

Oracle® Tuxedo Mainframe Adapter for SNA

Reference Guide

12c Release 2 (12.1.3)

August 2014

ORACLE®

Copyright © 2007, 2014, Oracle and/or its affiliates. All rights reserved.

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

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

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

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

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

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

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

Contents

1. ATMI to CPI-C Function Mapping

2. Application-to-Application Programming Examples

Distributed Program Link (DPL) Examples	2-1
ATMI Client Request/Response to CICS/ESA DPL	2-2
ATMI Client Asynchronous Request/Response to CICS/ESA DPL	2-3
ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DPL	2-4
CICS/ESA DPL to ATMI Request/Response Server	2-5
CICS/ESA DPL to ATMI Request/Response Server, Service in Autonomous Transaction.	2-6
ATMI Client Request/Response to CICS/ESA DPL, Autonomous Transaction	2-8
Transactional ATMI Client Multiple Requests/Responses to CICS/ESA DPL	2-10
Transactional CICS/ESA DPL to ATMI Request/Response Server.	2-12
CICS/ESA DPL to ATMI Requests/Responses Server for Channel /Container.	2-14
Distributed Transaction Processing (DTP) Examples	2-15
ATMI Client Request/Response to CICS/ESA DTP	2-16
ATMI Client Asynchronous Request/Response to CICS/ESA DTP	2-18
ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DTP	2-20
ATMI Conversational Client to CICS/ESA DTP, Server Gets Control	2-21
ATMI Conversational Client to CICS/ESA DTP, Client Sends/Receives Data	2-23
ATMI Conversational Client to CICS/ESA DTP, Client Grants Control.	2-25
CICS/ESA DTP to ATMI Conversational Server, Client Retains Control.	2-27
CICS/ESA DTP to ATMI Conversational Server, Client Relinquishes Control	2-29

Transactional ATMI Client Request/Response to CICS/ESA DTP	2-31
Transactional ATMI Conversational Client to CICS/ESA DTP, Server Gets Control	2-33
Transactional CICS/ESA DTP to ATMI Conversational Server, Host Client Relinquishes Control.	2-35
CPI-C Programming Examples	2-36
ATMI Client Request/Response to Host CPI-C	2-37
ATMI Client Asynchronous Request/Response to Host CPI-C	2-39
ATMI Client Asynchronous Request/Response to Host CPI-C with No Reply . . .	2-41
ATMI Conversational Client to Host CPI-C, Server Gets Control	2-43
ATMI Conversational Client To Host CPI-C, Client Retains Control	2-45
ATMI Conversational Client to Host CPI-C, Client Grants/gets Control	2-47
Host CPI-C to ATMI Asynchronous Request/Response Server with No Reply . . .	2-49
Host CPI-C to ATMI Server Request/Response	2-51
Host CPI-C to ATMI Conversational Service, Client Retains Control	2-53
Host CPI-C ATMI to Conversational Service, Client Grants Control	2-55
Transactional ATMI Client Request/Response to Host CPI-C	2-57
Transactional ATMI Conversational Client to Host CPI-C, Server Gets Control . .	2-59
Transactional Host CPI-C to ATMI Conversational Server, Client Grants Control	2-61
CICS/ESA Mirror Transaction Examples	2-62
Implicit Attachment of TRANSID (Outbound Requests Only).	2-63
Explicit Attachment of TRANSID for Outbound Requests.	2-64
Explicit Attachment of TRANSID for Inbound Requests	2-65
Additional Information	2-66

ATMI to CPI-C Function Mapping

The following tables list the most common ATMI function calls and show how their parameters map to CPI-C verbs. The mappings are listed by function call in the following order:

- `tpcall()`
- `tpacall()` with or without reply
- `tpgetrply()`
- `tpservice()`
- `tpreturn()`
- `tpcancel()`
- `tpconnect()`
- `tpsend()`
- `tprecv()`
- `tpdiscon()`
- `tpforward()`

The tables show the parameters of the ATMI call, the contents or meaning of the parameters, and notes on usage with the CPI-C verbs.

Table 1-1 tpcall

tpcall()	Parameters	Contents	CPI-C Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.
idata		User data	This data is sent in CMSENDS until completely transmitted.
len		Length of User data	
odata		Reply data	CMRCV receives the data until it has been completely transmitted (data_received is set to CM_COMPLETE_DATA_RECEIVED) and return code is set to CM_OK or CM_DEALLOCATE_NORMAL.
olen		Reply data length	
flags	TPNOTRAN	Not part of a transaction	
	TPNOCHANGE	N/A	Local
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 1-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
svc		Service Name	Used in CMALLC to identify the CICS transaction to be invoked.
data		User data	This data is sent in CMSENDS until completely transmitted.
len		Length of user data	

Table 1-2 tpacall

tpacall()	Parameter	Contents	CPIC Notes
flags	TPNOREPLY	false	The last data is sent with a CMSEND with send_type set to CMSEND_AND_PREP_TO_RECEIVE. This changes the state of the conversation to receive and a CMRCV is issued to await the reply.
		true	Since no reply is expected, a CMDEAL deallocates the conversation after all data has been received.
	TPNOTRAN	Not part of a transaction	
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 1-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
cd		call descriptor	The call descriptor is mapped to the CONVID returned by the CMINIT when the LU6.2 was initiated.
data		User data	Data received from CMRCV if WHAT_RECEIVED set to DATA_COMPLETE.
len		Length of user data	

Table 1-3 tpgetrply

tpgetrply()	Parameters	Contents	CPIC Notes
flags	TPGETANY	If true, data is returned from any conversation. If false, data is returned from conversation associated with the <code>cd</code>	Data available on any conversation is returned to the requestor.
	TPNOCHANGE	Local to the requestor	Limited buffer types supported.
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 1-4 tpSERVICE

tpSERVICE()	Parameters	Contents	CPIC Notes
svcinfo		Service information and data	User Data captured from a CMRCV populates the TPSVCINFO structure user data area. Service characteristics are obtained from the service attributes in the DMCONFIG and UBBCONFIG files.
name		Service name	The service name associated with the 8 character RNAME sent from CICS.
data		User data	Data captured from CMRCV.
len		Length of user data	
cd		call descriptor	The call descriptor associated with the CONVID returned by the CMINIT when the LU6.2 was initiated.
appkey		32-bit key (if used)	For security.
cltid		set by Oracle Tuxedo	For security.

Table 1-4 tpservice

tpservice()	Parameters	Contents	CPIC Notes
flags	TPCONV	If true, service is conversational.	
	TPTRAN	N/A	.
	TPNOREPLY	If true, requestor not expecting a reply.	The conversation is terminated with a CMDEAL normal.
	TPSENDONLY	N/A	If set, the CPIC conversation in CICS should be in receive state. If not set, the CICS CPIC conversation state will be in send state.
	TPRECVONLY	N/A	If set, the CPIC conversation in CICS remains in send state.

Table 1-5 tpreturn

tpreturn()	Parameters	Contents	CPIC Notes
rval		TPSUCCESS	Set to TPSUCCESS when conversation terminates with a normal deallocation.
		TPSVCERR	Set to TPESVCERR when the conversation has terminated with a non-normal deallocation type or other error.
rcode		Set by the application	N/A
data		User data	Data is returned to the CICS transaction from a successful CMRCV with data received set to CM_DATA_COMPLETE and return code of CM_DEALLOCATE_NORMAL. If the service fails, no data is returned to the caller and the conversation is deallocated abnormally.
len		Length of data returned	0 < data <= 32K
flags		N/A	N/A

Table 1-6 tpcancel

tpcancel()	Parameters	Contents	CPIC Notes
cd		The connection descriptor on which a <code>tpgetreply()</code> is waiting.	CMDEAL abnormal is issued on the conversation with CONVID mapped from call descriptor.

Table 1-7 tpconnect

tpconnect()	Parameters	Contents	CPIC Notes
svc		The local service name representing the service to be invoked. in CICS	The name is used to find the RNAME. The RNAME should match the TPName in CICS and will be used by CMINIT and CMALLC to initiate and allocate the conversation.
data		User data	This data is sent in CMSENDS until completely transmitted.
len		Length of User data	
flags	TPNOTRAN	True	
	TPSENDONLY	If true, the conversation stays in or changes to send state	The conversation remains in send state. This is the default.
	TPRECVONLY	If true, the conversation stays in or changes to receive state	Immediately after the allocate Oracle Tuxedo sends a CMSEND with no data and send_type set to CM_SEND_AND_PREP_TO_RECEIVE.
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local

Table 1-8 tpsend

tpsend()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC on behalf of the <code>tpconnect()</code> .
data		User data	ASCII/EBCDIC conversion may be required before sending to CICS.
len		Length of User data	
flags	TPRECVONLY	If true, the conversation changes to receive state.	The state of the conversation changes from send to receive. A CMSEND is sent with <code>send_type</code> set to <code>CM_SEND_AND_PREP_TO_RECEIVE</code> .
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local
revent	TPEV_DISCONIMM	If set, the LU6.2 conversation has been terminated abnormally.	If the return code from a CMRCV is <code>deallocate_abnormal</code> , the conversation is terminated. A disconnect event is sent to the sending process.
	TPEV_SVCERR	If set, the LU6.2 conversation has been terminated abnormally.	Any return code other than <code>CM_OK</code> or <code>CM_DEALLOCATE_NORMAL</code> is treated as a <code>TPEV_SVCERR</code> .
	TPEV_SVCFAIL	If set, the LU6.2 conversation has been terminated abnormally.	If the return code from CMRCV is <code>CM_TP_NOT_AVAIL_NO_RETRY</code> or <code>CM_TP_RESOURCE_FAILURE_NO_RETRY</code> , <code>revent</code> is set to <code>TPEV_SVCFAIL</code> .

Table 1-9 tprecv

tprecv()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This locally assigned connection descriptor has been mapped to the CONVID returned in the CMINIT and CMALLC issued by the initiator of this conversation.
data		User data	Date to be received using a CMRCV_immediate and returned to the Oracle Tuxedo service.
len		Length of User data	
flags	TPNOCHANGE	Local	Must be a supported buffer type.
	TPNOBLOCK	N/A	Local
	TPNOTIME	N/A	Local
	TPSIGRSTRT	N/A	Local
revent	TPEV_DISCONIMM	If set, the LU6.2 conversation has been terminated abnormally.	If the return code from a CMSEND is deallocate_abnormal, the conversation is terminated. A disconnect event is sent to the sending process.
	TPEV_SENDOONLY	If set, the LU6.2 conversation changes to send if partner allows it.	The sending partner has sent a CMSEND with send_type set to CM_SEND_AND_PREP_TO_RECEIVE.
	TPEV_SVCERR	If set, the LU6.2 conversation has been terminated abnormally.	Any return code other than CM_OK or CM_DEALLOCATE_NORMAL is treated as a TPEV_SVCERR.
	TPEV_SVCFAIL		If the return code from CMRCV is CM_TP_NOT_AVAIL_NO_RETRY or CM_TP_RESOURCE_FAILURE_NO_RETRY, revent is set to TPEV_SVCFAIL.
	TPEV_SVCSUCC	If set, the conversation has completed normally.	The return code from CMRCV was set to CM_DEALLOCATE_NORMAL. This indicates that the sending TP has completed and deallocated the conversation normally.

Table 1-10 tpdiscn

tpdiscn()	Parameters	Contents	CPIC Notes
cd		The connection descriptor	This connection descriptor is mapped to the CONVID returned from CMINIT or CMAACP to the originator of the conversation.

Table 1-11 tpforward

tpforward ()	Parameters	Contents	CPIC Notes
svc		Service name	tpforward() is treated as if it were a tpacall(). A CMINIT and subsequent CMALLC are issued to initialize and allocate a session for a conversation. ClientID must be propagated to the CICS transaction in a TPSVCINFO record.
data		User data	Data is sent using CMSEND. The last CMSEND is sent with send_type of deallocate_normal.
len		Length of data returned	
flags		Refer to tpacall()	

Application-to-Application Programming Examples

This section provides the following transaction scenarios for the programming environments supported by Oracle Tuxedo Mainframe Adapter for SNA:

- [“Distributed Program Link \(DPL\) Examples”](#)
- [“Distributed Transaction Processing \(DTP\) Examples”](#)
- [“CPI-C Programming Examples”](#)
- [“CICS/ESA Mirror Transaction Examples”](#)

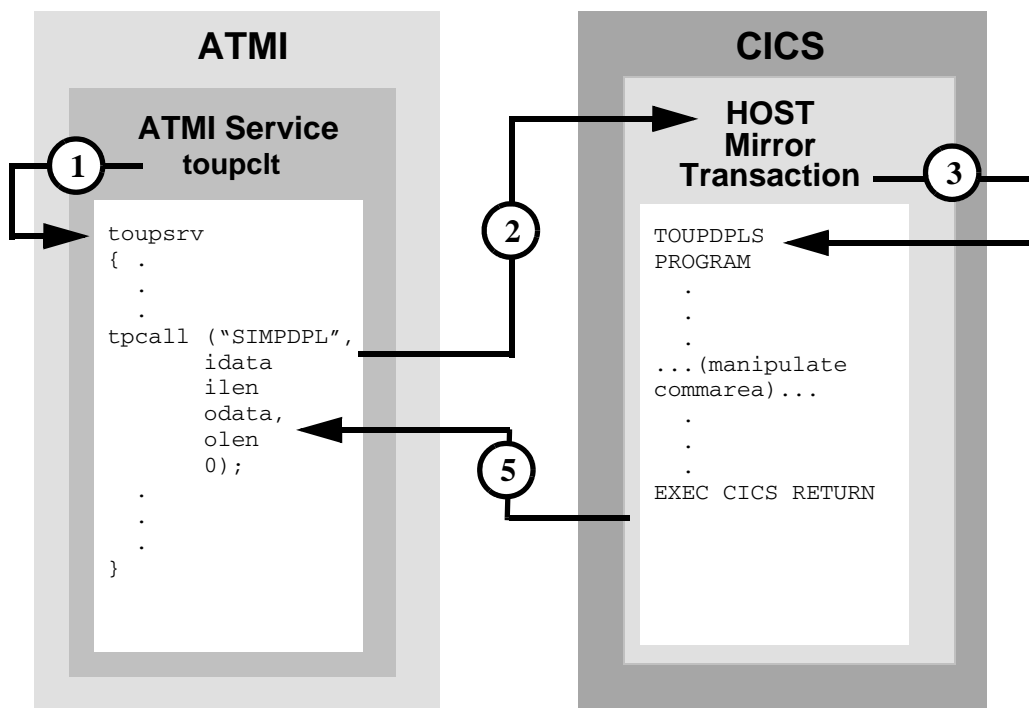
Caution: The scenarios in this section demonstrate how ATMI calls relate to CICS/ESA programming structures. They are not intended for use in developing application code, or for the replacement of existing application code. The use of any of these examples in actual situations may have unpredictable results.

Each example provides a graphical illustration of the scenario followed by a description of each step of the scenario.

Distributed Program Link (DPL) Examples

The examples in this section represent a few of the many programming scenarios available for using DPL and ATMI service invocations. These examples employ the most natural and efficient approaches.

ATMI Client Request/Response to CICS/ESA DPL



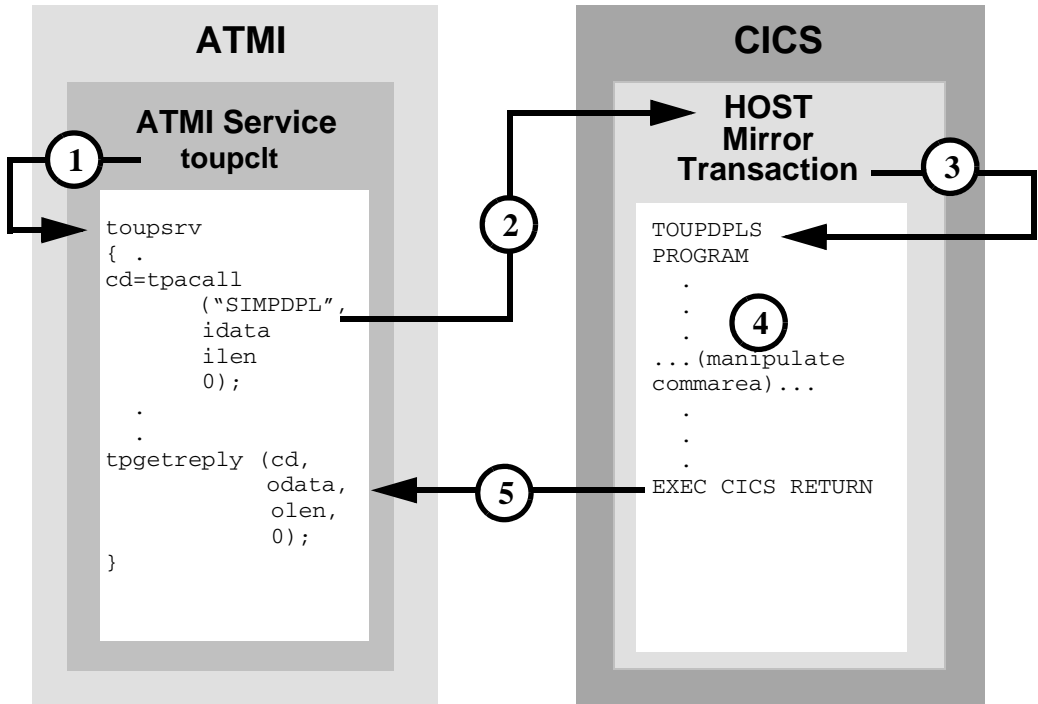
DMCONFIG File Entry

DM_REMOTE_SERVICES

SIMPDPDPL RNAME=TOUPDPLS FUNCTION=DPL CONV=N

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpcall` for `SIMPDPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.
5. The CICS/ESA server returns the `commarea` into the client's `odata` buffer.

ATMI Client Asynchronous Request/Response to CICS/ESA DPL



DMCONFIG File Entry

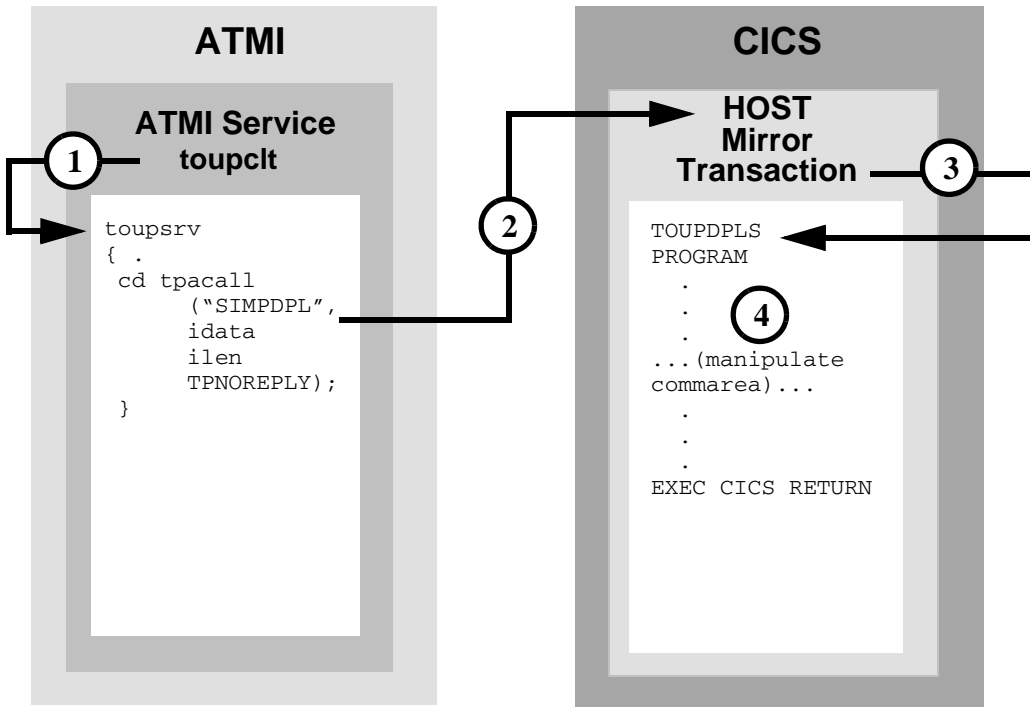
```
DM_REMOTE_SERVICES
```

```
SIMPDPL RNAME=TOUPDPLS FUNCTION=DPL CONV=N
```

1. ATMI client invokes `toupshr` service.
2. The `toupshr` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
4. The `TOUPDPLS` program processes data.

5. The CICS/ESA system returns the commarea into the client's `tpgetreply odata` buffer.

ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DPL



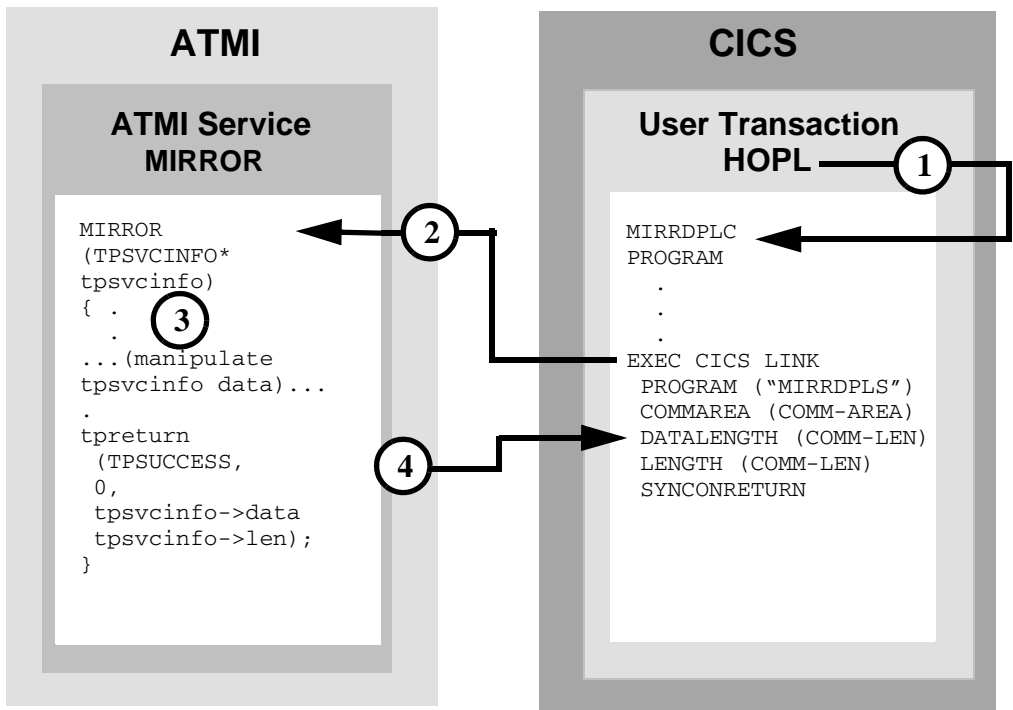
DMCONFIG File Entry

```
DM_REMOTE_SERVICES
SIMPDPL RNAME=TOUPDPLS FUNCTION=DPL CONV=N
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `toupsrv` service uses `TPNOREPLY` to specify that no reply is expected.

3. Host mirror transaction starts TOUPDPLS program and passes `idata` buffer contents for processing.
4. The TOUPDPLS program processes data.

CICS/ESA DPL to ATMI Request/Response Server



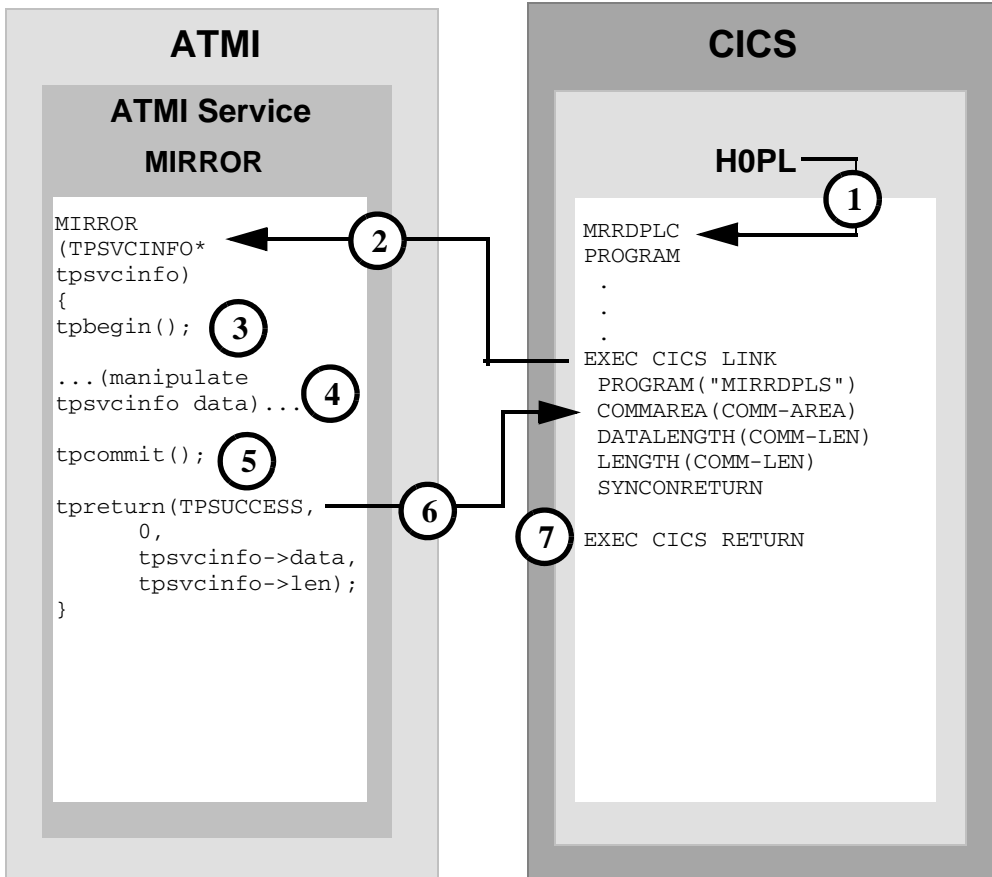
DMCONFIG File Entry

```
DM_LOCAL_SERVICES
MIRROR RNAME=MIRRDPLS CONV=N
```

1. User-entered HOPL invokes MIRRDPLC program.
2. The EXEC CICS LINK command causes the advertised service mapped to MIRRDPLS (in the DM_LOCAL_SERVICES section of the DMCONFIG file) to execute.
3. The MIRROR service processes the data received in the service TPSVCINFO data buffer from the EXEC CICS LINK.

4. The `tpreturn` call returns the data into the `COMM-AREA` buffer.

CICS/ESA DPL to ATMI Request/Response Server, Service in Autonomous Transaction



DMCONFIG File Entry

```

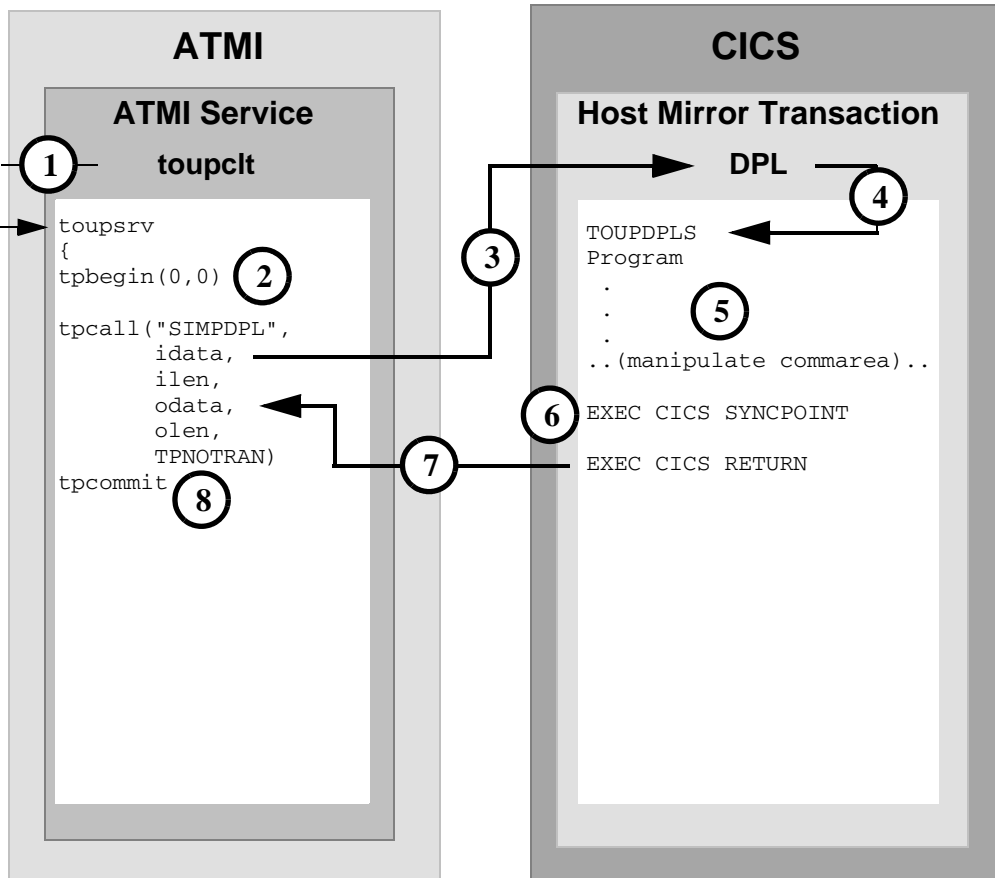
DM_LOCAL_SERVICES
MIRROR RNAME=MIRRDPLC CONV=N

```

1. User-entered `H0PL` invokes `MIRRDPLC` program.

2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file) to execute. The `SYNCONRETURN` option indicates that the invoked service will not participate in the CICS/ESA transaction.
3. The `MIRROR` service request `tpbegin` incorporates all further operations in a transaction.
4. The `MIRROR` service processes the data.
5. The `tpcommit` indicates the end of the transaction; all updates performed within the service transaction are to be committed.
6. The `tpreturn` call returns the data into the `commarea` buffer.
7. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.

ATMI Client Request/Response to CICS/ESA DPL, Autonomous Transaction



DMCONFIG File Entry

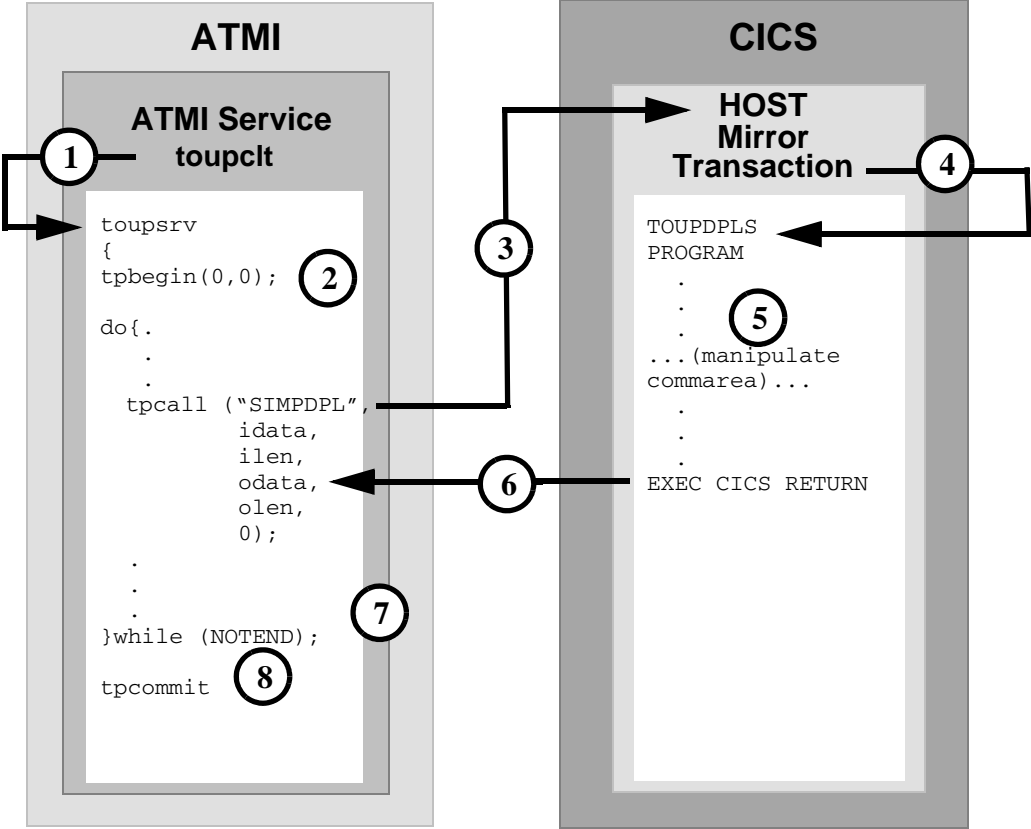
```
DM_REMOTE_SERVICES
```

```
SIMPDPL RNAME=TOUPDPLS FUNCTION=DPL CONV=N
```

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpbegin` to start the transaction.

3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPNOTRAN` parameter indicates the CICS/ESA application does not participate in the service transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing.
5. The `TOUPDPLS` program processes data.
6. The `EXEC CICS SYNCPOINT` is an explicit commit request. All updated resources in the CICS/ESA transaction are committed.
7. The CICS/ESA server returns the `commarea` into the client's `odata` buffer.
8. The `toupsrv` service `tpcommit` request signals the successful completion of the transaction, causing a commit of its own updated resources.

Transactional ATMI Client Multiple Requests/Responses to CICS/ESA DPL



DMCONFIG File Entry

```

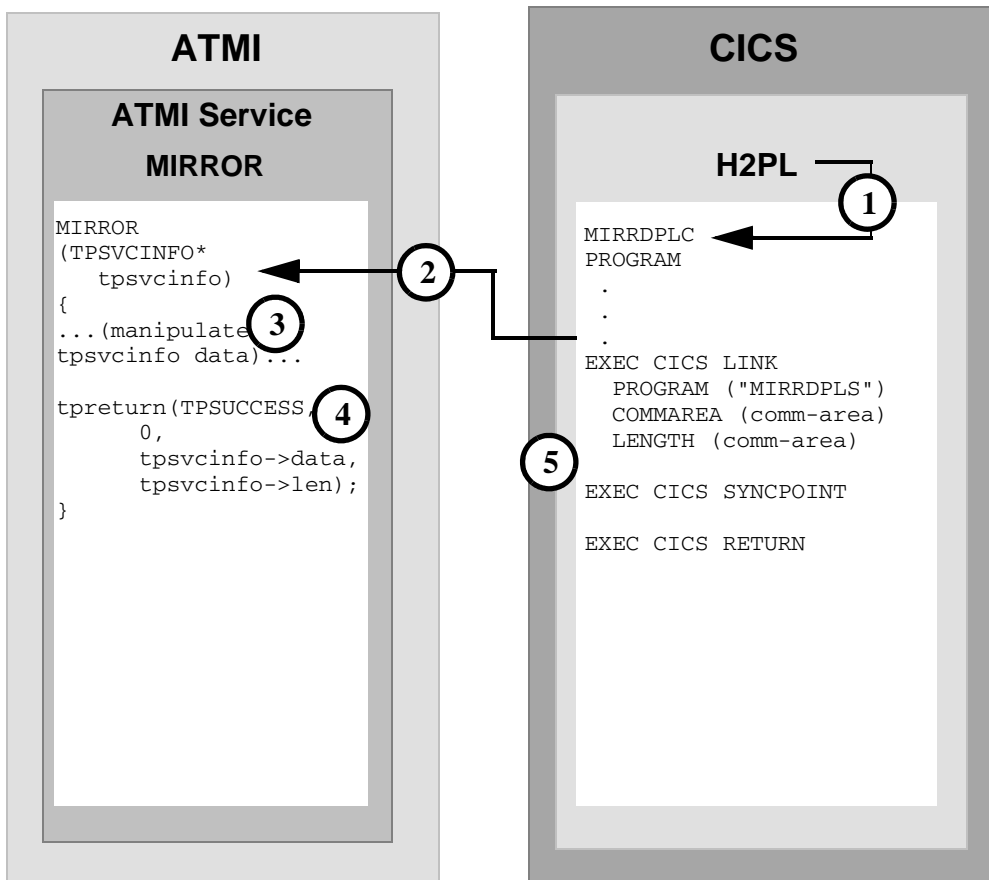
DM_REMOTE_SERVICES
SIMPDPL RNAME=TOUPDPLS FUNCTION=DPL CONV=N

```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.

3. The `toupsrv` service issues `tpcall` for `SIMPDPL`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `tpcall` is requested multiple times within the same transaction.
4. Host mirror transaction starts `TOUPDPLS` program and passes `idata` buffer contents for processing. The host mirror transaction remains as a long-running task to service all further requests on the transaction.
5. The `TOUPDPLS` program processes data.
6. The CICS/ESA system returns the `commarea` into the client's `odata` buffer.
7. Step 3 through Step 6 are repeated until the `toupsrv` service loop end conditions are met.
8. The `tpcommit` request indicates the successful completion of the transaction, causing a commit of its own resources and the resources held by the host mirror transaction.

Transactional CICS/ESA DPL to ATMI Request/Response Server



DMCONFIG File Entry

```

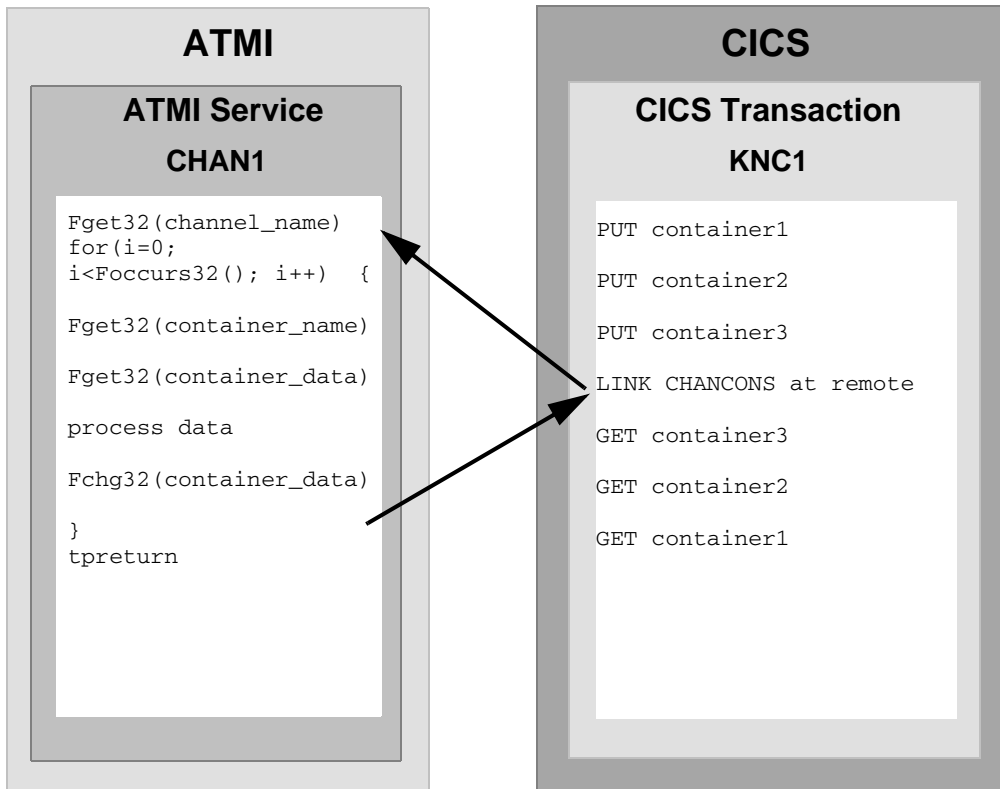
DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=N

```

1. User-entered H2PL invokes MIRRDPLC program.

2. The `EXEC CICS LINK` command causes the advertised service mapped to `MIRRDPLS` (in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file) to execute. The invoked service participates in the CICS/ESA transaction.
3. The `MIRROR` service processes the data.
4. The `tpreturn` call returns the data into the `commarea` buffer.
5. The `EXEC CICS SYNCPOINT` is an explicit commit request indicating a successful end of the conversation. All updated resources in the transaction are committed.

CICS/ESA DPL to ATMI Requests/Responses Server for Channel /Container



DMCONFIG File Entry

```
*DM_LOCAL_SERVICES
CHAN1
    CONV=N
    RNAME= "KNC1 : CHANCONS "
    INBUFTYPE= "FML32 "
    OUTBUFTYPE= "FML32 "
```

1. User-entered KNC1 invokes CHANCONC program.

2. Three containers are put in a channel.
3. The EXEC CICS LINK command causes the advertised service mapped to CHANCONS to execute.
4. The CHAN1 service gets all containers from the channel, processes data of them, and puts them back to channel.
5. The tpreturn call returns the data into the commarea buffer.
6. The CHANCONC gets all containers from the channel and CICS RETRUN.

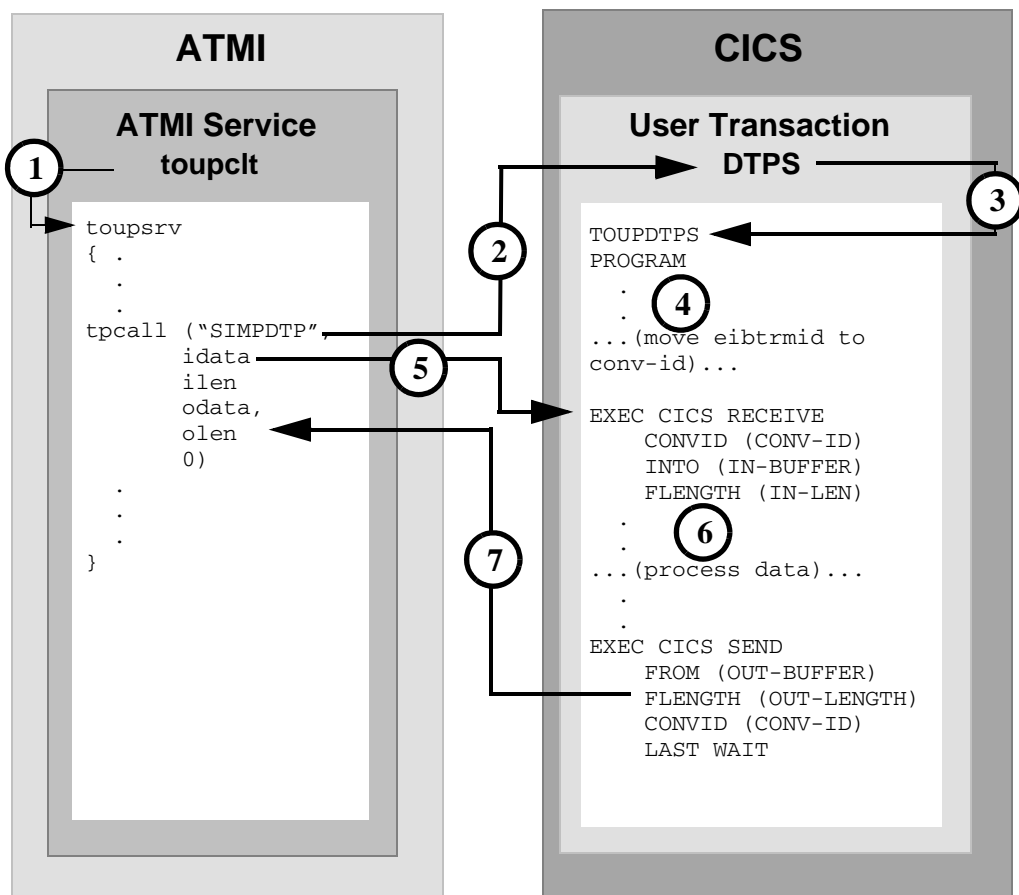
Distributed Transaction Processing (DTP) Examples

The following examples represent programming scenarios for using DTP and ATMI service invocations.

Although it is most suited for the DPL environment, the `tpcall` is usually used for the DPL environment, it can also be used for a request/response to a DTP server.

The examples in this section represent some of the programming scenarios available for using DTP and ATMI service invocations. These examples employ the most natural and efficient approaches.

ATMI Client Request/Response to CICS/ESA DTP



DMCONFIG File Entry

```

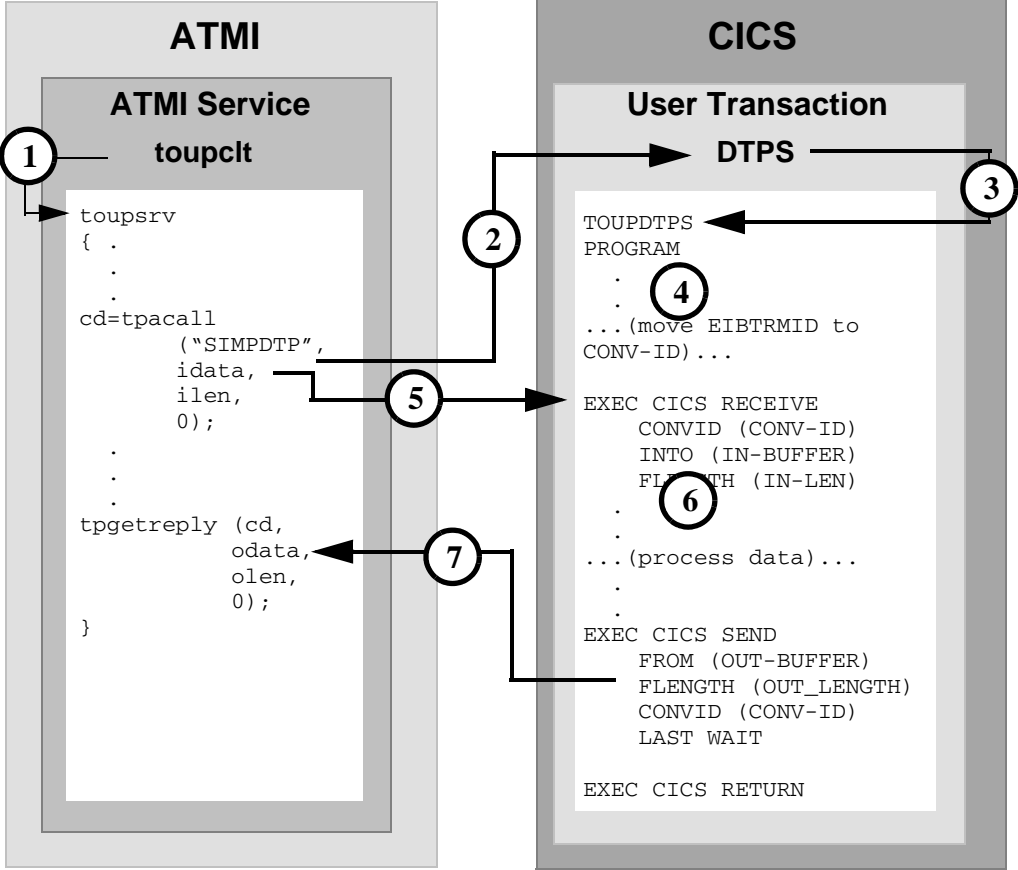
DM_REMOTE_SERVICES

SIMPDTP RNAME=DTPTS FUNCTION=APPC CONV=N
  
```

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. User transaction `DTPTS` starts `TOUPDTPS` program.

4. It is recommended you save the `eibtrmid` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `odata` buffer. `LAST` indicates the conversation is finished. `WAIT` suspends processing until the data has successfully been received.

ATMI Client Asynchronous Request/Response to CICS/ESA DTP



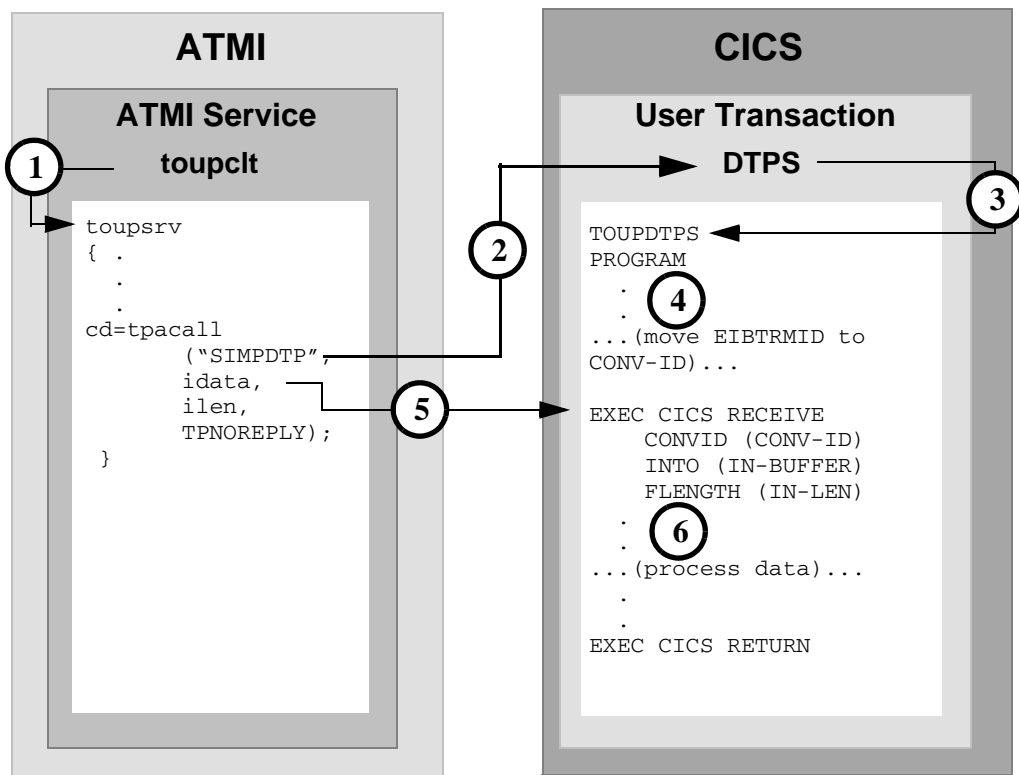
DMCONFIG File Entry

```
DM_REMOTE_SERVICES
SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=N
```

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpacall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.

3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation in your CICS/ESA APPC verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tpgetreply odata` buffer. `LAST` indicates the conversation is finished. `WAIT` suspends processing until the data has successfully been received.

ATMI Client Asynchronous Request/Response with No Reply to CICS/ESA DTP



DMCONFIG File Entry

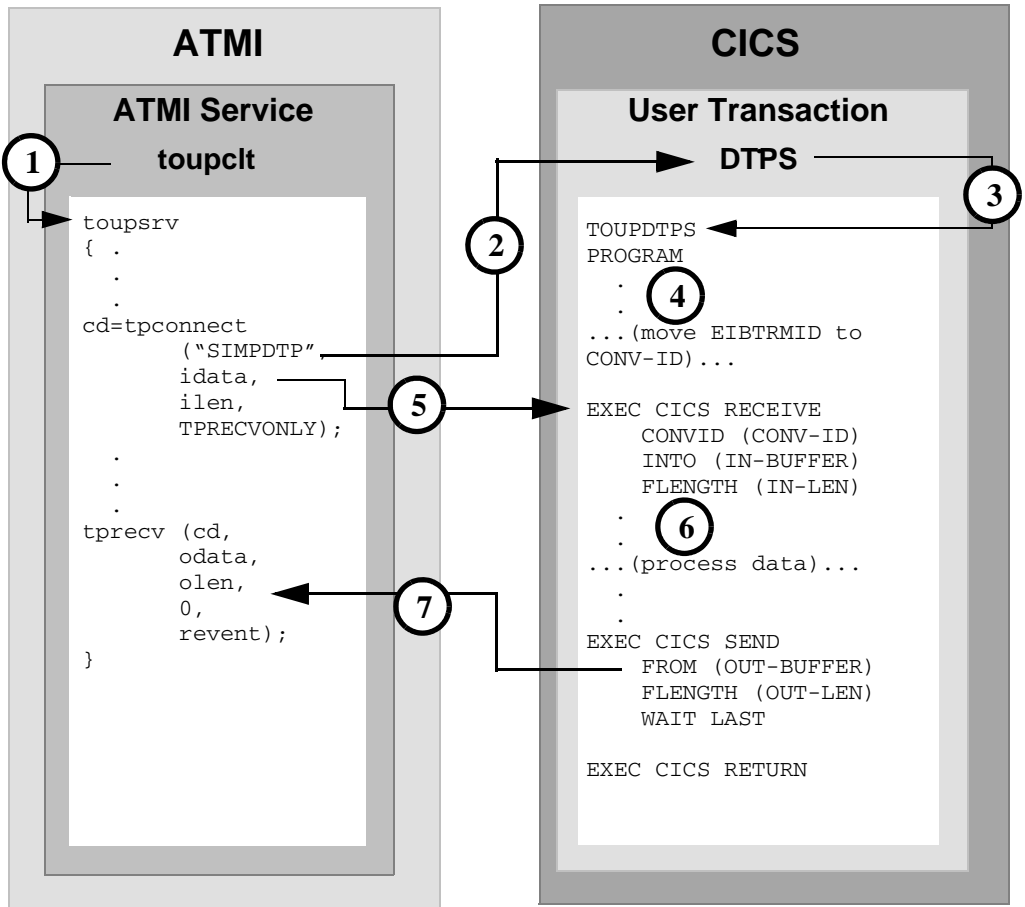
```
DM_REMOTE_SERVICES
```

```
SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=N
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` with a `TPNOREPLY` request for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your CICS/ESA APPC verbs.

5. The EXEC CICS RECEIVE command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.

ATMI Conversational Client to CICS/ESA DTP, Server Gets Control



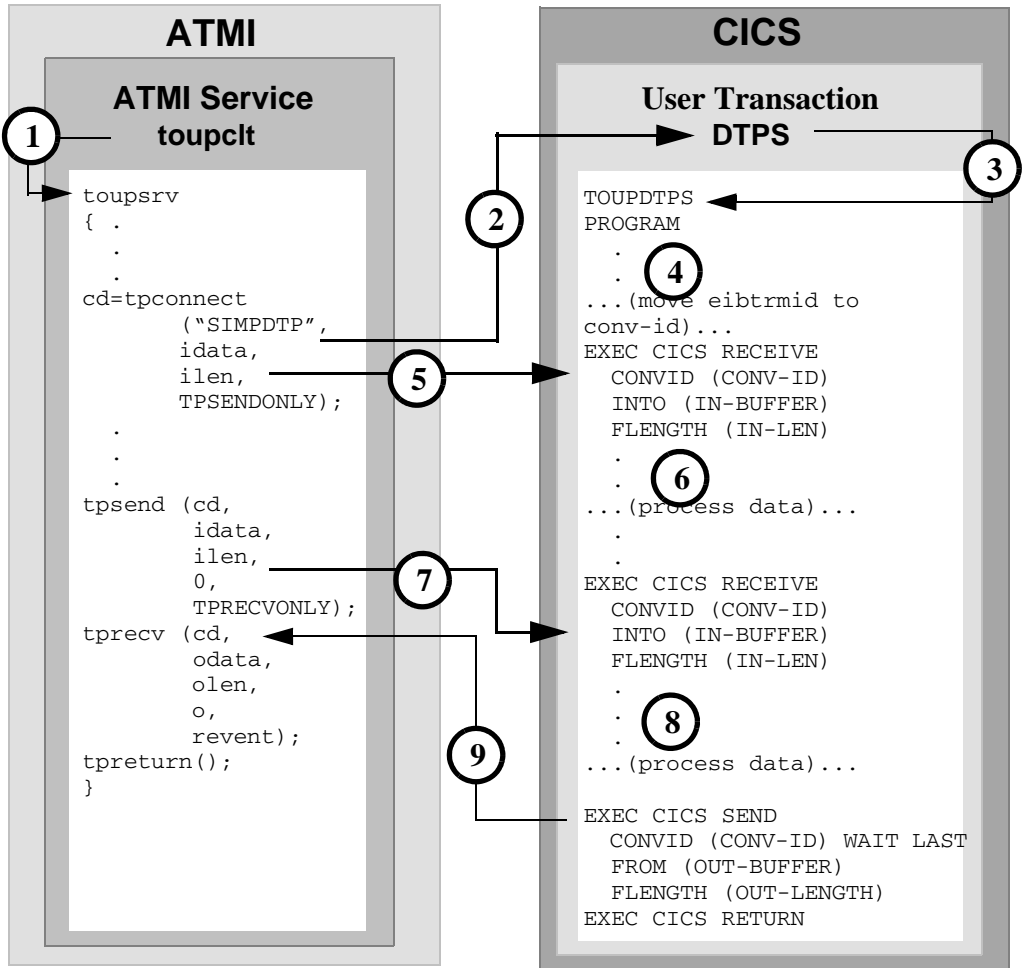
DMCONFIG File Entry

```

DM_REMOTE_SERVICES
SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=Y
    
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` flag indicates the server gets control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your `CICS/ESA APPC` verbs.
5. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv` `odata` buffer. `WAIT` suspends processing in `TOUPDTPS` until the data has successfully been received. `LAST` indicates the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.

ATMI Conversational Client to CICS/ESA DTP, Client Sends/Receives Data



DMCONFIG File Entry

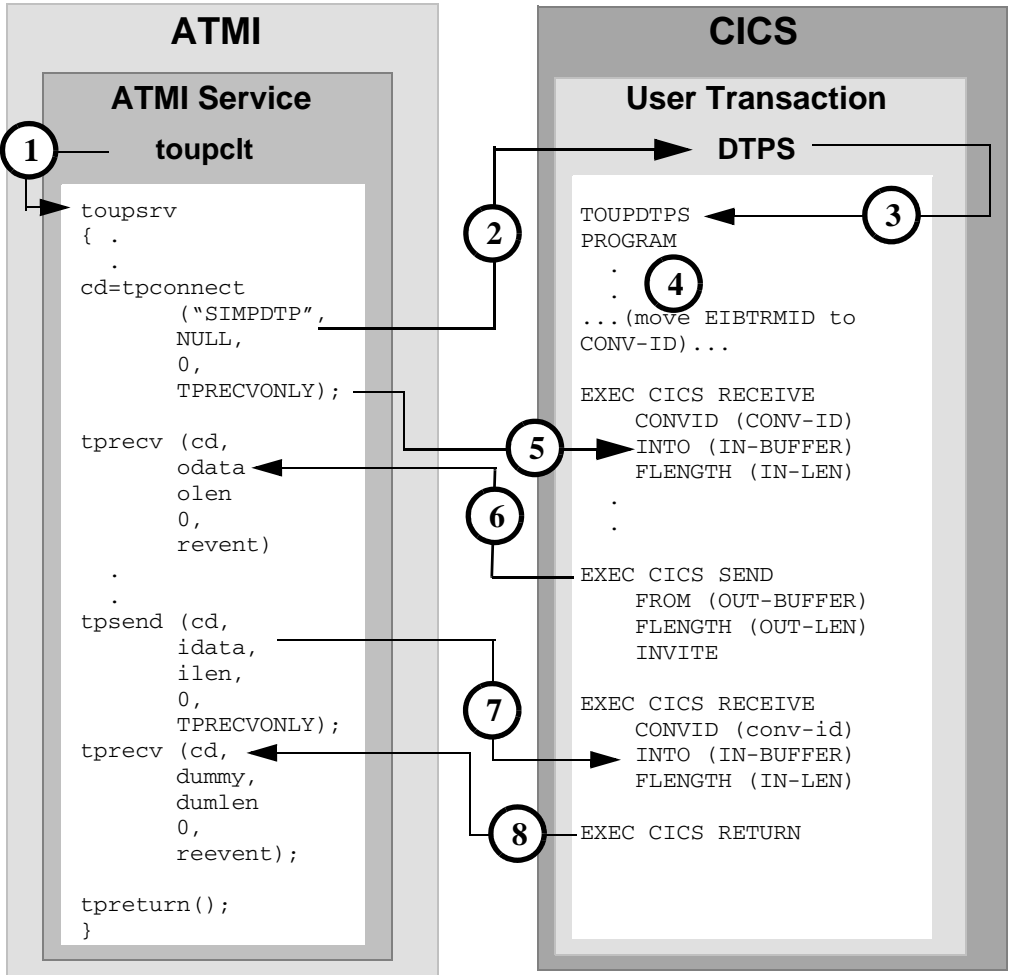
DM_REMOTE_SERVICES

SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=Y

1. ATMI client invokes toupleclt service.

2. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPSENDONLY` indicates the client retains control and continues to send data. Data is sent on the `tpconnect` in the `idata` buffer.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your `CICS/ESA APPC` verbs.
5. The `EXEC CICS RECEIVE` command receives the `tpconnect idata` buffer contents for processing.
6. The `TOUPDTPS` program processes data.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`.
8. The server processes the data.
9. The `EXEC CICS SEND WAIT LAST` returns `OUT-BUFFER` data in the `tprecv odata` buffer, along with notification that the conversation is over.

ATMI Conversational Client to CICS/ESA DTP, Client Grants Control



DMCONFIG File Entry

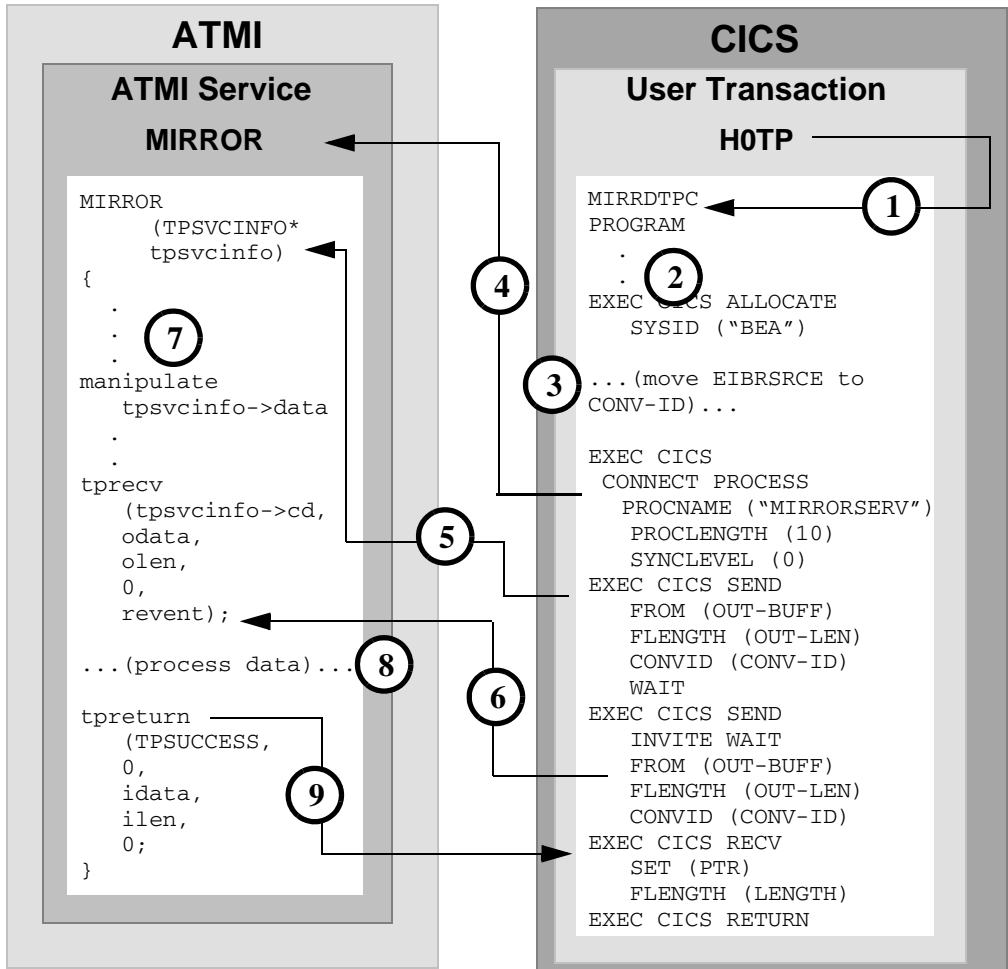
DM_REMOTE_SERVICES

SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=Y

1. ATMI client invokes `toupsrv` service.

2. The `tpoupsrv` service issues `tpconnect` for `SIMPDTTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` indicates the server gets control and the first conversation verb `tpoupsrv` can issue is `tprecv`.
3. User transaction `DTPS` starts `TOUPDTPS` program.
4. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your `CICS/ESA APPC` verbs.
5. The `EXEC CICS RECEIVE` command receives a `send state` indicator from the `tpconnect` `TPRECVONLY` flag. No data is received into the `INBUFFER`.
6. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv` `odata` buffer. The `EXEC CICS SEND` command relinquishes control to the client by using the `INVITE` option. This is communicated to the `tprecv` as `TPEV_SENDOONLY`.
7. The `EXEC CICS RECEIVE` command receives the `tpsend idata` contents into the server's `IN-BUFFER`, along with notification that the server has relinquished control.
8. The `EXEC CICS RETURN` ends the conversation, communicated to the `tprecv` as `TPEV_SVCSUCC`.

CICS/ESA DTP to ATMI Conversational Server, Client Retains Control



DMCONFIG File Entry

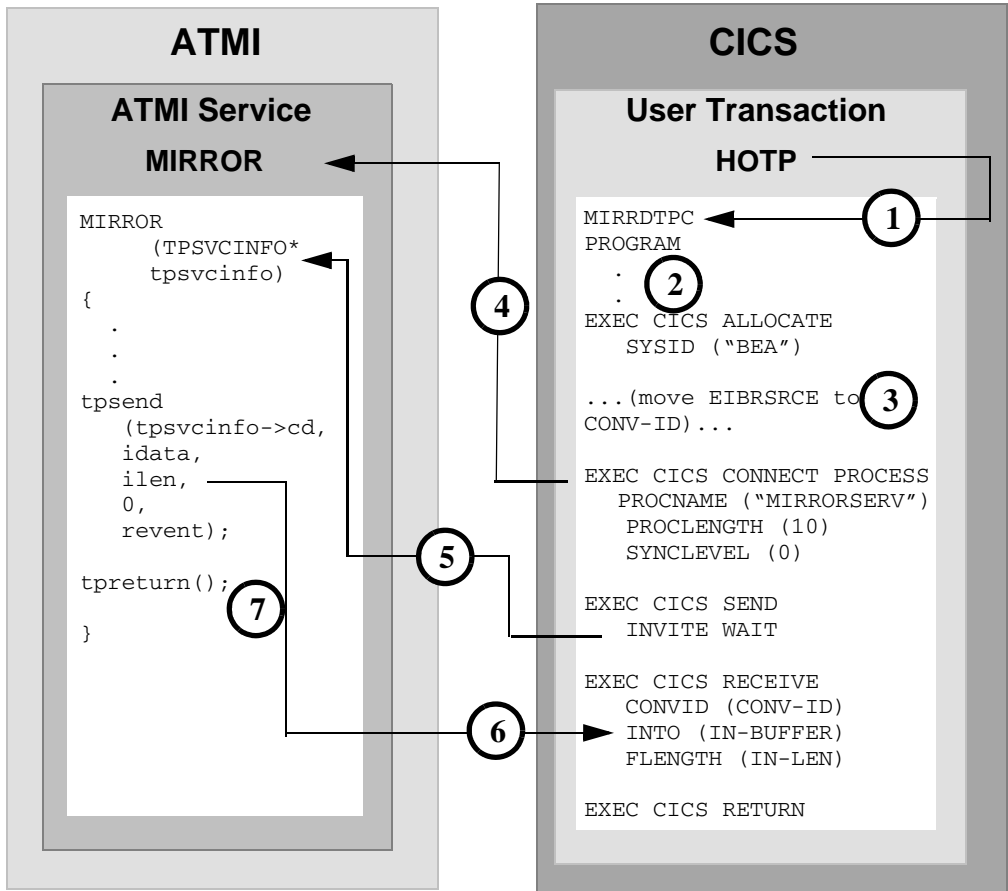
```

DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=Y
    
```

1. User-entered H0TP invokes MIRRDTPC program.
2. The EXEC CICS ALLOCATE acquires a session to the remote Tuxedo domain.

3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation in your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRROR` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
5. Execute the `EXEC CICS SEND` command to send the contents of the `OUT-BUFFER` to the Tuxedo service in the `tpsvcinfol->data` buffer. The contents might be sent immediately.
6. The `EXEC CICS SEND INVITE WAIT` command sends `out-buff` contents into the `tprecvodata` buffer. The `INVITE` parameter relinquishes control of the conversation, seen as a `TPEV_SENDOONLY` in the `reevent` parameter on the `tprecv` command. The data is sent immediately, along with the data from the previous `SEND` operation.
7. The Tuxedo service processes data.
8. The CICS/ESA server processes data.
9. The ATMI `tpreturn` data returns data to the `EXEC CICS RECEIVE`, along with notification that the conversation completed successfully.

CICS/ESA DTP to ATMI Conversational Server, Client Relinquishes Control



DMCONFIG File Entry

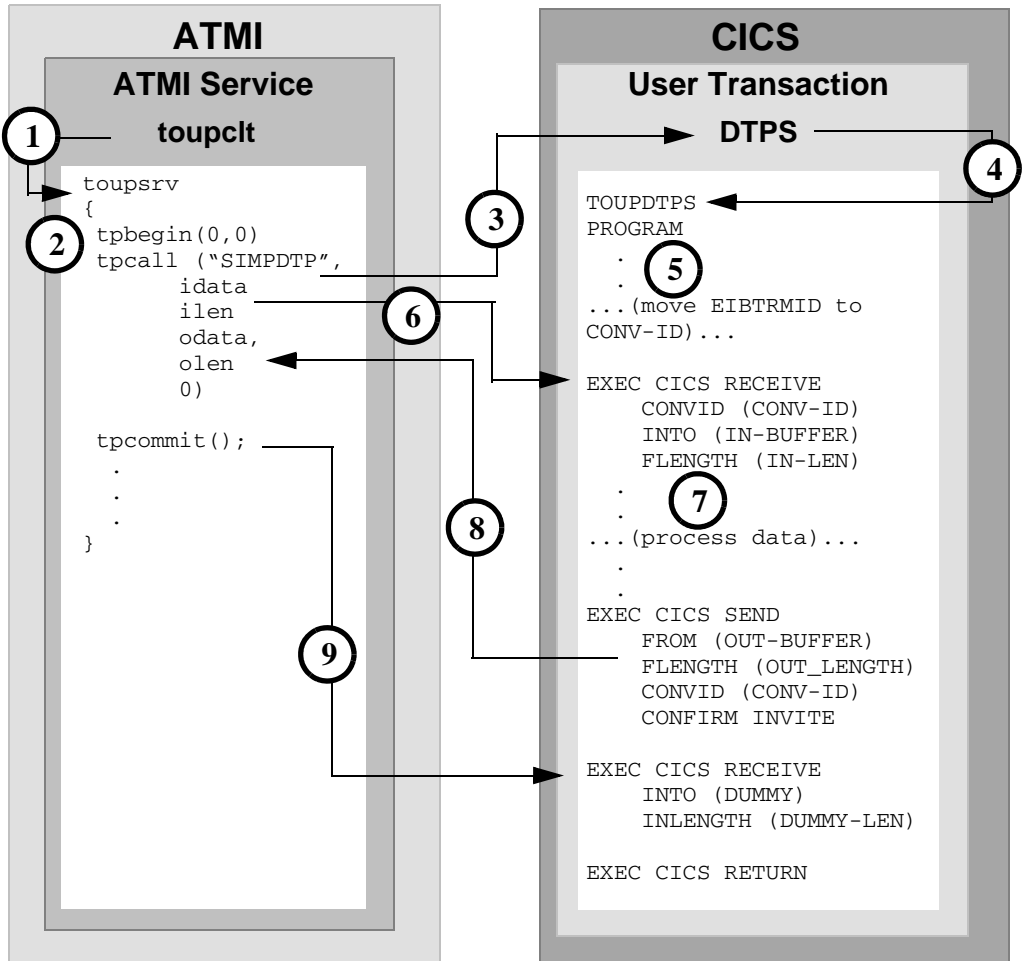
```

DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=Y
  
```

1. User-entered HOTP invokes MIRRTPC program.
2. The EXEC CICS ALLOCATE acquires a session to the remote Tuxedo domain.

3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation in your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRROR` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
5. The `EXEC CICS SEND` command relinquishes control with the `INVITE WAIT` option.
6. The `EXEC CICS RECEIVE` command receives the `tpsend idata` buffer contents into the `IN-BUFFER`.
7. The `tpreturn` request tears down the conversation and indicates on the `EXEC CICS RECEIVE` that the conversation is over.

Transactional ATMI Client Request/Response to CICS/ESA DTP



DMCONFIG File Entry

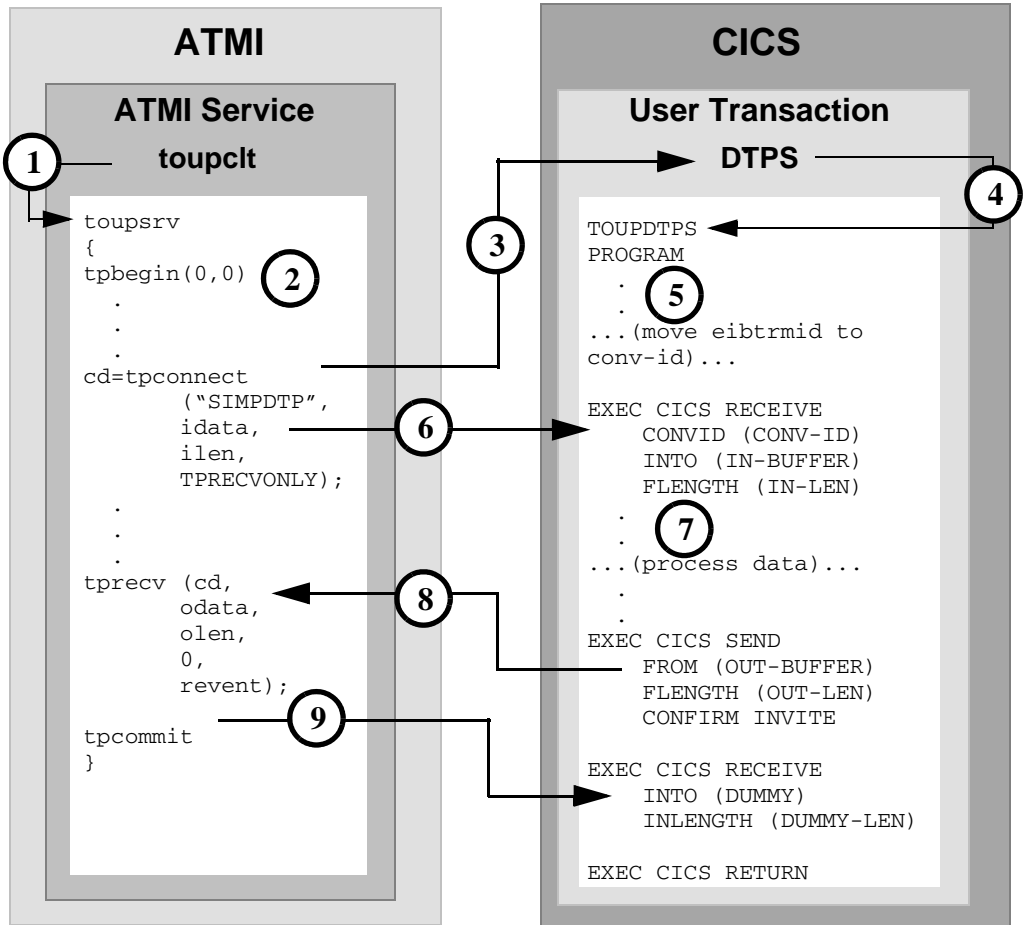
DM_REMOTE_SERVICES

SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=N

Note: This is not the recommended method of performing a DTP transactional service. Please refer to the transactional DPL using request/response for the recommended method.

1. ATMI client `toupclt` invokes `toupsrv` service. (Note that each `tpcall` made in the program must be bookended with a `tpbegin` and a `tpcommit`.)
2. The service issues `tpbegin` to start a transaction.
3. The `toupsrv` service issues `tpcall` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
4. User transaction `DTPS` starts `TOUPDTPS` program.
5. Save the `EIBTRMID` to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUUFER` contents into the clients `odata` buffer. `CONFIRM` indicates the conversation is finished. `INVITE` allows the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

Transactional ATMI Conversational Client to CICS/ESA DTP, Server Gets Control



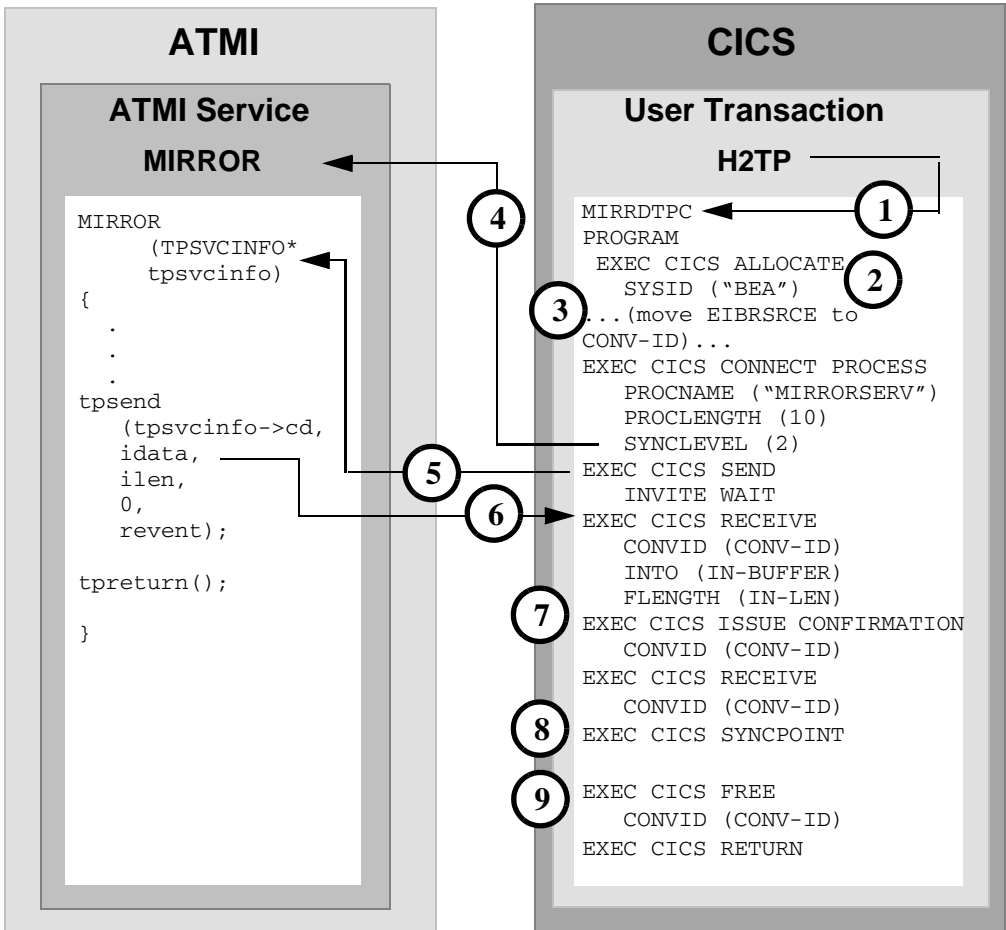
DMCONFIG File Entry

```
DM_REMOTE_SERVICES
SIMPDTP RNAME=DTPS FUNCTION=APPC CONV=Y
```

1. ATMI client invokes `toupshr` service.
2. The `toupshr` service issues `tpbegin` to start the transaction.

3. The `toupsrv` service issues `tpconnect` for `SIMPDTP`, which is advertised in the `DM_REMOTE_SERVICES` section of `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb `toupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
4. User transaction `DTPS` starts `TOUPDTPS` program.
5. It is recommended you save the `EIBTRMID` to a program variable. This value may be used to identify the specific conversation on your `CICS/ESA APPC` verbs.
6. The `EXEC CICS RECEIVE` command receives the `idata` buffer contents for processing.
7. The `TOUPDTPS` program processes data.
8. The `EXEC CICS SEND` command returns the `OUT-BUFFER` contents into the clients `tprecv` `odata` buffer. `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`. `INVITE` enables the client to respond with a `COMMIT` request.
9. The `toupsrv` service issues `tpcommit` to end the transaction. The `COMMIT` is received on the `EXEC CICS RECEIVE` verb and the server issues `EXEC CICS RETURN` to commit the resources, terminate the transaction, and free the outstanding conversation.

Transactional CICS/ESA DTP to ATMI Conversational Server, Host Client Relinquishes Control



DMCONFIG File Entry

```
DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=Y
```

1. User-entered H2TP invokes MIRRDTPC program.
2. The EXEC CICS ALLOCATE acquires a session to the remote Tuxedo domain.

3. Save the conversation ID returned in `EIBRSRCE` to a program variable. This value is used to identify the specific conversation on your CICS/ESA APPC verbs.
4. The `EXEC CICS CONNECT PROCESS` command initiates the advertised service mapped to `MIRRDTPS`. The `SYNLEVEL(2)` parameter indicates the inclusion of the ATMI service in the CICS/ESA transaction.
5. The `EXEC CICS SEND INVITE WAIT` command causes the client to immediately relinquish control to the Tuxedo server. This is communicated to the service in `TPSVCINFO` as `TPSENDONLY`. No data is sent to the server on this request.
6. The `EXEC CICS RECEIVE` command receives the `tpsend` idata buffer contents into the `IN-BUFFER`. The `EXEC CICS RECEIVE` command receives a confirm request indicating the conversation should be terminated.
7. The `EXEC CICS ISSUE CONFIRMATION` verb responds positively to the confirm request.
8. The `EXEC CICS SYNCPOINT` is an explicit commit request to end the conversation and update all resources in the transaction.
9. The `EXEC CICS FREE` verb explicitly frees the outstanding conversation.

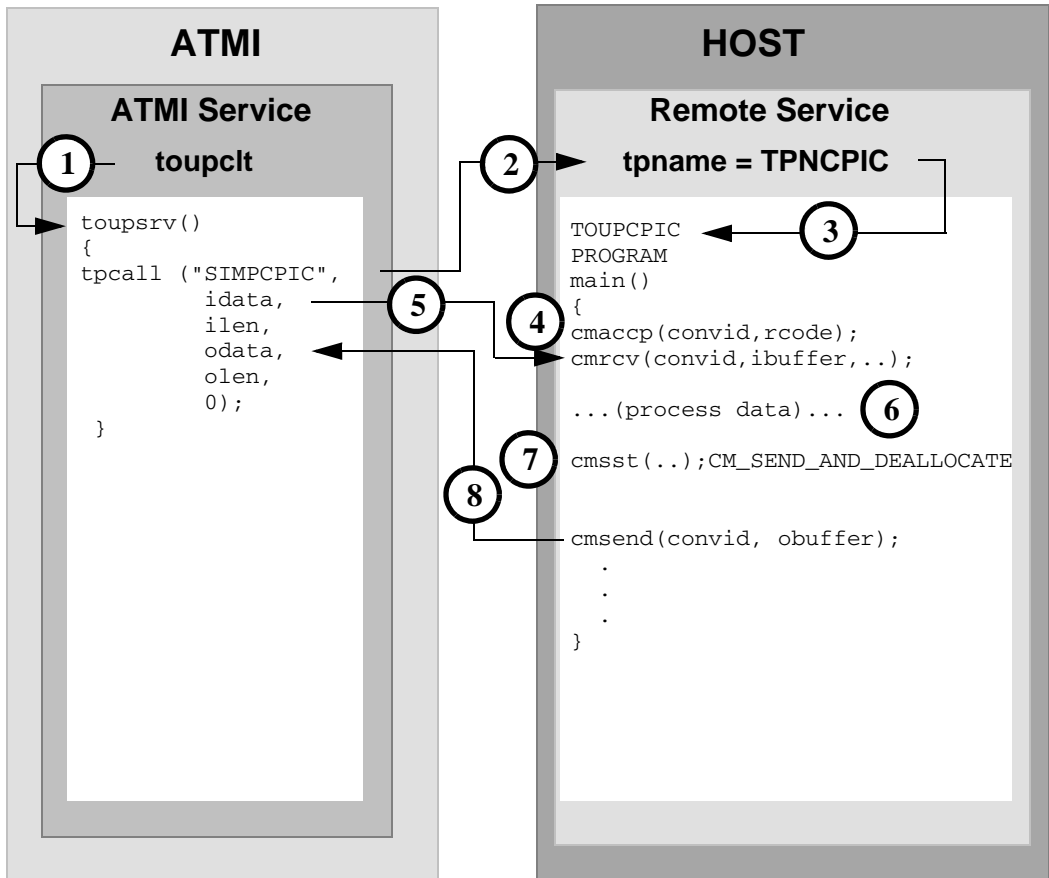
CPI-C Programming Examples

The examples in this section show the protocol exchanges between the local ATMI platform and remote host application program. The type of ATMI service request determines the nature of the client/server communication model. For requests initiated by the host application, the configuration information for the local service determines the protocol exchanges on the conversation.

Although it is most suited for the DPL environment, the `tpcall` is usually used in the DPL environment but can also be used for a request/response to an APPC server.

The examples in this section represent a few of the many programming scenarios available for using CPI-C and ATMI service invocations. These examples employ the most natural and efficient approaches.

ATMI Client Request/Response to Host CPI-C



DMCONFIG File Entry

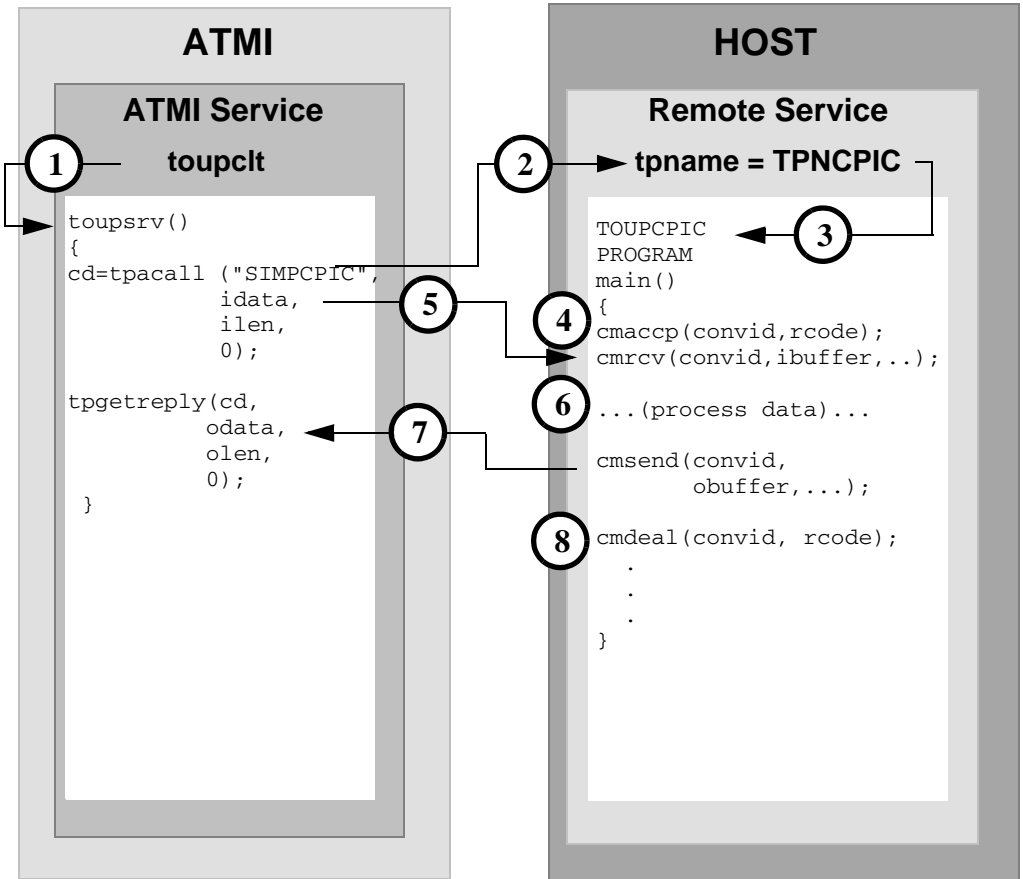
```
DM_REMOTE_SERVICES
```

```
SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=N
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.

3. The remote service with the `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing
6. The `TOUPCPIC` program processes data.
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_DEALLOCATE`.
8. The `cmsend` request returns the `obuffer` contents into the client `odata` buffer. The buffer is flushed, and the conversation ended.

ATMI Client Asynchronous Request/Response to Host CPI-C



DMCONFIG File Entry

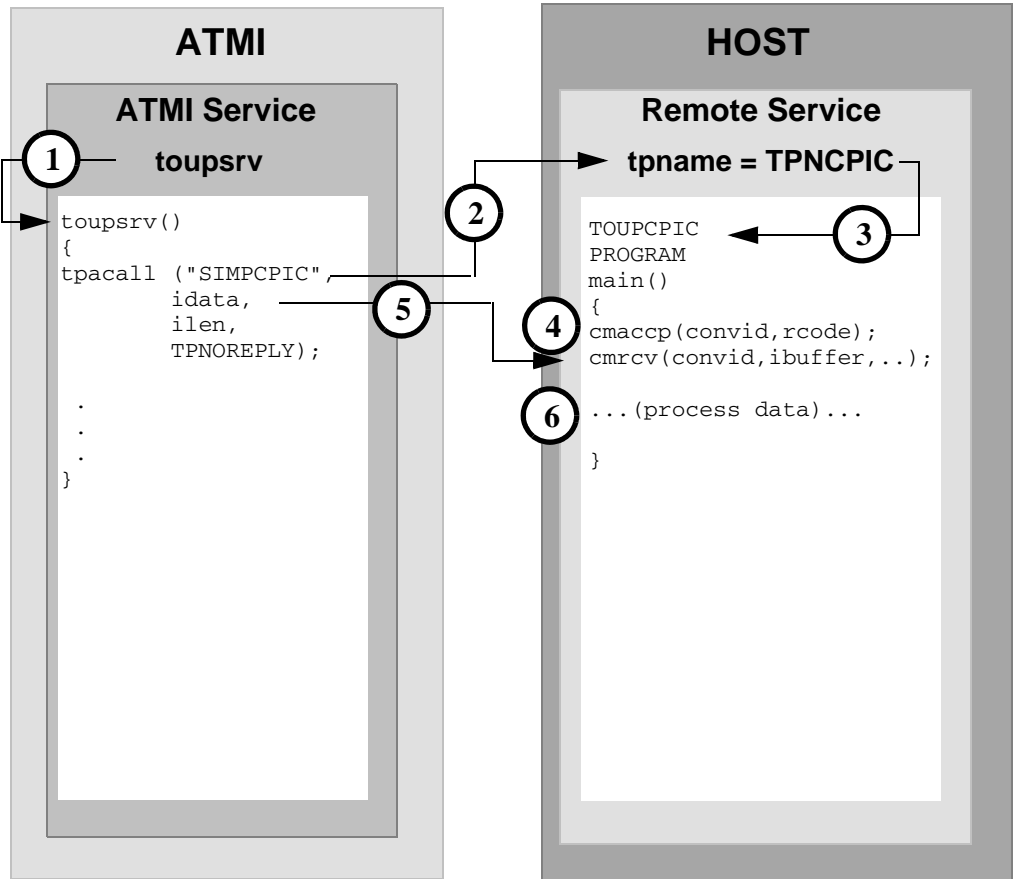
```
DM_REMOTE_SERVICES
```

```
SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=N
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpacall` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.

4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data.
7. The `cmsend` command returns the `obuffer` contents into the client `tpgetreply odata` buffer. The data may not be immediately sent to the `tpgetreply odata` buffer on this request.
8. The `cmdeal` flushes the data to the client, and indicates the conversation is finished.

ATMI Client Asynchronous Request/Response to Host CPI-C with No Reply



DMCONFIG File Entry

```

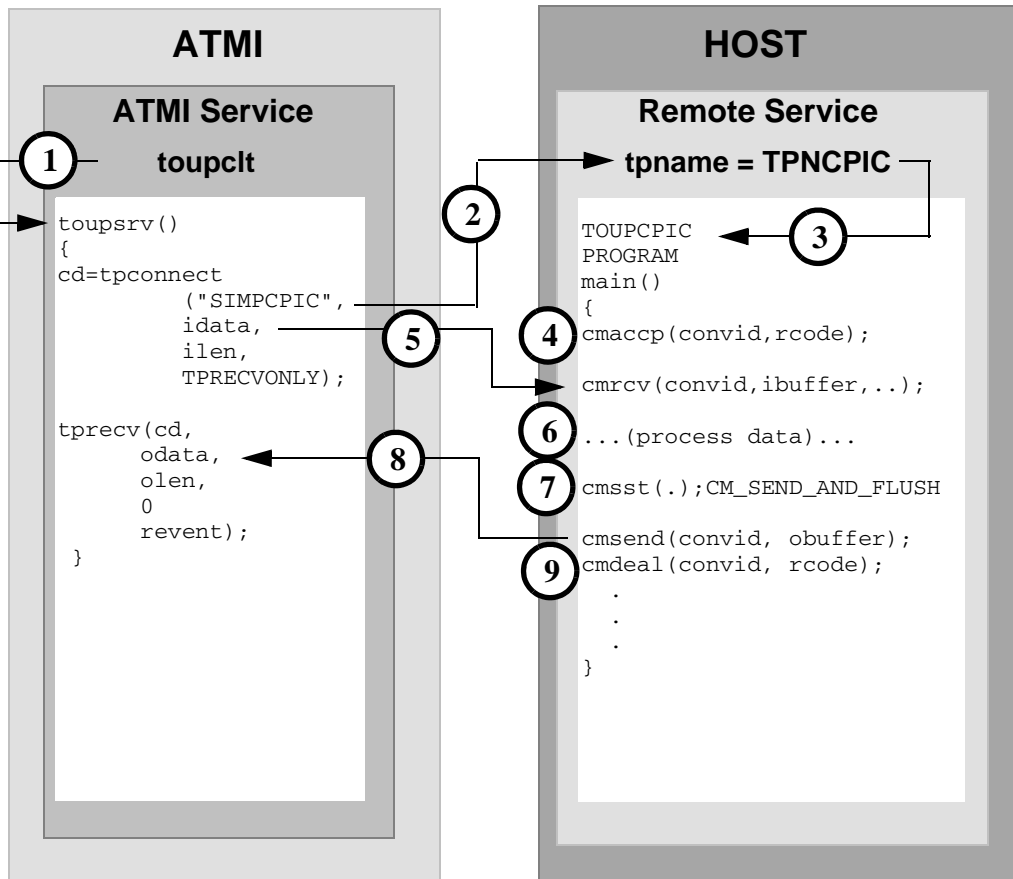
DM_REMOTE_SERVICES

SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=N
  
```

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpacall` with a `TPNOREPLY` request for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file.

3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.
6. The `TOUPCPIC` program processes data.

ATMI Conversational Client to Host CPI-C, Server Gets Control



DMCONFIG File Entry

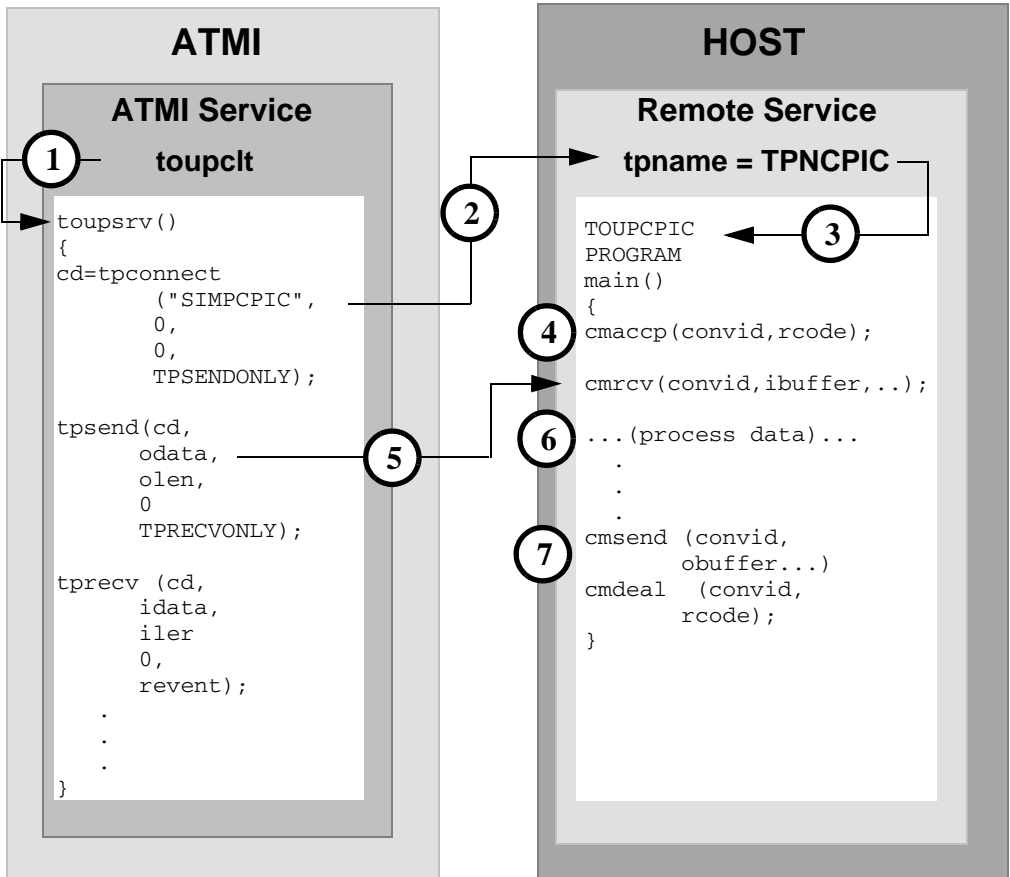
```
DM_REMOTE_SERVICES
```

```
SIMPCPIC RNAME=TPNPCIC FUNCTION=APPC CONV=Y
```

1. ATMI client invokes toupsrv service.

2. The `tpoupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb the `tpoupsrv` can issue is `tprecv`. Data is sent on the `tpconnect` in the `idata` buffer.
3. The remote service with `tpname TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `idata` buffer contents for processing.
6. The `TOUPCPIC` program processes data.
7. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_FLUSH`.
8. The `cmsend` command returns the `obuffer` contents into the client `tprecv odata` buffer. The data is immediately flushed on the send request.
9. The `cmdeal` request ends the conversation.

ATMI Conversational Client To Host CPI-C, Client Retains Control



DMCONFIG File Entry

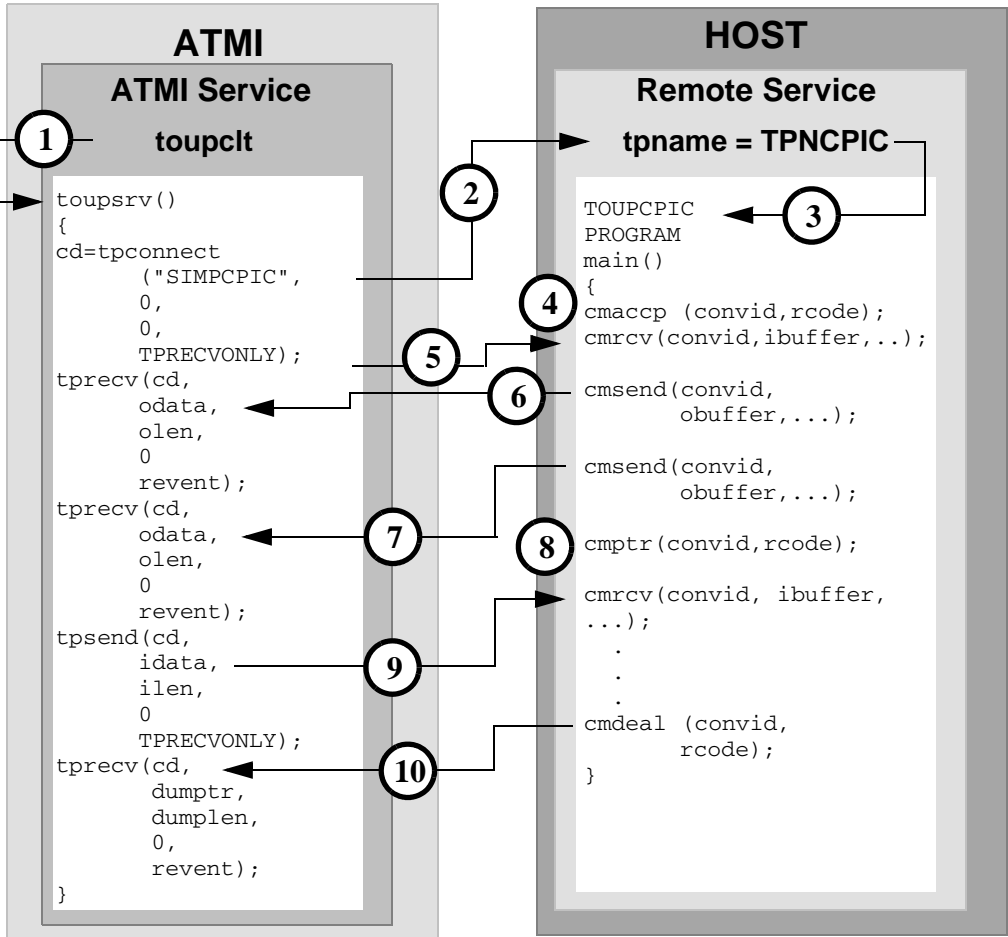
```

DM_REMOTE_SERVICES
SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=Y
  
```

1. ATMI client invokes toupshrvc service.

2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPSENDONLY` indicates the client retains control and continues to send data. No data is sent with the `tpconnect`.
3. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` call. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` request receives the `tpsend idata` buffer contents for processing. The conversation is relinquished with the `TPRECVONLY` flag.
6. The `TOUPCPIC` program processes data.
7. The `cmsend` returns a response in the `tprecv idata` buffer, along with notification from the `cmdeal` command that the conversation is over. The `cmdeal` flushes the data buffer and the `tprecv reevent` parameter is set to `TPEV_SUCCESS`.

ATMI Conversational Client to Host CPI-C, Client Grants/gets Control



DMCONFIG File Entry

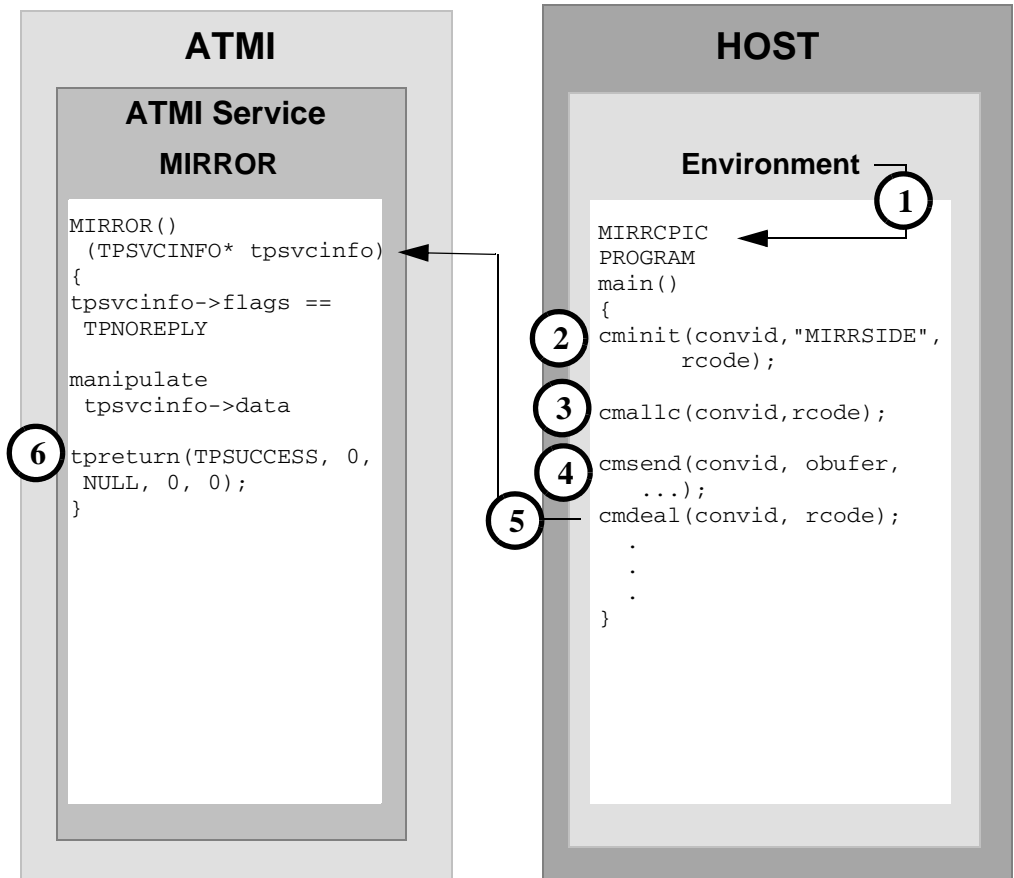
DM_REMOTE_SERVICES

SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=Y

1. ATMI client invokes `toupsrv` service.

2. The `toupsrv` service issues `tpconnect` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. The `TPRECVONLY` indicates the server gains control and the first conversation verb the `toupsrv` can issue is `tprecv`.
3. The remote service with `tpname` `TPNPCIC` invokes `TOUPCPIC` program.
4. The server accepts the conversation with the `cmaccp` request. The conversation id returned on the request in `convid` is used for all other requests on this conversation.
5. The `cmrcv` requests receives the indicator that control has been granted to the server.
6. The `cmsend` request returns its `obuffer` contents into the first client `tprecv odata` buffer. The data may not immediately be sent.
7. The `cmsend` request returns its `obuffer` contents into the second client `tprecv odata` buffer. The data may not immediately be sent.
8. The `cmptr` request flushes the data to the client, and grants control to the client.
9. The `cmrcv` request receives the `tpsend idata` buffer contents for processing. The `TPRECVONLY` is passed to the `tprecv`, relinquishing control of the conversation.
10. The `cmdeal` indicates a successful completion of the conversation to the `tprecv`; no data is passed.

Host CPI-C to ATMI Asynchronous Request/Response Server with No Reply



DMCONFIG File Entry

```

DM_LOCAL_SERVICES

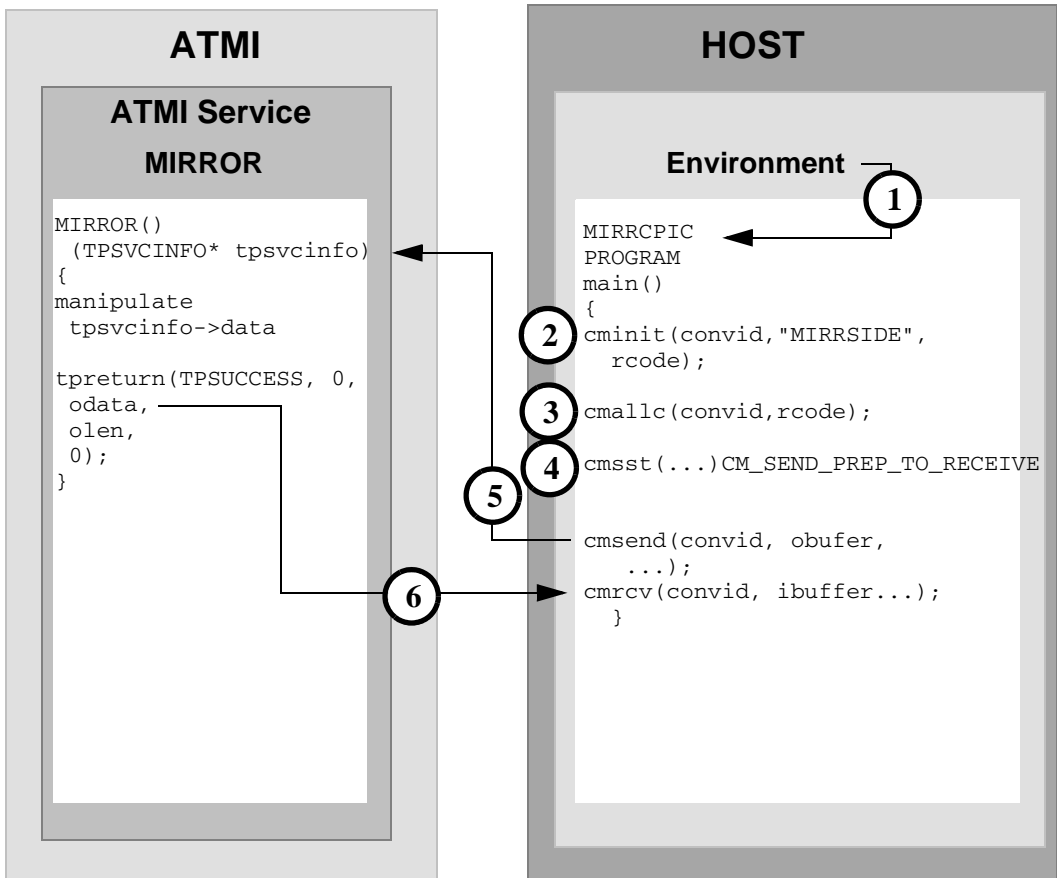
MIRROR RNAME=MIRRORSERV CONV=N

```

1. The CPI-C application program `MIRRCPIC` is invoked using environment start-up specifications.

2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cma11c` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsend` request sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer.
5. The `cmdeal` request flushes the data, and indicates the conversation is finished with the `TPNOREPLY` in the `tpsvcinfo->flag` field.
6. The service completes with the `tpreturn`.

Host CPI-C to ATMI Server Request/Response



DMCONFIG File Entry

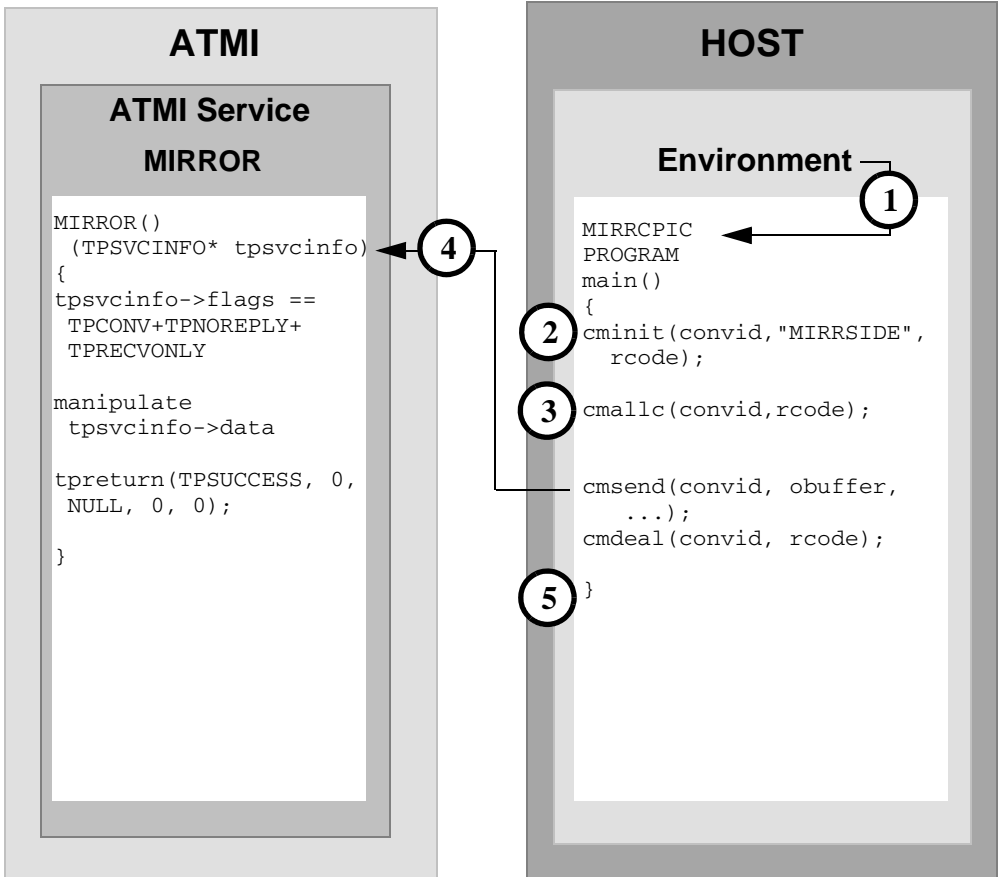
```
DM_LOCAL_SERVICES
```

```
MIRROR RNAME=MIRRORSERV CONV=N
```

1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.

2. The `MIRRPCIC` client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cma11c` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsst` request prepares the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE`.
5. The `cmsend` request immediately sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer and relinquishes control to the `mirror serv` service.
6. The `cmrcv` request receives the contents of the `odata` returned on the ATMI `tpreturn` service, and notification that the conversation has ended with the return code value of `CM_DEALLOCATED_NORMAL`.

Host CPI-C to ATMI Conversational Service, Client Retains Control



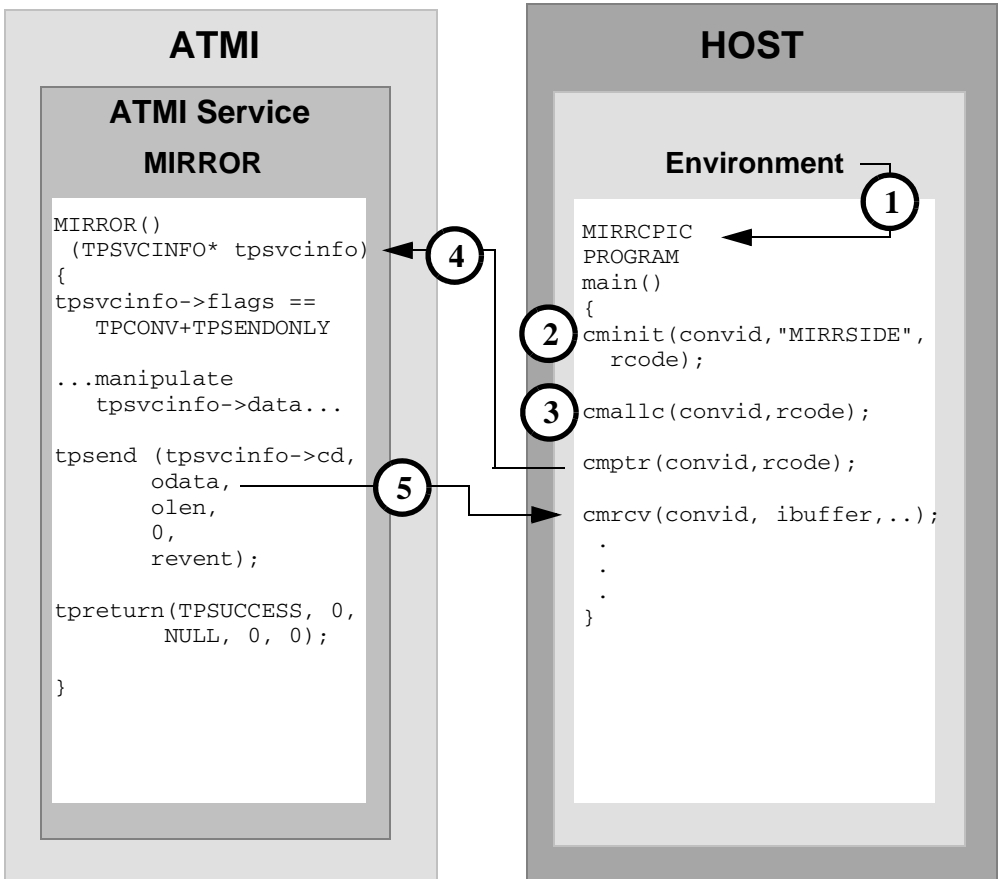
DMCONFIG File Entry

```
DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=N
```

1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.

2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation id that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cma11c` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmsend` request sends the contents of `obuffer` to the ATMI service in the `tpsvcinfo->data` buffer.
5. The `cmdeal` request flushes the data and ends the conversation, as indicated by `TPNOREPLY` in the `tpsvcinfo->flag` field.

Host CPI-C ATMI to Conversational Service, Client Grants Control



DMCONFIG File Entry

```

DM_LOCAL_SERVICES

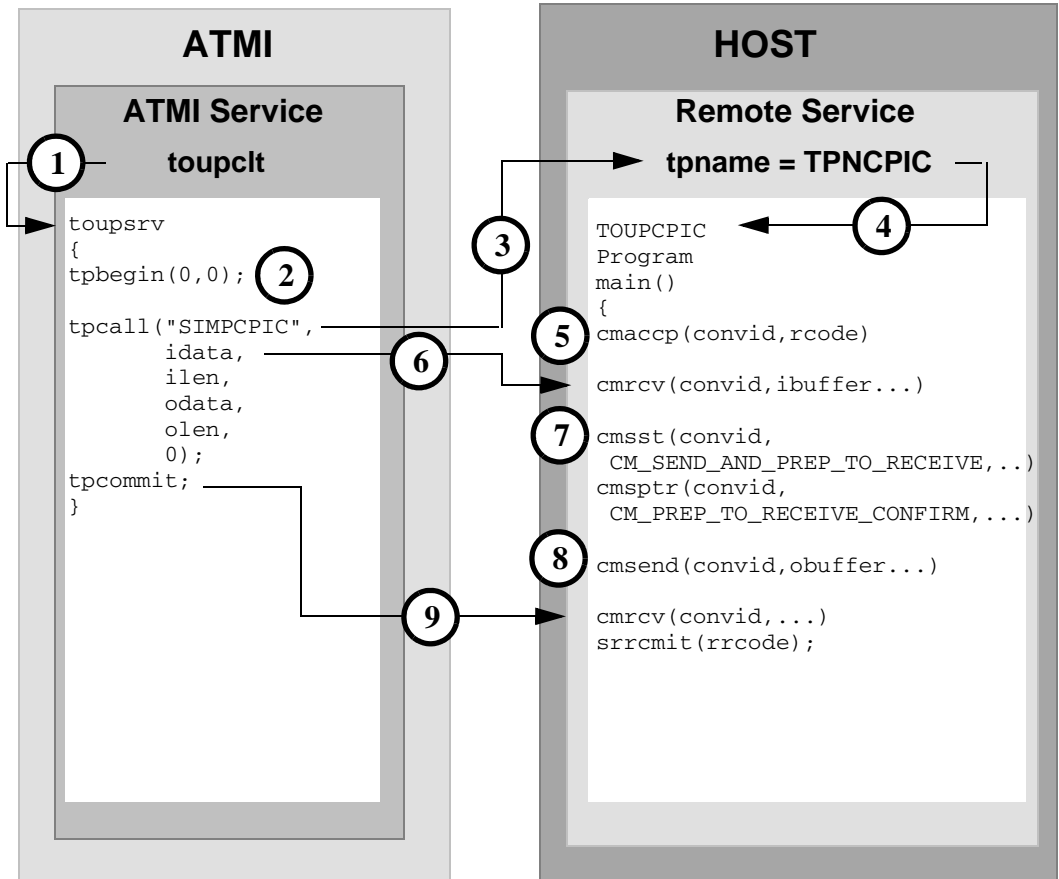
MIRROR RNAME=MIRRORSERV CONV=Y

```

1. The CPI-C application program MIRRCPIC is invoked using environment start-up specifications.

2. The `MIRRCPIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cma11c` request initiates the advertised service mapped to `MIRROR` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
4. The `cmptr` relinquishes control of the conversation to the ATMI service indicated as `TPSENDONLY` in the `tpsvcinfo->flag` field. No data is passed in the `tpsvcinfo->data` field.
5. The `cmrcv` receives the contents of the `tpsend odata` into the `ibuffer`. The end of the conversation is passed from the `tpreturn` service as return code value `CM_DEALLOCATED_NORMAL`.

Transactional ATMI Client Request/Response to Host CPI-C



DMCONFIG File Entry

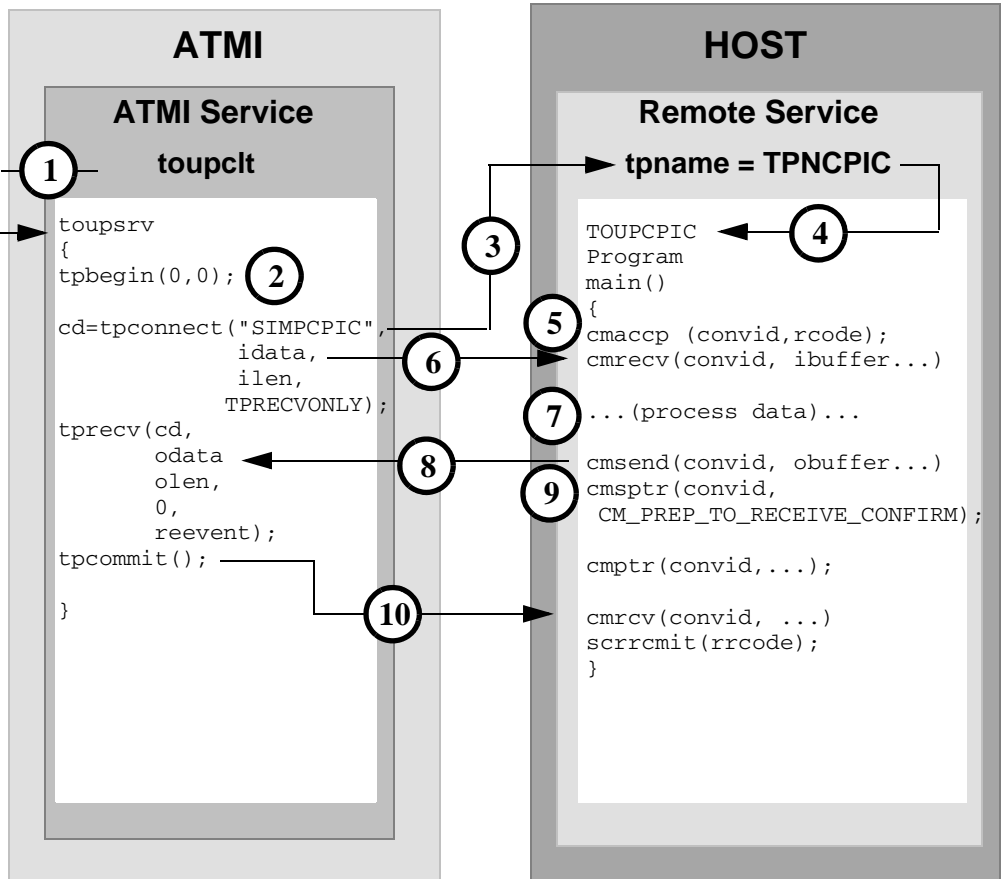
```
DM_REMOTE_SERVICES
```

```
SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=N
```

1. ATMI client invokes `toupsrv` service.
2. The `toupsrv` service issues `tpbegin` to start the transaction.

3. The `toupsrv` service issues `tpcall` for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. Data is sent from the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname` `TPNPCIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents for processing.
7. The `cmsst` and `cmsptr` prepare the next send request by setting the send type to `CM_SEND_AND_PREP_TO_RECEIVE` and by setting the prepare-to-receive type to `CM_PREP_TO_RECEIVE_CONFIRM`.
8. The `cmsend` request immediately returns the `obuffer` contents into the client's `odata` buffer. The server relinquishes control to the server and indicates the end of the conversation with the `CONFIRM` request.
9. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request, and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

Transactional ATMI Conversational Client to Host CPI-C, Server Gets Control



DMCONFIG File Entry

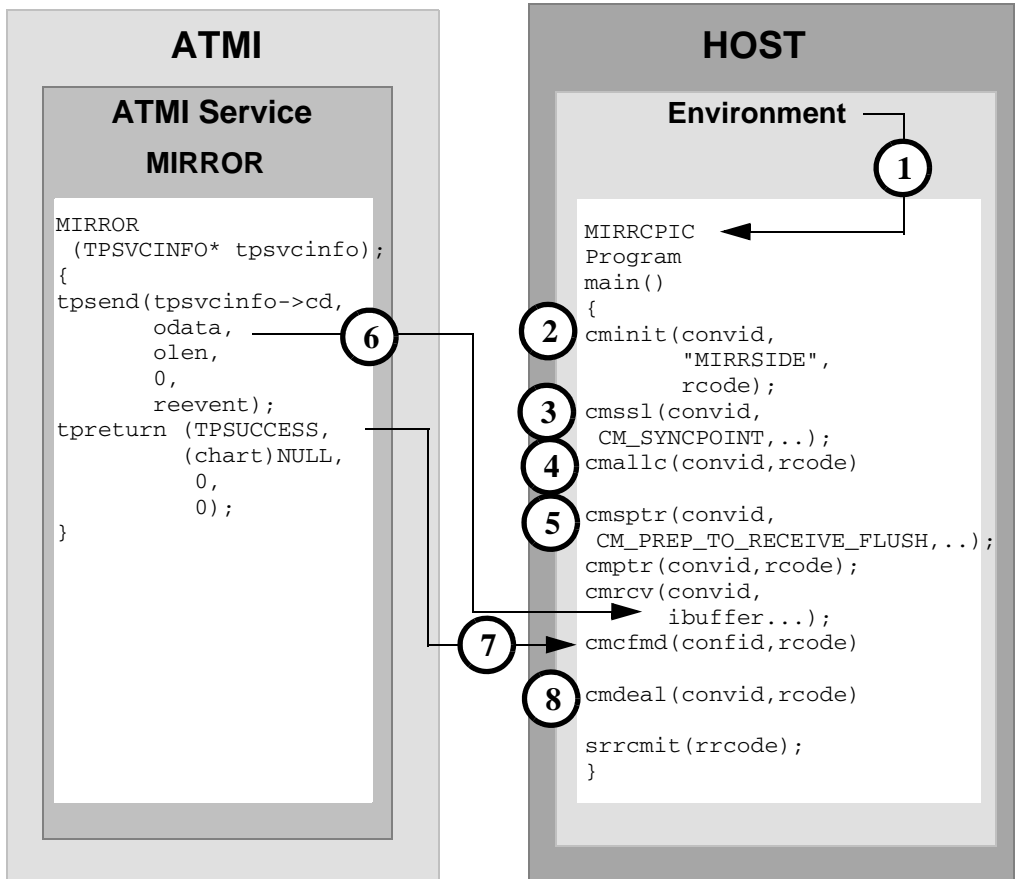
```
DM_REMOTE_SERVICES
```

```
SIMPCPIC RNAME=TPNCPIC FUNCTION=APPC CONV=Y
```

1. ATMI client invokes `touplesrv` service.
2. The `touplesrv` service issues `tpbegin` to start the transaction.

3. The `toupsrv` service issues a `tpconnect` service request for `SIMPCPIC`, which is advertised in the `DM_REMOTE_SERVICES` section of the `DMCONFIG` file. Data is sent in the `idata` buffer on the `tpconnect`.
4. The remote service with `tpname` `TPNCPIC` invokes `TOUPCPIC` program.
5. The server accepts the conversation with the `cmaccp` call. The conversation ID returned on the request in `convid` is used for all other requests during this conversation.
6. The `cmrcv` request receives the `idata` buffer contents sent on the `tpconnect` for processing.
7. The `TOUPCPIC` program processes the data.
8. The `cmsend` returns the `obuffer` contents into the client's `tprecv odata` buffer. The buffer contents may not be sent immediately.
9. The `cmsptr` prepares the prepare-to-receive request with `CM_PREP_TO_RECEIVE_CONFIRM`. The `cmptr` request with `CONFIRM` indicates that the conversation is finished and is communicated to the `tprecv` as `TPEV_SVCSUCC`.
10. The `toupsrv` issues the `tpcommit` to successfully complete the transaction and commit all updated resources. The `cmrcv` request receives the commit request and responds explicitly to the request with the SAA Resource/Recovery commit call `srrcmit`. The conversation is ended after the successful commit exchange.

Transactional Host CPI-C to ATMI Conversational Server, Client Grants Control



DMCONFIG File Entry

```

DM_LOCAL_SERVICES
MIRROR RNAME=MIRRORSERV CONV=Y

```

1. The CPI-C application program MIRRPCIC is invoked using environment start-up specifications.

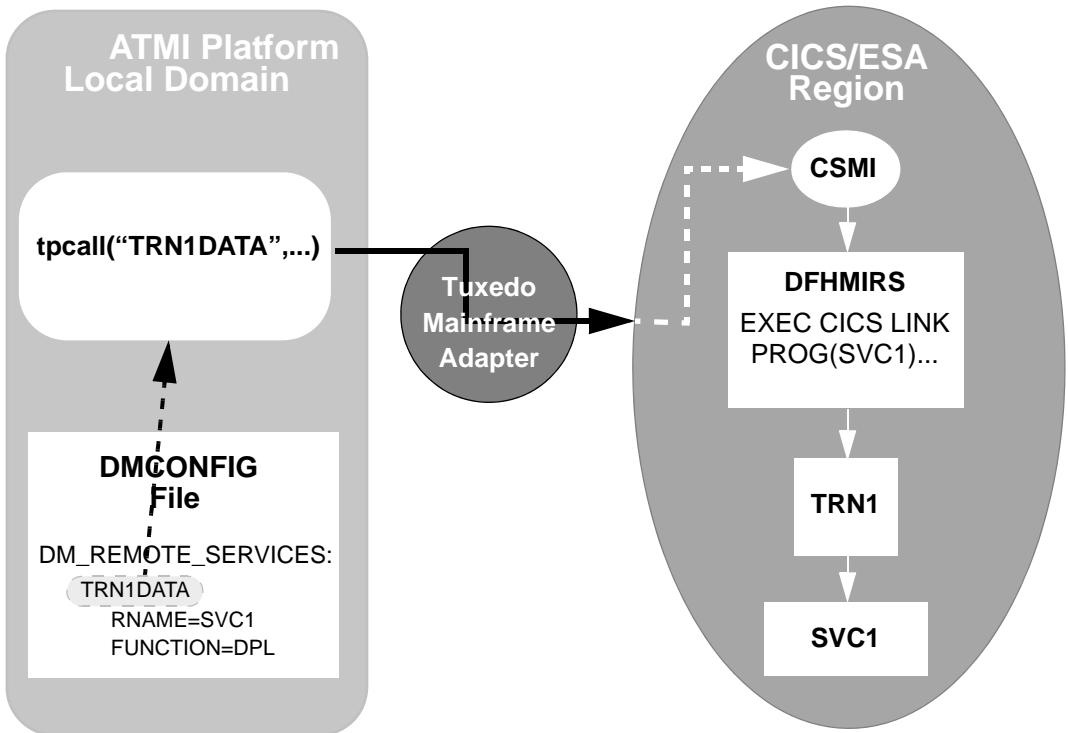
2. The `MIRRPCIC` client requests `cminit` to establish conversation attributes and receive a conversation ID that will be used on all other requests on this conversation. The remote server and service are named in the CPI-C side information entry `MIRRSIDE`.
3. The `cmssl` sets the conversation attributes to sync-level 2 with `CM_SYNCPOINT`. This allows the ATMI service to participate in the transaction.
4. The `cmalloc` request initiates the advertised service mapped to `MIRRORSERV` in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
5. The `MIRRPCIC` causes the client to relinquish control to the ATMI server with a prepare-to-receive request. The `cmsptr` sets the prepare-to-receive type to `CM_RECEIVE_AND_FLUSH`. The `cmptr` request immediately relinquishes control.
6. The `MIRROR` service sends the data contents of the `odata` buffer to the `cmrcv ibuffer`. The `cmrcv` receives a confirm request from the server indicating the conversation should be terminated.
7. The client replies positively to the confirm request with `cmcfmd`.
8. The `MIRRPCIC` client prepares to free the conversation with the `cmdeal` request. The conversation in `CM_DEALLOCATE_SYNC_LEVEL` commits all updated resources in the transaction and waits for the SAA resource recovery verb, `srrcmit`. After the `commit` sequence has completed, the conversation terminates.

CICS/ESA Mirror Transaction Examples

- [Implicit Attachment of TRANSID \(Outbound Requests Only\)](#)
- [Explicit Attachment of TRANSID for Outbound Requests](#)
- [Explicit Attachment of TRANSID for Inbound Requests](#)

Implicit Attachment of TRANSID (Outbound Requests Only)

Figure 2-1 Implicit Attachment of TRANSID (Outbound Requests Only)



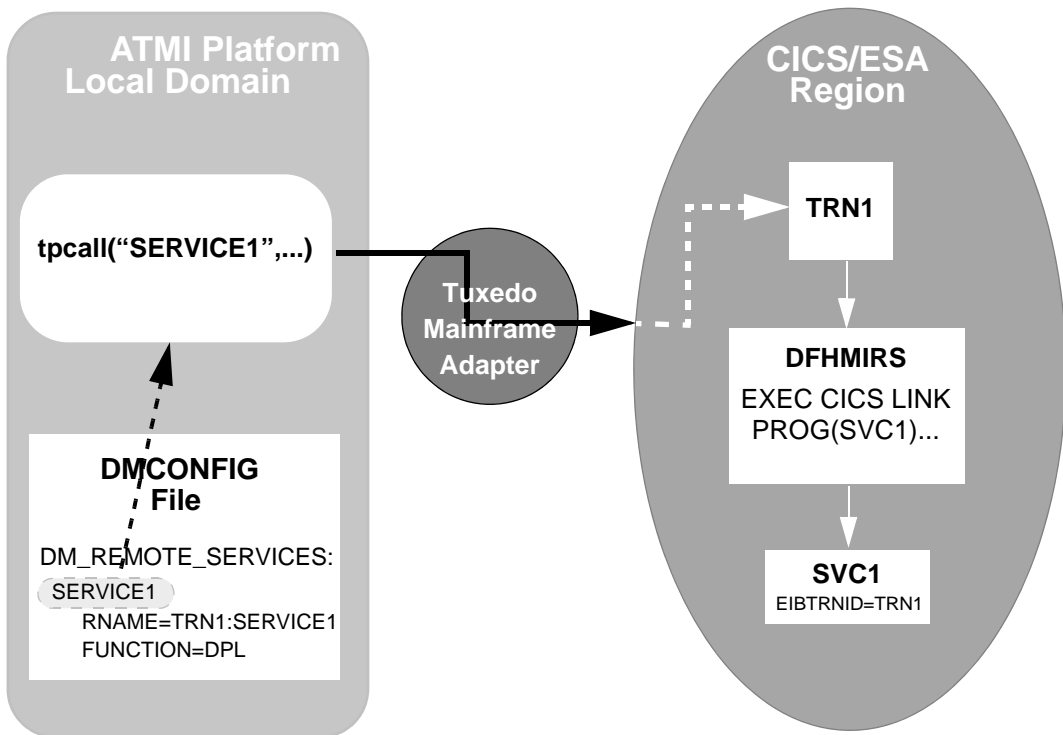
The following list describes the process for implicit attachment as illustrated in [Figure 2-1](#):

1. The ATMI service makes a request to the service `TRN1DATA`, which is advertised as a remote service in the `DMCONFIG` file. It is a DPL request to a program named `SVC1` in the CICS/ESA region.
2. The first four characters of the remote service tag name (`TRN1`) are extracted and passed to the CICS/ESA region as the invoking `TRANSID`. No CICS/ESA resource definition for the `TRANSID` is required in the region.
3. The mirror transaction `CSMI` is attached in the CICS/ESA region, starting the mirror program `DFHMIRS`. The program performs the DTP requests for the service.

4. The mirror program now attaches the invoking `TRANSID` (`TRN1`) and then invokes the application service program `SVC1`. The program can interrogate the `EIBTRNID` field to find this value.

Explicit Attachment of TRANSID for Outbound Requests

Figure 2-2 Explicit Attachment of TRANSID for Outbound Requests



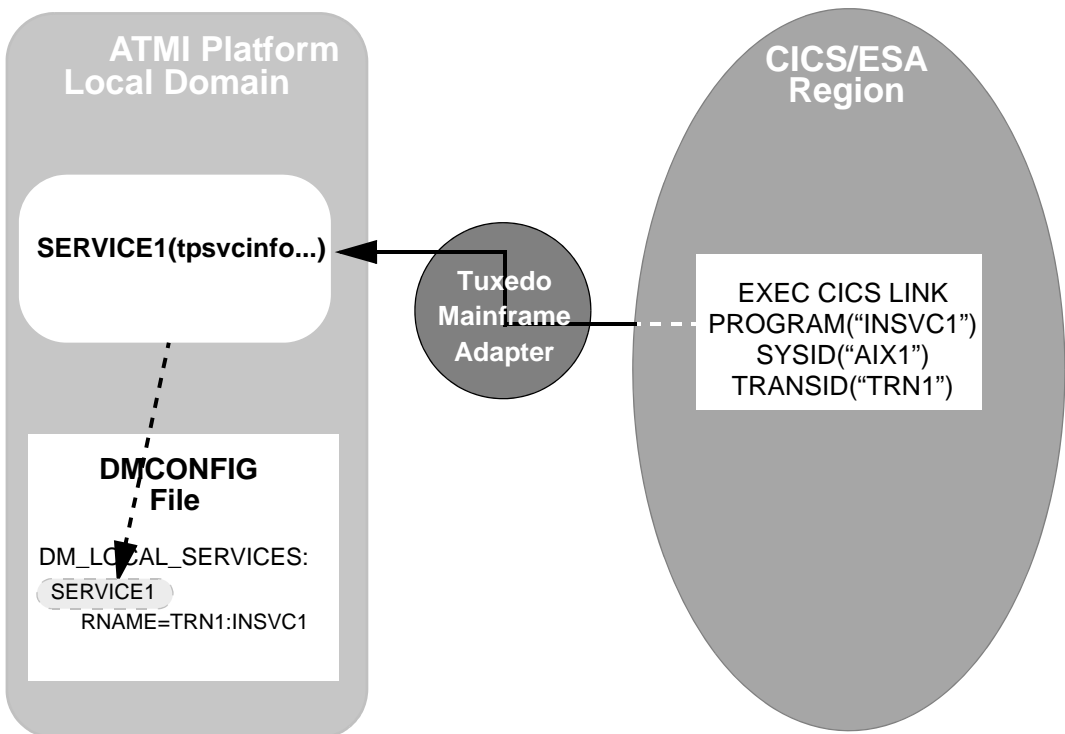
The following list describes the process for explicit attachment as illustrated in [Figure 2-2](#):

1. The ATMI program makes a service request for `SERVICE1`, which is advertised as a remote service in the `DMCONFIG` file. The `FUNCTION` option indicates the remote service is invoked as a DPL.
2. The request extracts `TRN1` as an alternate mirror transaction ID for the remote region, along with the remote program name `SERVICE1`.

3. The `TRN1` ID is attached instead of the default mirror transaction, `CSMI` or `CVMI`. The `TRN1` ID must be defined as a transaction resource in the remote region and must point to the mirror transaction program `DFHMIRS`.
4. The mirror program `DFMMIRS` calls the server application program, passing the `TRN1` ID in the `EIBTRNID` field.

Explicit Attachment of TRANSID for Inbound Requests

Figure 2-3 Explicit Attachment of TRANSID for Inbound Requests



The following list describes the process for implicit attachment as illustrated in [Figure 2-3](#):

1. The CICS/ESA program makes a request to `INSVC1`, which is a local ATMI service. The `SYSID` and `PROGRAM` values in the request identify the local system and the name of the local service. The `TRANSID` option indicates the mirror transaction to be initiated.

2. The `PROGRAM` and mirror `TRANSID` are extracted from the DPL request and are used to find an exact `RNAME` match in the `DM_LOCAL_SERVICES` section of the `DMCONFIG` file.
3. The service `SERVICE1`, which is advertised locally in the ATMI platform application, is initiated.

Additional Information

Additional information about CICS/ESA Intersystem Communications may be found in the following IBM publications:

- *CICS/ESA Intercommunication Guide*, IBM publication No. SC33-0657
- *CICS/ESA Distributed Transaction Programming Guide*, IBM publication No. SC33-00783
- *CICS/ESA Recovery and Restart Guide*, IBM publication No. SC33-0658