it may simply be re-started by entering the command: -

```
$RUN SYS$SYSTEM:T3$STRTTIP
```

The hotTIP Transaction Manager is responsible for providing the interface, and brokering requests, between all coordinating remote Transaction Managers and DECdtm on the local node.

### 7.2.1 Logical Names

The following list contains the System/Executive-Mode logical-names that effect the performance of the hotTIP Transaction Manager: -

| Logical Name | Description |
|---|---|
| T3$TIP_MAX_LINKS | Maximum number of simultaneous connections that the Local Transaction Manager can maintain with remote TMs. Range: 100 to 8191. Default: 512 |
| T3$TIP_OPERATOR | Site-specific operator number to send TIP status messages to. Range: 1 to 12. Default: Network Operator only. |
| T3$TIP_TM_PORT | TCP/IP port number that hotTIP will listen on for incoming Transaction Manager connection requests, and will also be returned to the system service T3$TIP_GET_TM_URL. Default: 3372. |
| T3$TIP_UNRESOLVED_ SEARCH_DETENT | Number of seconds to wait before initiating another scan of unresolved transactions. When contact is lost with a remote TM and that TM was coordinating a transaction on the local node that is currently in a Prepared state, hotTIP will attempt to discover the outcome of the transaction by re-establishing the connection to the remote TM. If the connection attempt fails then T3$TIP_UNRESOLVED_SEARCH_DETENT is the wait interval before retrying. This is a dynamic parameter that can be changed without having to stop and re-start the Transaction Manager. Range:1 to 9999. Default 3. |
| T3$TIP_PREP_TXN_MAX | Maximum number of transactions concurrently in a Prepared state. This logical name is only effective when a new Transaction Journal is being created. Range:100 to 32767. Default 1024. |
| T3$TIP_VERIFY_TM_ID | This logical name has been introduced to help reduce the risks of DoS attacks. If this logical is set to “1” (which is the default) then hotTIP will insist that the TmID in the TIP IDENTIFY command matches the IP address from where the command was sent. |
| T3$TIP_VERIFY_TM_PORT | This logical name has been introduced to help reduce the risks of DoS attacks. If this logical is set to a valid, non-zero numerical value then hotTIP will insist that all IDENTIFY commands received from cooperating Transaction Managers correctly specify this port number for TIP communication. |

### 7.2.2 TIP Manager Log File

All Transaction Manager output, including connection status information, is recorded in the T3$MANAGER:T3$TIP_nodename.LOG log file. Where *nodename* is the SCSNODE name of the VMS node on which the Transaction Manager that created the file was running.

You can interrogate this log file at any time by using the DCL commands TYPE or SEARCH to obtain such information as, why a remote Transaction Manager has been unable to connect to hotTIP. When necessary, the Transaction Manager will also broadcast status information to any terminal enabled as a Network Operator.

### 7.2.3 Transaction Journal

hotTIP maintains an on-disk copy of all currently Prepared transactions in its own Transaction Journal called SYS$JOURNAL:SYSTEM$nodename.T3$TIP_JOURNAL. Where *nodename* is the SCSNODE name of the VMS node on which the Transaction Manager that created the file was running. If the Transaction Journal does not exist when you first start the Transaction Manager on your system then hotTIP will automatically create it and size it according to the logical name T3$TIP_PREP_TXN_MAX.

The hotTIP Transaction Journal is normally a set-and-forget maintenance-free structure. But in the rare event that the Transaction Manager aborts processing with the error T3$_BADPREPDB, you will have to re-size the Transaction Journal using the following procedure: -

1. Re-start the Transaction Manager as described above.
2. Wait until there are no more active transactions by using the following Log Manager command: MC LMCP DUMP/ACTIVE/RM="T3$" SYS$JOURNAL:SYSTEM$nodename.LM$JOURNAL
3. STOP/ID the T3$TIP_MANAGER process.
4. Re-check that there are no active transactions pending. If there are then you must either re-start the Transaction Manager or manually resolve the outstanding transactions using LMCP.
5. Once there are no active transactions, and the Transaction Manager is not running, you can safely delete the hotTIP Transaction Journal.
6. Define the logical name T3$TIP_PREP_TXN_MAX to the value you require.
7. Re-start the Transaction Manager.

### 7.3 Client API

No Tier3 or hotTIP specific software needs to be installed on the client node, nor does hotTIP have any control over, or place any restrictions on, the client API to the Transaction Internet Protocol on that node. The documentation for your client operating system will discuss its own TIP API in detail, but as Windows MTS/DTC is currently the only supported transaction coordinator, a brief description of the actions that your client application will need to perform in order to be able to successfully push a transaction to hotTIP will be included here.

First of all, the CoGetObjectContext function can be used to check whether an object is being run within COM+, and hence whether transactions are handled declaratively or whether the DTC needs to be contacted directly. The following code shows how to perform this check:

```
#include <ComSvcs.h>
. . .
HRESULT       hr;
IObjectContextInfo   *pObjectContextInfo;

hr = CoGetObjectContext (IID_IObjectContextInfo,(void
**)&pObjectContextInfo);

if (SUCCEEDED(hr)) {
  // object being run within COM+, get transaction if any
} else {
  // object not within COM+, start a transaction
}
```

The HRESULT type is declared in winnt.h, and the SUCCEEDED() macro is declared in winerror.h, which also contains a description of most error codes. For the sake of brevity, checks for failure against the codes returned by any of the function calls in the following code are omitted.

To get a handle to the current transaction when run within COM+ the following code can be used:

```
IUnknown      *pTransUnknown;
ITransaction    *pTransaction

hr = pObjectContextInfo->GetTransaction (&pTransUnknown);
hr = pTransUnknown->QueryInterface (IID_ITransaction,(void **)&pTransaction);
```

On the other hand, to start a new distributed transaction explicitly the code is:

```
#include <XoleHlp.h>    // define DTCGetTransactionManager (requires
xolehlp.lib)
. . .
ITransactionDispenser   *pTransactionDispenser;
ITransaction    *pTransaction

hr = DtcGetTransactionManager(
             NULL,                       // pszHost
             NULL,                       // pszTmName
             IID_ITransactionDispenser, // ID of the interface
             0,                          // Reserved: must be null
             0,                          // Reserved: must be null
             0,                          // Reserved: must be null
             (void **)&pTransactionDispenser  // the interface
                              );

hr = g_pTransactionDispenser->BeginTransaction (
             0,                          // Must be null
             ISOLATIONLEVEL_ISOLATED,    // Isolation level
             ISOFLAG_RETAIN_DONTCARE,    // Isolation flags
             0,                          // Transaction options
             (void **)&pTransaction);    // The transaction object
```

Once a handle to the transaction has been obtained, a remote transaction manager can be contacted to enlist any remote services in the transaction as follows:

```
ITipTransaction         *tpTipTransaction;
Char        *remoteTxUrl;

hr = pTransaction->QueryInterface (IID_ITipTransaction,(void
**)&pTipTransaction);

hr = tpTipTransaction->Push("tip://remotehost.co.uk/",&remoteTxUrl);
```

Local work and remote work can now be carried out within the scope of the transaction provided that all the data sources involved have a resource manager that is enlisted with each node's transaction manager. In this case, the application would send the remoteTxUrl over to the remote application for enlisting with hotTIP and DECdtm

Note that when using ADO to perform work against a local database, only COM+ transactions can be joined as ADO automatically checks for the context and enlists the database with the transaction manager (the DTC in this case). Transactions that are started explicitly through the DTC cannot be joined, as the underlying OLE DB interface for doing so is not surfaced by ADO.

Once all the work is complete the transaction can be terminated by either committing or rolling back the changes. The following code can be used to commit a transaction explicitly:

```
hr = pTransaction->Commit (FALSE, XACTTC_ASYNC, 0);
```

For transactions under COM+ control the composite object 'votes' for the transaction outcome as follows:

```
IObjectContext   *pObjectContext;

hr = CoGetObjectContext (IID_IObjectContext,(void **)&pObjectContext);
hr = pObjectContext->SetComplete();
```

The COM+ services will then commit or rollback the transaction according to how all the objects involved in the transaction have voted (a COM+ transaction can be declared to span multiple compound objects).

Once again, it should be stressed that what has been described above are standard Windows APIs for which complete documentation can be found on the Microsoft Developer Network. No Tier3 or hotTIP specific software is required at the coordinating node.

### 7.4 Identifying the Coordinating SQL Server to hotTIP

If contact with a coordinating transaction manager is lost after hotTIP has placed a transaction into a Prepared state then, in accordance with the TIP standard, hotTIP will need to re-establish communications with the coordinator. To do this, hotTIP must be able to resolve the SQL Server name that it was presented with by the coordinating TM when identifying itself in the original connection request. You must ensure therefore, that your local hosts database or your Distributed Name Server is able to translate the name of the SQL Server on the coordinating node. If hotTIP is unable to resolve the SQL Server name to a valid IP address then hotTIP will refuse to accept transaction requests for that SQL Server.

Follow these steps on your client node in order to obtain the SQL Server name for your coordinating transaction manager: -

- Right click on My Computer and select “Manage”
- Expand “Services and Applications”
- Expand “Microsoft SQL Servers”
- The SQL Server name(s) required will now be listed
- Use the UCX command SHOW HOST to test that the coordinator is known

# 8 APPLET UPLOADERS

Tier3 Applet Uploaders provide a light-weight and extremely fast HTTP server capability for your VMS-based servers. This functionality is aimed primarily at those sites that are either not currently running a Web-Server, are using Operating Systems other than VMS as their primary browser-facing architecture, or are using Intranets in which the HTTP-specific protocol and functionality requirements are minimal.

In versions of Java prior to 1.6_10 (and continuing in more recent versions where Policy Files have not deployed), applet-based TCP/IP Sockets are only permitted to connect back to their "codebase", that is, the host from where the JAR or CLASS files were retrieved. (Note that this was also the case with early versions of Adobe's Flex and Flash-Player.) Because of this restriction the browser must be able to retrieve the applet code directly from the VMS server that will be hosting your Tier3 Application(s). You may choose to deploy either Tier3 Applet Uploaders or your existing HTTP Server technology in order to satisfy this requirement.

## 8.1 Logical Names

In this version of Tier3, the following logical names are global in nature and effect the performance of all Tier3 Applet Uploader processes on a given server. (It is envisaged that in later versions of Tier3, each instance of an Applet Uploader will have more finely-grained control and configuration) All logical names must be in Executive Mode in the System logical name table.

Note that if you change the value of one of these logical names then you must stop/restart existing Applet Uploader processes in order for the new values to take affect.

| Logical Name | Description |
|---|---|
| T3$APPLET_ROOT | This must be a concealed logical name pointing to the Web root directory for hosting your HTML, JavaScript and Archive files. For example: -<br><br>`$define/system/exec/translation=concealed -`<br>`_$ t3$applet_root dka0:[www.applets.]` |
| T3$LDR_OPERATOR | Additional site-specific operator number to receive Applet Uploader status messages. Range: 1 to 12. Default: Network Operator only. |
| T3$LDR_LOGGING | When set to Y, T, or 1 this logical name will cause each Applet Uploader to trace all requests in it's log file. The information to be logged includes: - who made the request, what file were they after, and what was the final outcome of the request. |
| T3$LDR_PAGE_SIZE | Data transfer size for disk and network i/o. The value represents the number of 512-byte blocks that constitute a data "page". Larger page sizes can increase performance by reducing the number of i/o required to transfer your file, at the expense of a little extra memory.<br>Range:1 to 63. Default 1. |

## 8.2 Starting an Applet Uploader

To start an Applet Uploader server for a given port number you should enter the following commands:-

$t3$start_ldr:==$t3$strtldr
$t3$start_ldr 80
%TIER3-I-LDRSTRT, Applet Loader started for port 80 PID = 00000F42

The above command creates an Applet Uploader process that will listen on Port 80 for HTTP Get and Head requests. The process name for the Applet Uploader would be T3$APP_LDR_0050. Where 0050 is the hexadecimal representation of the TCP/IP port that the server is listening on.

Note that you'll need (SYSPRV,DETACH) privileges to start and Applet Uploader process.

## 8.3 Stopping an Applet Uploader

There is currently no automated Stop functionality for Applet Uploaders. To stop these processes simply use the VMS STOP/ID command.

## 8.4 Applet Uploader Log Files

Applet Uploader log files are located at T3$MANAGER:T3$APP_LDR_port_scsnode.log. (Where "port" is the hexadecimal port number that the Applet Uploader is listening on and "scsnode" being the SCS Node Name of the cluster node that the Applet Uploader is running on.)

## 8.5 Limitations and Restrictions

The following sections detail the limitations and restrictions that may be encountered when deploying Tier3 Applet Uploaders. In any of these restrictions is considered debilitating then it is suggested that a specialized HTTP server be deployed instead.

### 8.5.1 HTTP version 1.0

Tier3 Applet Uploaders are only capable of supporting HTTP 1.0 syntax and features, and publish this compatibility restriction to all clients. If you rely on HTTP functionality that is only available in version 1.1, or higher, then it is recommended that you move to a specialized web-server.

### 8.5.2 GET and HEAD method support only

Only the GET and HEAD request methods are supported by Tier3 Applet Uploaders. Any other methods will receive a 501 "Not Implemented" response.

### 8.5.3 Block-Mode file processing

Tier3 Applet Uploaders are designed primarily for serving up Java Applet Archive (JAR) files that have a record format of UDF (undefined) and process all files in block-mode only. Therefore they will not honour any implicit VFC file processing, RMS control characters, or record delimiters such as <CR> and/or <LF>. If you would like to serve any such files via an Applet Uploader then they must first be converted to STREAM format. The following is an example of using the FDL utility to convert a HTML page from "Variable Length" format to "Stream" format: -

```
$convert/fdl=sys$input my_page.html my_page.html
File
        organization            sequential
Record
        block_span              yes
        carriage_control        none
        format                  stream
$!
```

### 8.5.4 No SSL Authentication and Encryption

Tier3 does not currently support Application Layer Security such as SSL communication. If it is a requirement of your application(s) that all communication takes place between only authenticated servers and/or clients, or only over encrypted network connections, then it is recommended that you either: -

1. Configure a network layer protocol such as IPsec, whether targeted globally at all communications between the computers involved in the application, or on a specific port number, or range of port numbers, basis
2. Implement a VPN via other means
3. Front-end, or at least provide an alternative access-point for, your servers via a tool such as STUNNEL which will then relay the decrypted traffic to your Tier3 Applet Uploader(s) and Application Server(s).

# 9 EXAMPLE PROGRAMS

There are several comprehensive programming examples located in the T3$EXAMPLES directory that demonstrate both Tier3 and hotTIP functionality. It is strongly recommended that you, at least, familiarise yourself with the coding practises contained in these examples before embarking on your own server development.

## 9.1 Tier3 only

This example creates a Tier3 Application Server that provides a remote printer and batch queue lookup facility. Also included are example VMS client programs to facilitate simultaneous DECnet and TCP/IP client connections.

The following source files and command procedure create the DEMO application that was registered in the Tier3 Configuration file in section 2.1.1.

### 9.1.1 DEMO_UARS.COB

This file contains the COBOL source for the six user action routines that Tier3 will activate, on your behalf, when satisfying client requests for the DEMO application. Some of the features demonstrated in this example are: -

- Argument format and handling for User Action Routines
- Tier3 system services T3$SETCTX, T3$SEND and T3$PERSONA_ASSUME (Alpha and IA64)
- How to use compilation switches to launch a debug session
- How your RECEIVE routine is notified of fragmented messages
- 1:M relationship between messages received and messages sent
- What to do if the INTERRUPT routine is called

### 9.1.2 BUILD_UARS.COM

DCL build procedure for compiling and linking DEMO_UARS.COB into a shareable image that can be registered for the application in the Tier3 Configuration File. You can run this command file in the T3$EXAMPLES directory before registering the DEMO application.

### 9.1.3 DEMO_CLIENT_DECNET.COB

COBOL source file for a potential client of the DEMO application. This client program employs DECnet as the network transport of choice for communicating with your Application Server. Please note that the DEMO application must have DECnet enabled in the Tier3 Configuration File in order to be able to accept connect requests from this program.

Examples of all the VMS system service calls necessary for connecting to, and exchanging messages with, the DEMO application via DECnet are contained in this program. Of particular interest will be the APPLICATION_LOGON section that demonstrates the handshake procedure that must be followed by all clients when connecting to Tier3 server applications.

The build instructions for this client are included at the top of this source file. NB: Before compiling this program, the nodename in the NCB and NCB_ID fields needs to be changed from TIER3:: to the DECnet nodename of the node which is to host the DEMO application.

### 9.1.4 DEMO_CLIENT_TCP_IP.COB

COBOL source file for a potential client of the DEMO application. This client program employs TCP/IP as the network transport of choice for communicating with your Application Server. Please note that the DEMO application must have TCP/IP enabled in the Tier3 Configuration File in order to be able to accept connect requests from this program.

Examples of all the VMS system service calls necessary for connecting to, and exchanging messages with, the DEMO application via TCP/IP are contained in this example. Of particular interest will be the APPLICATION_LOGON section that demonstrates the handshake procedure that must be followed by all clients when connecting to Tier3 server applications.

The build instructions for this client are included at the top of this source file. NB: Before compiling this program, the host address in the REM_NODE_ADDR field needs to be changed from 1.2.3.4 to the TCP/IP host address of the node that is to host the DEMO application.

#### 9.1.5 DEMO_TCP_IP_DEF.MAR

External symbol definitions that are referenced by DEMO_CLIENT_TCP_IP.COB and that are needed for link-time resolution.

### 9.2 Tier3 and hotTIP

In this example Rdb is enlisted as a resource manager and, when updating the DEPARTMENTS table in the MF_PERSONNEL database, is also a participant in a hotTIP brokered, distributed transaction with a remote coordinating transaction manager. Commit them all or roll them all back.

#### 9.2.1 DEMO_TIP.COB

This file contains the COBOL source for the six user action routines that together constitute the Tier3 Application Server. Some of the features demonstrated in this example are: -

- Using the T3$DEBUG DCL symbol to decide whether or not to launch a debug session.
- Using Rdb as a data source and a resource manager under the control of hotTIP
- Demarcation and control of hotTIP transactions and their interaction with DECdtm
- Tier3 system services T3$TIP_GET_TM_URL and T3$TIP_URL_TO_TID

#### 9.2.2 DEMO_TIP_SQL.SQLMOD

SQL Module Language source file containing the SQL necessary to attach to, and update, the personnel database.

#### 9.2.3 BUILD_TIP_DEMO.COM

Shareable image build procedure to compile and link the COBOL, SQL and external MACRO symbol definitions into the six User Actions Routines required by Tier3. Some of the features demonstrated in this example are: -

- When to use the SQLMOD qualifier /CONTEXT
- Deferring database constraint checking so as to test Rdb's ability to veto a transaction
- Use of the T3$DEF macro for compile-time bit flag manipulation

### 9.3 hotTIP only

This is the simplest coding example of a VMS server that incorporates hotTIP functionality.

#### 9.3.1 DEMO_TIP_AUXS.COM

This file contains all of the DCL/COBOL/SQLMOD/UCX required to create an Auxiliary Server/INETd process listening on port 303, that will insert a row into the MF_PERSONNEL.EMPLOYEES table in cooperation with a Windows client that is inserting a row into the NORTHWIND.EMPLOYEES table, commit them all or roll them all back. This working example is less than 500 lines long.

## 9.4 Database Server for Flex Charting

This example creates a Tier3 Application Server that retrieves Employee and Salary details from the Rdb database MF_PERSONNEL. The results will be presented via Flex Pie-Charts and Datagrids in the DEMO_CLIENT_FLEX.HTML client example.

The following source files and command procedure create the FLEX example application. As was the case with the previously discussed DEMO application, the FLEX example also uses a buffer size of 510 bytes. Although the other parameters for FLEX are similar to those for DEMO (as seen registered in the Tier3 Configuration file in section 2.1.1.) please note that the Image Name is now DEMO_FLEX and the User Action Routine names are now prefixed with FLEX_ instead of DEMO_. Furthermore, a different TCP/IP port number (1500) has been chosen, and DECnet support has not been requested.

### 9.4.1 DEMO_FLEX.COB

This file contains the COBOL source for the six user action routines that Tier3 will activate, on your behalf, when satisfying client requests for the FLEX application. Using the selection criteria supplied from the DEMO_CLIENT_FLEX.HTML web page, the Employee result-set is returned back to the client for processing on a row by row basis, as soon as the data becomes available. This strategy permits a fair degree of parallelism between the data-retrieval and presentation layers.

### 9.4.2 DEMO_FLEX_SQL.SQLMOD

SQL Module Language source file containing the SQL necessary to attach to the MF_PERSONNEL database and retrieve the current salary details for each selected employee.

### 9.4.3 BUILD_FLEX_DEMO.COM

DCL command procedure for compiling and linking DEMO_FLEX.COB and DEMO_FLEX_SQL.SQLMOD into a shareable image that can be registered for the application in the Tier3 Configuration File. You can run this command file in the T3$EXAMPLES directory before registering the FLEX application.

## 9.5 Web Browser GUI Client Examples

There are now also two browser-based client examples that can be found in the T3$EXAMPLES directory, and which demonstrate just some of the ways that Tier3 can be used to service your Rich Internet Applications. Each example is invoked via its respective target web-page: -

1. DEMO_CLIENT_WEB.HTML - a GUI version of the VMS Queue lookup facility.
2. DEMO_CLIENT_FLEX.HTML - an example of how Flex Charting, FABridge functionality, and JavaScript can be combined with Tier3 to deliver up your VMS security, performance, business-rules, and data, for enhancement and presentation via high-quality graphical front-ends.

These examples will be discussed in detail below, but it is important to stress once again at this point that absolutely no Tier3-specific software needs be installed on the client in order to communicate with a Tier3 Application Server. What is being demonstrated is just one way of solving the problem.

While the Java Applet solution provided is generic in nature and multi-purpose enough to satisfy many application requirements, you are certainly not restricted to using only the Java code that has been provided.

Each of these target web-pages contains a HTML Frameset that breaks up the visible page into a header frame containing CORNUCOPIAE.HTML and the body frame(s) that will contain the application-specific web-pages and forms. The HTML "src" for the second frame in each example is left blank as this will be populated automatically by the Java Applet's init() method, in CornuCopiae.java, with the value specified in the FORM_NAME variable. This is done once user authorization has been successful.

### 9.5.1 Application Commonality and Code Re-use

One of the many facets of these examples that you may find useful is the extent to which existing code can be re-used for subsequent browser-based client applications. In particular, the source files for the Java Applet Archive file TIER3.JAR are capable of providing the network connectivity for almost any browser-based Tier3 client. In fact, this same archive file and it's corresponding HTML Applet Declaration is used, without modification, for both of the included examples. Only the parameters to the applet need change.

#### 9.5.1.1 CORNUCOPIAE.HTML

This file constitutes the screen header frame for each application. Once a connection has been established to your Tier3 Application Server and the user has successfully logged-on, the form is populated with the following fields: -

- *Username*: The VMS username that the user has logged-on as.
- *Node*: The SCSNODE name of the cluster node the user is connected to.
- *Host*: The Host Name the user is connected to, as specified in the applet's codebase URL.
- *Connected*: Elapsed time for the connection, updated each second.

There is no-scrolling and no-resizing set on the frame that will hold this page and it is designed as a standard header with contextual information that will assist the user as they navigate through numerous browser instances, or several tabs within a single browser instance.

Also to be found on this page is the HTML Applet Declaration. Note: You must change the URLs in this Applet Object declaration to point to your "codebase" directory containing the JAR file, before attempting to run these examples.

#### 9.5.1.2 COMMON.JS

This file contains global-variable and function declarations that are generally useful, and common to both Tier3 Web Browser Client examples. You can include these features into your code as you would any other copybook, or include-file, by using the following HTML syntax: -

```
<script type="text/javascript" src="common.js" ></script>
```

#### 9.5.1.3 ACCESSDENIED.HTML

This file contains the most basic of web pages. It is loaded automatically by the Java applet's init() method, in CornuCopiae.java, once user authorization has been unsuccessful. The loading of a new top level page results in the applet's destroy() method being called and any network link torn down.

### 9.5.2 Java Applet Archive file TIER3.JAR

Java has been chosen as the mechanism for exposing the underlying TCP/IP Socket interface to the example JavaScript client code. Please be aware that both Adobe Flex and Microsoft Silverlight also provide a Socket interface, and there is further discussion in HTML5 about WebSockets and how they will be implemented at the HTML level. But when it came to the Tier3 Web Browser Client examples, Java was chosen as it currently provides the most robust and feature rich API for accessing the underlying TCP/IP Socket layer.

Once the HTML Applet Declaration has been included into your web-page, and that page has been loaded by your browser, your JavaScript code is free to access any of the applet's public methods. See CornuCopiae.java for more.

#### 9.5.2.1 Applet Input Parameters

The names of the applet's input parameters, that are declared and supplied via the HTML Applet Declaration, and the effect they have on applet behaviour are as follows: -

- *FORM_NAME*. Name of the HTML page that will be loaded into the second frame of the frameset, and that contains the application-specific logic and functionality. If this parameter has not been provided then obviously no attempt to load the page will be made.
- *PORT*. Identifies the TCP/IP Port number that your Tier3 Application Server is listening on at the destination server. (The host name for the outgoing Socket connection is automatically derived from the "codebase" as specified in HTML Applet Declaration.)
- *MAXBUF*. Maximum size of a data buffer that can be read or written as a single chunk. This value corresponds to the value specified when the application was registered in the Tier3 Configuration file in 2.1.1.3.1. Its use on the client is mainly in the dimensioning of internal buffer areas and to provide a default value for the Tier3Socket.readMessage() method when no explicit length value has been specified.
- *APPLICATION*. The Application identifier displayed in the "Welcome" dialogue box.
- *HOSTCHARSET*. Character encoding to be referenced when retrieving strings from the byte-stream returned from the VMS server.
- *SSL_REQD*. If specified as "Y" the default Secure Socket Layer protocol for the client (which can be over-ridden in the Java security properties file) will be used for encryption and for server authentication. NB: Tier3 Application Servers do not currently support SSL. If you do set this parameter to "Y" then you are expected to front-end your Tier3 application with a product such as STUNNEL. A recommended alternative to the SSL protocol is to configure IPsec between client and server.

#### 9.5.2.2 Class Source Files

The following four source files represent the entirety of the Java code required to connect to and communicate with your Tier3 Application Servers.

What has been offered is a fully-functional, yet bare-bones, framework for "What needs to be done". Obviously, you may choose to use something a little more state-of-the-art than AWT for your dialog boxes, or you may choose to obtain user authentication information directly from HTML rather than from Java; the choices, as always, are yours.

##### 9.5.2.2.1 CornuCopiae.java

Contains the mainline event processing logic for the applet, and exposes those public methods that are directly accessible to your JavaScript code.

The CornuCopiae.init() method is automatically invoked as part of the page loading process for CORNUCOPIAE.HTML. It is at this stage that the applet attempts to connect to your Tier3 Application Server and query the user for their credentials. If user authorization has been successful then the HTML page specified in the FORM_NAME applet parameter will be loaded into the second frame, otherwise the ACCESSDENIED.HTML page will be displayed in the top-level frame.

The CornuCopiae.destroy() method is automatically invoked whenever the browser instance or browser tab is closed, or when the page in the top-level frame is replaced.

The page in the top-level frame will be replaced if: -

- There was a failure connecting to the remote Tier3 Application Server.
- User-authentication failed. (The reason for the failure can be found in the Communication Server Log Files for your application.)
- A logic error occurred during processing and the JavaScript function “reportError(txt)” was called. (This function can be found in COMMON.JS)
- The user is simply navigating to a new page.

Note that is possible to code your Java Applet in such a way as to have the Socket, and underlying network connection, survive applet destruction/rundown and therefore survive browser navigation across various unrelated web-pages and back again. For Tier3 example purposes, the CornuCopiae class was designed to bind the instance of the network connection closely to the instance of the web-page that created it.

#### 9.5.2.2.2 Tier3Socket.java

The Tier3Socket class is responsible for establishing and managing the local TCP/IP Socket, and for publishing the methods necessary for connecting to, and exchanging messages with, your Tier3 Application Servers.

See the jobLookup() function in QUEUE_LOOKUP.HTML for an example of how the setTimeout(int) method can be deployed to provide an erstwhile "blocking" socket Read with the ability to surrender the browser’s Event Dispatching Thread, for things like processing the Abort button, and for ticking over the clock.

#### 9.5.2.2.3 Tier3Logon.java

The Tier3Logon class is responsible for interrogating the user for the necessary credentials in order to access your Tier3 Application Server. It does this by popping up a modal dialogue box and asking for a Username and Password

#### 9.5.2.2.4 Tier3Welcome.java

If Tier3 has authorized this user to access your Application Server, and if the user had previously left the “*Display logon confirmation*” checkbox ticked when they entered their Username and Password, then a second modal dialog box will be displayed. The information displayed will be similar to that as if the user had logged onto VMS via a dumb-terminal i.e. Last login times, Number of failures since last successful login, and so on.

Of further interest may also be the inner class DateVMS, and the general VMS-date to Java-date conversion handling.

### 9.5.2.3 HTML Applet Declaration

The following example is the JavaScript declaration for the Java Applet Archive file TIER3.JAR that will handle all of the TCP/IP Socket manipulation necessary to communicate with your Tier3 servers.

```
<script type="text/javascript">

var appletDef = navigator.appName;

if (appletDef == "Microsoft Internet Explorer")
   document.write
      (
      '<object classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93" ',
               'width= "0" height= "0" name="CornuCopiae" id="CornuCopiae">',
                        '<param name="archive"     value="tier3.jar">',
                        '<param name="codebase"    value="http://1.2.3.6/example2/applets/">',
                        '<param name="code"        value="CornuCopiae">',
                        '<param name="mayscript"   value="yes">',
                        '<param name="scriptable"  value="true">',
                        '<param name="name"        value="CornuCopiae">',
                        '<param name="PORT"        value="', parent.portName, '">',
                        '<param name="HOSTCHARSET" value="ISO-8859-1">',
                        '<param name="MAXBUF"      value="', parent.buffSize, '">',
                        '<param name="APPLICATION" value="', parent.appnName, '">',
                        '<param name="FORM_NAME"   value="', parent.formName, '">',
                        '<param name="SSL_REQD"    value="', parent.sslReqd, '">',
      '</object>'
      );
else
   document.write
      (
      '<object classid="java:CornuCopiae.class" ',
          'type="application/x-java-applet" archive="http://1.2.3.6/example2/applets/tier3.jar"',
               'width= "0" height= "0" name="CornuCopiae" id="CornuCopiae">',
                        '<param name="archive"     value="tier3.jar">',
                        '<param name="codebase"    value="http://1.2.3.6/example2/applets/">',
                        '<param name="code"        value="CornuCopiae">',
                        '<param name="mayscript"   value="yes">',
                        '<param name="scriptable"  value="true">',
                        '<param name="name"        value="CornuCopiae">',
                        '<param name="PORT"        value="', parent.portName, '">',
                        '<param name="HOSTCHARSET" value="ISO-8859-1">',
                        '<param name="MAXBUF"      value="', parent.buffSize, '">',
                        '<param name="APPLICATION" value="', parent.appnName, '">',
                        '<param name="FORM_NAME"   value="', parent.formName, '">',
                        '<param name="SSL_REQD"    value="', parent.sslReqd, '">',
      '</object>'
      );

</script>
```

Note that while there are separate <OBJECT> definitions, depending on whether or not the browser is Microsoft Internet Explorer, all of the parameter information is the same. Please also note that the CornuCopiae applet is faceless and takes up absolutely no real-estate on the web-page itself. All user interaction is carried out directly from within the applet in the form of modal dialogue boxes.

The application-specific applet parameters (*port*, *maxbuf*, *application*, *form_name*, and *ssl_reqd*) are retrieved from JavaScript variables that were initialized in the respective “parent” framesets. This strategy allows the applet declaration to become generic in nature and located commonly in the CORNUCOPIAE.HTML file. If you choose not to use framesets or would like to include your applet definition in perhaps just a single page, then you can simply hard-code the values for these parameters.

If you intend to use a Tier3 Applet Uploader to serve up your TIER3.JAR file in the above example, then it would need to be placed in the T3$APPLET_ROOT:[EXAMPLE2.APPLETS] directory on your VMS server.

#### 9.5.2.4 Building the Applet Archive file

Before creating your JAR file you must first create the Java CLASS files from the source code found in the T3$EXAMPLES directory. Note that if you do not currently have your *classpath* environmental variable defined to include your current working directory then you'll have to explicitly include it as an option when compiling CornuCopiae.java. All example Java source code included achieved clean compiles and successful execution down to version 1.4.2_13

An example of the Java source code build is as follows: -

```
javac Tier3Socket.java
javac Tier3Logon.java
javac Tier3Welcome.java
javac -classpath c:\this\default CornuCopiae.java
```

Now we can combine all of the new CLASS files into a single JAR file: -

```
jar -cf tier3.jar CornuCopiae.class Tier3Socket.class Tier3Logon.class Tier3Welcome.class
Tier3Welcome$DateVMS.class
```

Your TIER3.JAR archive file can now be moved to the appropriate web-accessible directory on the VMS server that will host your Tier3 Application Server(s).

NB: Java is extremely case-sensitive! You must adhere strictly to the camel-case conventions in the class names above, especially when building your JAR file. If not you may receive *classnotfound* errors when attempting to invoke your applet.

### 9.5.3 Web-Browser Restrictions and Limitations

In order to be able to execute these example applications correctly you must first ensure your browser and network setting are configured so as to conform to the following restrictions and limitations: -

- JavaScript must be enabled.
- Java Applets must be enabled.
- You must be running a JRE plug-in that should not an older version than that used to compile your code. (The example code was tested with SUN's JRE 1.4.2_13 and 1.6)
- To view the DEMO_CLIENT_FLEX.HTML example you must be running the latest release of Flash Player 9 or above.

### 9.5.4 DEMO_CLIENT_WEB.HTML

Is a much more feature rich GUI version of DEMO_CLIENT_TCP_IP.COB, yet it still talks to exactly the same server code that can be found in DEMO_UARS.COB.

#### 9.5.4.1 QUEUE_LOOKUP.HTML

Some of the functionality highlights to be found on display here include: -

1) A Hot-Abort button! After you have pressed the Get Job Info button you'll notice that the Abort Request button becomes active and turns red. Actually you probably won't notice because this query completes too quickly. You can edit the DEMO_UARS.COB code and change the value of the DEBUG_DELAY field if you want to see your 3GL Interrupt routine in action. In this case the CANCEL_REQUEST flag set in the AST routine is picked up in the mainline code, resulting in the graceful termination of the loop that controls Next Queue/Row retrieval.

2) Predictive text on the Queue Name field so that all matching VMS queues are retrieved on-demand as the user types each character. As is now common-place with many websites, a drop down select list of matching options is automatically retrieved from the server and made available for the user to select from.

3) Result-set drill-down. Many database queries return a result-set of rows for the user to scan through and possibly drill-down into for more detail. What has been provided here is a reasonably generic example of this, where all matching Job Entries have been populated into a dynamic HTML Select List. Note that the user was able to see the select-list grow, the scroll-bar diminish, and the Jobs Found field tick over in real-time, whilst continually being empowered (by the Abort button) to curtail the results at any time!

If you click on one of the line entries in the Select List then the <frame> changes and the ENTRY_DETAILS.HTML web-page appears. See the parent.entry_details.getReady() call in QUEUE_LOOKUP.HTML to see how the handover to the new frame takes place. (Also see goBack() in ENTRY_DETAILS.HTML to see how simply that operation is reversed.)

4) Local Result-Set Sort. If you click on the "header" or "first" row in the Select List of queues, you will get a popup prompting you for a sort key. If you select one, the contents of the Select List are sorted in the chosen order. (Try entering "*" for the Queue Name and then click "Get Job Info" to retrieve some data worth sorting)

#### 9.5.4.2 ENTRY_DETAILS.HTML

Displays detailed information for the individual queue entry that was chosen from the select-list.

Once this web-page has been displayed, the user is free to move forward, back, first, last, refresh, and delete queue entries, or return to the previous frame. (Thanks to the deployment of the VMS Persona functionality, the user is only permitted to see those queue entries that the username they logged in under is permitted to see. They can also *only* delete those entries that this username is allowed to delete.)

You'll notice that any queue names are highlighted in bold and italics; if you mouseover any of these fields when they are not blank then the current status information for that queue will be retrieved from the server and displayed in a quasi-popup, floating DIV.

### 9.5.5 DEMO_CLIENT_FLEX.HTML

Combines the RIA capabilities of Flex Charting, FABridge functionality, and JavaScript with the power of Tier3 to deliver up your VMS security, performance, business rules, and data for enhancement and presentation via high-quality graphical front-ends.

#### 9.5.5.1 EMPLOYEE_LOOKUP.HTML

The user is prompted for a partial surname to match against the Employees table from the Rdb example database MF_PERSONNEL. (Please be aware that the first character should be upper-case and all subsequent characters in lower-case. Or just press *GO* to retrieve all Employees.) See the Database Server for Flex Charting section for a discussion of the code that will service requests from this web-page.

Of particular interest in this example is the way Adobe’s FABridge functionality has been deployed in order to expose the Flex ArrayCollections (employeeFeed, genderFeed, deptTable) as defined in BRIDGETEST.MXML, and make them as accessible as any other JavaScript objects. See the formLoad() function for details.

As was the case with QUEUE_LOOKUP.HTML, a Hot Abort button has been implemented enabling the user to terminate a long running query at any time. It should be noted that, for example purposes, the decision was made to have the *departments* pie chart visible during the entire time that it is being populated. Although it may be aesthetically pleasing to achieve the almost animated affect of the chart rotating and growing before the user's eyes, it is clearly not an optimal solution from a rendering performance view point. In a real-world application it may be desirable to hide such a chart until it was completely populated with only the final instance of the chart rendered on the web-page.

#### 9.5.5.2 BRIDGETEST.MXML

This file contains the source code necessary for creating the Shockwave Flash file BridgeTest.swf that will be loaded into the *flashoutput*  in EMPLOYEE_LOOKUP.HTML via AVOIDPATENT.JS. The actual build and export procedures are the responsibility of Adobe's Flex Builder IDE, and outside the scope of this manual.

Obviously all of the intrinsic Flex features such as Panel-resizing, and Grid-sorting are made available to the user. But one feature worth noting in this particular example is the ability to drill-down into the data simply by exploding a slice of the pie chart. That is, if you click on a slice of either pie chart with the left button of the mouse, then that wedge will be exploded and those employees belonging to the selected Department or Gender will be automatically reflected in the DataGrid below.

Another useful feature demonstrated in this example is DataTipping. When you hover over a slice in either of the pie charts, a DataTip box will appear informing the user of the numerical value of the data, in this case Salary, that is being represented by that slice.

#### 9.5.5.3 FABRIDGE.JS

This file is supplied by Adobe and contains all of the magic needed to facilitate the bridge between JavaScript and Flex.

#### 9.5.5.4 AVOIDPATENT.JS

Contains the JavaScript controlling the runtime inclusion of the Flash Object for the BridgeTest.swf file, thus bypassing the annoying "Click to activate this control" message that would have been presented to the user had a static declaration been used.

### 9.6 Server Push Technology

This final example has little to do with Tier3 Application Server development directly, and would continue to be technically relevant and function correctly even if Tier3 was not installed on your system. Tier3 is founded on the Client-Pull style of network communication and relies on the client initiating the message exchange in which your Application Server will participate. Although server-affinity for any application is completely under the control of your User Action Routines, it would be highly unusual to effectively dedicate an Execution Server process to each client by continuing that affinity for the duration of an extended long-poll.

What follows in this section is a discussion of an implementation of true Server Push technology and how that may be combined with your Tier3 Application Servers to meet your business requirements.

#### 9.6.1 TIER3PAGER.HTML

Unlike the other Tier3 Web Browser GUI Client Examples, this web-page does not set up a frameset nor does it deploy CORNUCOPIAE.HTML as a screen-header. Instead the applet's HTML Object definition is included directly into this web-page, and Tier3Pager.java is the applet class rather than CornuCopiae.java.

As you can see from the description on this web-page under the heading *"But why is nothing happening?"*, you won't witness much activity here until you activate the server-push part of the example - DEMO_UDP_MSG.COB. But a couple of points worth noting on this web-page are firstly, the "mayscript" applet parameter has been set to "yes" as is required by some browsers in order to permit the applet to invoke JavaScript functions, and secondly, the makeBid() JavaScript function has been left as a stub. It is envisaged that within the makeBid() function is where the hooks into your Client Pull Tier3 Application Servers would be included. In other words, asynchronous events like stock-price movements or chat-messages are covered by Server Push technology, but when it's time to bid for a Stock, or perform some other transactional activity, e reliability, performance, and integrity of a Tier3 Application Server.

### 9.6.2 Additional Java Classes

Two additional Java classes were created to provide the functionality needed for the Server Push example. The first responsible for managing the applet and the second responsible for looking after the Frame that will display the "Chat-Session" messages.

#### 9.6.2.1 Tier3Pager.java

This class fires up a separate Thread, via the inner class MessageThread, that will listen constantly for incoming UDP messages on port 1234. Because the processing of all messages pushed from the server takes place within the context of a separate thread, the regular user interaction with the browser and its Event Dispatching Thread may proceed unencumbered.

Stock Price messages are sent straight to JavaScript via the priceUpdate(netPrice) function contained in TIER3PAGER.HTML and Chat Messages are appended to the Frame created via Tier3Talk.java.

#### 9.6.2.2 Tier3Talk.java

Creates a simple Frame and TextArea for appending text messages that have been received after being pushed asynchronously from the server.

#### 9.6.2.3 Updating JAR File With New Classes

The additional Java source files need to be compiled as follows: -

```
javac Tier3Pager.java
javac Tier3Talk.java
```

Now we can combine the new CLASS files into the JAR file from 9.5.2.4: -

```
jar -uf tier3.jar Tier3Pager.class Tier3Pager$MessageThread.class Tier3Talk.class
```

Note that these class files could have been located in a separate Java Archive File of their own and that there is no requirement for them to be included in the same JAR file that was created in the previously discussed web client examples. They have been included here purely for convenience and to limit the number of JAR files floating around the system.

### 9.6.3 DEMO_UDP_MSG.COB

This COBOL program will simulate the *pusher* in a real-world Server Push application. When executing this program the user is initially prompted for the host name, or IP address, of the client system that is currently browsing the TIER3PAGER.HTML web-page. From that moment on, the following two types of messages will be sent to the remote client's system and picked up in a separate thread run by Tier3Pager.java: -

1) Stock-Price Updates. A new random Stock Price (between 0 and 100) is generated every 2 seconds and the result transmitted to the remote client asynchronously. The client applet responds to these messages by invoking the JavaScript function priceUpdate(netPrice) so that the results can be reflected in the TIER3PAGER.HTML web-page.
2) Chat-Session Messages. While the Stock Ticker is busy doing its thing, the user is also being prompted for some message text to transmit to the remote client. The client applet responds to any such messages by appending them to a separate Frame in order to simulate a chat session.

In a real-world application it is envisaged that each client would subscribe to the Chat-Session or Stock-Ticker service and that it would be the responsibility of each service to keep track of current subscribers, their IP addresses, and the port number(s) they're accepting messages on.

You compile and link this program as follows: -

```
$COBOL/LIS DEMO_UDP_MSG
$MACRO/LIS DEMO_TCP_IP_DEF
$LINK DEMO_UDP_MSG,DEMO_TCP_IP_DEF
```

If you have already compiled or assembled the DEMO_TCP_IP_DEF.MAR file, as part of the DEMO_CLIENT_TCP_IP.COB build procedure, then obviously you do not need to repeat that step here.