MQSeries® Integrator Agent for CICS® Transaction Server

Tutorial and Techniques

Version 1 Release 1 Modification 1

SC34-6087-00

MQSeries® Integrator Agent for CICS® Transaction Server

Tutorial and Techniques

Version 1 Release 1 Modification 1

SC34-6087-00

Note: Before using this information and the product it supports, read the information in "Notices" on page 263.

First edition March 29th, 2002

This edition applies to version 1 release 1 modification 1 of MQSeries Integrator Agent for CICS Transaction Server (product number 5655-F25) and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can make comments on this information via e-mail at idrcf@hursley.ibm.com.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About MQSI Agent for CICS

Use of the MQSeries Integrator Agent for CICS Transaction Server (also know here as MQSI Agent for CICS) can be separated into two phases: a build time phase and a run time phase.

Build time is defined generically as the development period when a process or object is defined, modeled or modified electronically. The MQSI Agent for CICS component that is used at build time is called the *Adapter Builder*.

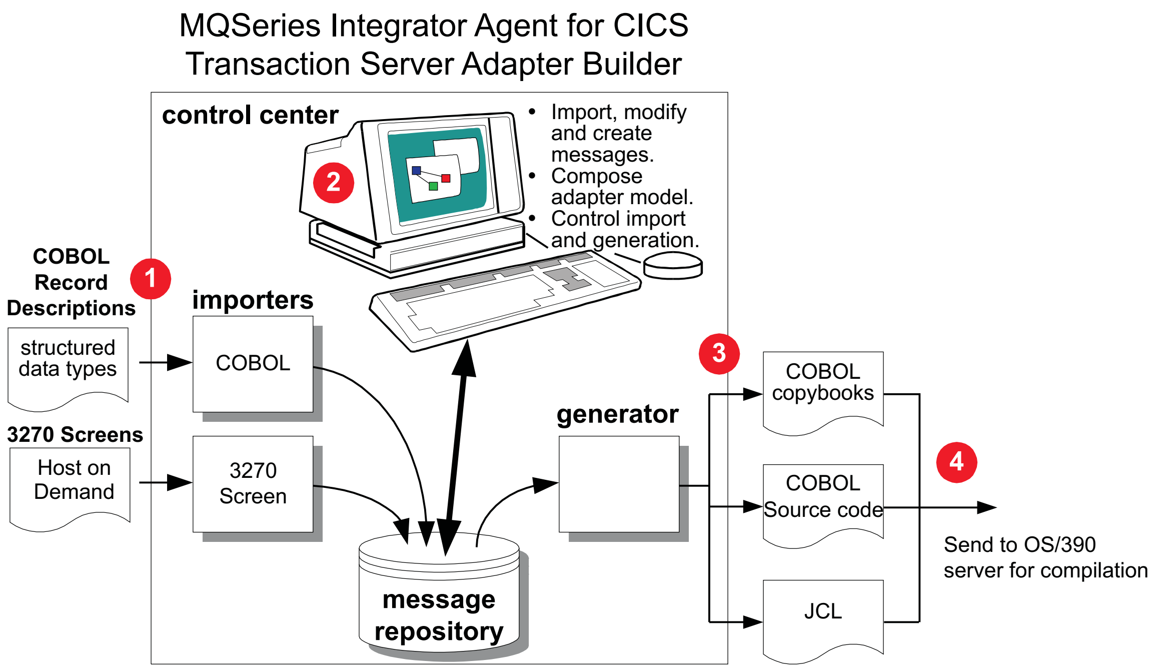

MQSeries Integrator Agent for CICS Transaction Server Adapter Builder
control center
2
Import, modify and create messages.
Compose adapter model.
Control import and generation.
COBOL Record Descriptions
1
importers
structured data types
COBOL
3270 Screens
Host on Demand
3270 Screen
3
generator
COBOL copybooks
COBOL Source code
JCL
4
Send to OS/390 server for compilation
message repository

Figure 1. Adapter Builder Components

The Adapter Builder provides a graphical environment for modeling *adapters*. Its intuitive visual interface enables users to model adapters to be used for host-based transaction processing. In the MQSI Agent for CICS paradigm, an adapter is the output of the Adapter Builder. Adapters are modeled, defined and generated using the Adapter Builder. As output from the builder, an adapter consists of COBOL (source and copybooks) and JCL.

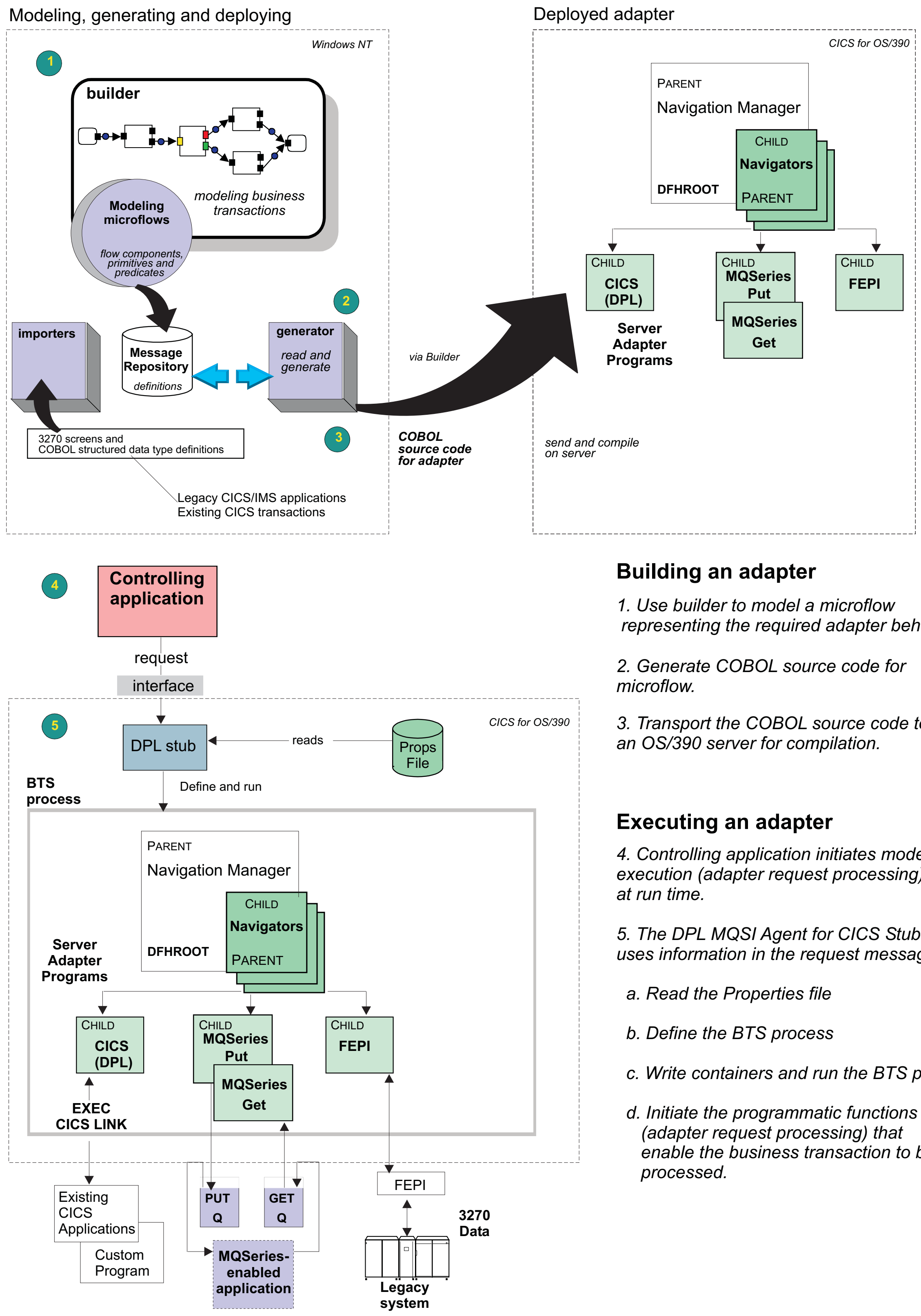
After a user creates an adapter, it is moved from the builder in the Windows/NT environment, to an OS/390 server, where it is compiled as a BTS application. A BTS application is a CICS application that uses the CICS business transaction services API.

Run time is defined generically as the time period in which the process or object created at build time becomes operational. In the MQSeries Integrator Agent for CICS Transaction Server paradigm, run time is when the adapter is invoked by a controlling application, and as a result, performs the business transaction processing that was modeled at build time.

About MQSI Agent for CICS

There is no user interface for the run time component. However, that is not to say that user’s do not interact with MQSeries Integrator Agent for CICS Transaction Server at run time. In order for the adapters to execute at run time, persons familiar with CICS and BTS will need to prepare the run time environment, by defining required resources. Run time users may also need to investigate errors that may occur during run time processing.

See Figure 2 on page xiii for an illustration of the sequence of steps that a user would go through to build, deploy and run an adapter using MQSI Agent for CICS.



Building an adapter

1. Use builder to model a microflow representing the required adapter behavior.

2. Generate COBOL source code for microflow.

3. Transport the COBOL source code to an OS/390 server for compilation.

Executing an adapter

4. Controlling application initiates model execution (adapter request processing) at run time.

5. The DPL MQSI Agent for CICS Stub program uses information in the request message to:

a. Read the Properties file

b. Define the BTS process

c. Write containers and run the BTS process

d. Initiate the programmatic functions (adapter request processing) that enable the business transaction to be processed.

Figure 2. Process for using MQSI Agent for CICS

The objectives of this tutorial

The objective of this tutorial is to instruct users on the logic and steps of using MQSeries Integrator Agent for CICS Transaction Server.

By modeling a business transaction and by generating and deploying the modeled adapter to an OS/390 server, you will gain an understanding of how to use this product in your environment to meet your business needs.

By following the information and instructions in the tutorials you should be able to:

- Understand the general guidelines for using the MQSI Agent for CICS Adapter Builder.
- Import application definitions with 3270 screens and COBOL structured data type definitions.

  The MQSI Agent for CICS Adapter Builder contains two *importers* that enable you to import an application's interface into the Adapter Builder in the form of messages and associated components. These components become the building blocks used in adapter modeling.
- Create workspaces to define adapter flow logic for the three types of adapters that run on the OS/390 server.

  MQSI Agent for CICS provides three specialized adapter types that can be used in microflow modeling:
  - FEPI Adapter - A composed component that describes the rules for sequencing a 3270 screen dialog. It models screen navigation and corresponds to the FEPI server adapter functionality in the server run time.
  - DPL Adapter - A composed component that describes a micro-controlflow with a CICS transaction via a Distributed Program Link. It corresponds to the DPL server adapter functionality in the server run time.
  - MQSeries Adapter - A composed component that describes a micro-controlflow with an MQSeries enabled application. It corresponds to the MQSeries server adapter functionality in the server run time.
- Create and generate the COBOL source code. This source code contains the adapter flow logic as well as static server run time information.
- Understand the different mechanisms for deploying the adapter to the OS/390 server.
- Understand the run time processing involved in executing each of the three adapters that you created and generated.
- Validate and test the adapters that you create in the tutorial.

See Chapter 2, “Tutorial overview” on page 9 for general information on the adapters that you will build from this tutorial and for information on the exercises that you will perform in this tutorial.

Who should use this tutorial

The information in this tutorial is primarily intended for people who want to become familiar with the builder component of MQSeries Integrator Agent for CICS Transaction Server. However, to gain the full benefits of the tutorial, which includes deploying the adapters to CICS and using the Simulator program to validate the adapters, the individual who participate in these tutorials should work with their site's CICS administrator and OS/390 specialist to make sure that the adapters can be deployed, defined and tested in a run time environment.

A CICS systems personnel and or OS/390 specialists that has access to CICS regions at your site will need to define adapter resources to CICS and to customize the build time templates.

An example of how adapter resources could be defined to CICS is included in “Example procedure for defining adapter resources to CICS” on page 239. **The information in this appendix should only be used as a reference**, as CICS environments and the procedures used to define resources to CICS will vary from site to site.

Related information

If you have not had any exposure to the Adapter Builder component of the MQSeries Integrator Agent for CICS Transaction Server product, you should read the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center*. This book contains information on the concepts of the MQSeries Integrator Agent for CICS Adapter Builder.

For reference information on the run time components, you should read the *MQSeries Integrator Agent for CICS Transaction Server Run Time User's Guide*.

See the MQSeries Integrator Agent for CICS website at http://www.ibm.com/software/ts/cics/mqiac/ for information on the MQSI Agent for CICS product, including a fact sheet, product overview and the latest SupportPac information.

You can access these documents from the Start menu:

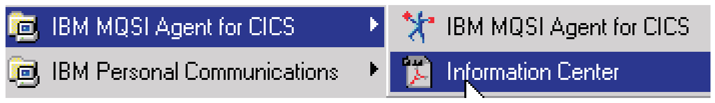


Figure 3. Accessing the MQSI Agent for CICS documentation

About MQSI Agent for CICS

Chapter 1. Guidelines for building adapters

This chapter describes the guidelines for building adapters using the MQSI Agent for CICS Adapter Builder. It contains information about the general steps that should be undertaken to develop and build a CICS Adapter that integrates a server application.

Detailed information and keystroke-by-keystroke instructions for most of these steps can be found in tutorials in other chapters in this document.

Requirements analysis and design considerations

Before you begin to use the MQSI Agent for CICS Adapter Builder you should spend some time analyzing the business objectives that the adapter will address and then spend some time considering how you will design the adapter.

Requirements analysis

It is essential that you understand your objectives and the environment in which you will be working. Therefore, before you begin to model an adapter using the MQSI Agent for CICS Adapter Builder, you should spend some time analyzing the requirements of the business transaction(s) that you will model. Requirements analysis involves performing the following tasks:

- Obtaining an understanding of the business objective of the transaction
- Knowing the programming, environmental and system resources to be used by the adapter

An understanding of the requirements and of the business objective will facilitate your design efforts and will make more efficient use of the adapter builder.

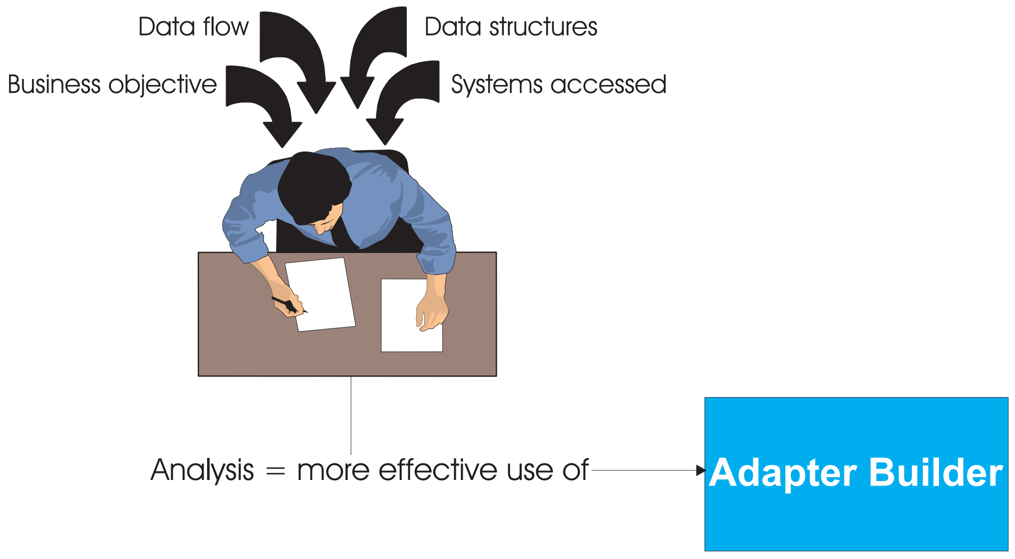


Figure 4. Analyze information before you begin modeling

Here are some points to consider in requirements analysis:

- What business transaction do you need to model and what resources will be required?

Guidelines for building adapters

- Identify back-end host information (system, method - DPL, MQ or FEPI, data structures).
- Where are you getting data from (source) and where is the data going to (destination)

Remember, the time that you spend in the requirements analysis phase will prove invaluable later.

Design considerations

After you have a solid understanding of the adapter requirements, you enter the design phase. You do not design an adapter using MQSI Agent for CICS Adapter Builder. Designing an adapter takes place after requirements analysis and prior to using the MQSI Agent for CICS Adapter Builder to build the adapter.

In your design phase:

- Determine the number of flows required to fulfill the transaction
- Determine how data moves through your run time environment, the flow of the adapter
- Determine if there is any commonality that you may wish to reuse in other adapters
- Determine data structures that are used as input and output interfaces for application data.
- Determine or identify naming conventions, including host back-end, host user, and client.
- Identify items you wish to make decisions on (for example, add, change or delete actions).
- Identify COBOL copybooks you will be importing.

  Note: The copybook generate removes underscores from the file namesF and only uses the first eight characters of the filename to generate the new copybook name. Therefore, you may need to rename some file structures where naming conflicts can arise. See "Accessing a completed workspace" on page 12.

Application interface

The server applications that are to be integrated using MQSI Agent for CICS need to be analyzed to determine their available interfaces. MQSI Agent for CICS supports three different interface methods. The three interface methods are:

- CICS Distributed Program Link (DPL)
- MQSeries messaging (MQ)
- 3270 datastreams using Front End Programming Interface (FEPI)

See Figure 5 on page 3 for an illustration of the interface methods supported by MQSI Agent for CICS.

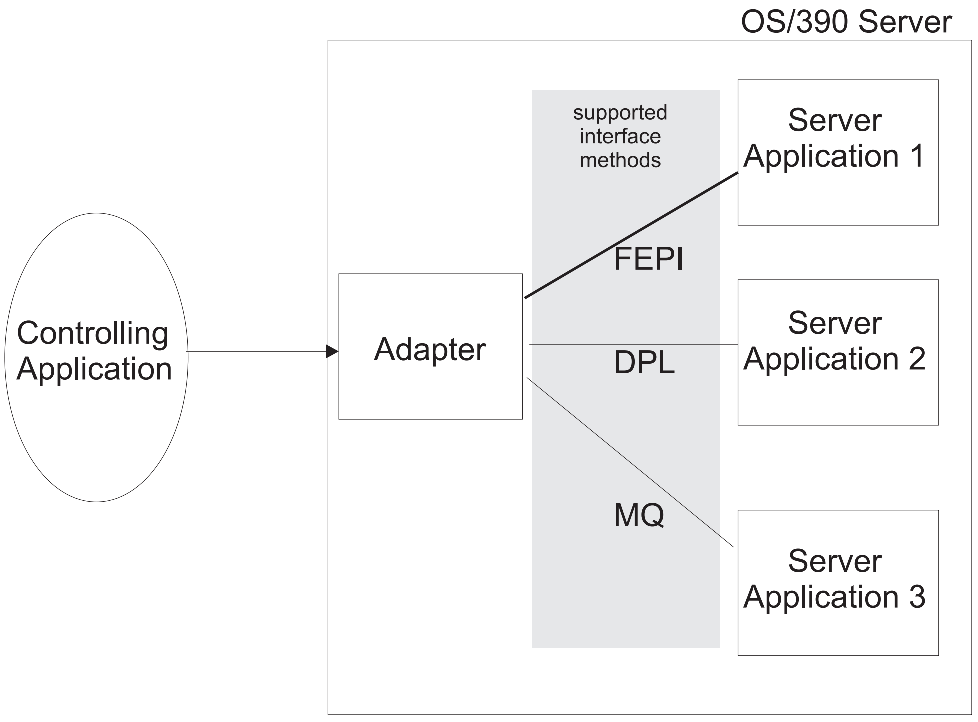


Figure 5. Supported interface methods

Multiple applications using differing interface methods can be incorporated in one model. Consideration can be given to adding one of these interfaces to server applications that currently do not support any of the three. However, subflows are allowed only in the FEPI interface.

Run time environment variables

Before you begin to model an adapter, you need to find out some information about your run time environment. This run time environment information is used in the Integration Specification file (.ispec) and/or Connector Resource file (.rsc) to provide mappings to the run time properties file (DFHMAMPF) and other generated adapter programs.

The following text describes the required information for each adapter type and indicates, in parentheses, the associated specification file symbolic. For detailed discussion on how to use specification files, see the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* manual.

Specification file information for the DPL interface: For applications that are accessed via DPL, determine the following information:

1. Name of the application program that is ″linked to″ (MAT_LINKNAME)
2. If required, the Transaction ID under which ″linked to″ program will execute (MAT_LINKTRAN)
3. The CICS region where the ″linked to″ program is defined (MAT_SYSID)
4. Maximum length that can be sent or returned in the link (generally, the commarea length) (MAT_MAXCALEN)
5. Whether resources should be committed (SyncOnReturn) when the DPL Link is completed (MAT_SYNCONRETURN)

Specification file information for the MQ interface: For applications that are accessed via MQ, determine the following information:

1. Name of the queue that the server application monitors (MAT_REQUEST_QNAME)
2. Name of the queue where the server application should put the reply (MAT_REPLY_QUEUE)
3. Name of the MQManager that owns the queue for the reply (MAT_REPLY_QMGR)
4. Maximum length of the reply that can be sent to or returned by the server application (MAT_MAXOUTMSGLEN)
5. The amount of time in seconds the reply MQManager should wait for a reply (MAT_WAIT_INTERVAL)
6. The MQ message Type (request, reply, or datagram) (MAT_MQMSGTYPE)

Specification file information for the FEPI interface: For applications that are accessed via FEPI, determine the following information:

1. Name of the defined FEPI Pool from which the terminal session is to be allocated (MAT_POOL)
2. Name of the defined FEPI Target that identifies the region where the transaction will be run (MAT_TARGET)
3. The length of time, in seconds, to wait for a response screen after a screen has been sent (MAT_TIMEOUT)
4. Whether a User ID and PassTicket will be generated for Logon by the server (MAT_USELUPASS)

Determining the critical data structures in the server application

As part of your preparation for creating an adapter, you will need to analyze the server applications to determine their *critical data structures*. Critical data structures provide the input and output interfaces, and intermediate holding areas for important application data. In COBOL applications, these structures are often ″01 Levels″ in the program. Sometimes, however, they are lower level or subordinate items. For applications that are accessed via FEPI, all screens should also be considered critical data structures. Screens are handled via a special 3270 Importer described later in this section.

Once you have identified the critical data structures (other than screens), you can ″import″ them into the builder using the COBOL Importer. The COBOL Importer can only access code that resides on a workstation, so you will need to ftp the critical data structures from the host to a workstation directory. Often the original, unchanged COBOL source code and/or copybooks can be used as input to the COBOL Importer. In some cases, the original code may need to be edited slightly to comply with COBOL Importer requirements. While importing may require some editing of existing COBOL code, it is nearly always less work than the alternative method of manually entering data structures.

After it is imported, a data structure exists in the Control Center as a message. The message name is the same as the 01 Level that was imported with the exception that hyphens (-) have been replaced with underscores (_). Whether imported or entered manually, all messages used in an adapter must be generated to a COBOL copybook before it can be deployed to the runtime. The generated copybook will look slightly different than the original import but will be syntactically equivalent.

An import operation creates a name for the copybook by taking the first eight letters or numbers (hyphens are discarded) of the 01 Level and adding a ″.cpy″ suffix.

For example, the following data structure,

```
01   EMP-DATA-RECORD.
          05  EMP-DATA-OF-BIRTH         PIC 9(8)  VALUE  ZERO.
 etc.
```

creates a message name of EMP_DATA_RECORD and a copybook name EMPDATAR.cpy. The need for more desirable message and copybook names is another reason that you might want to edit the original COBOL code prior to importing a data structure.

CAUTION:
Verify that the file name of the 8–character copybook is unique before you begin to work on your adapter.

For all applications, one request data structure and potentially multiple reply data structures are generally identified as critical data structures. The request data structure is similar to the ″Start Data″ that is passed to the application when a CICS transaction is started. It often contains file key information, such as account number or customer name. Reply data structures contain the data retrieved by the server application. Flow routing and decision making in the builder support the use of multiple reply data formats. If the server application uses a separate data structure to hold or return error information, then that structure should be considered critical as well.

Intermediate data areas should also be considered for import in the builder. For instance, areas that temporarily hold accessed data or are used to ″build up″ data accumulated via multiple operations are likely to be critical data structures. If it can not be determined whether a data structure is needed or not, it is best to delay its import until the need is established.

For DPL applications, the data structure(s) that define the Commarea of the DPL ″linked to″ program is a critical data structure. Often different data structures are used to map the ″passed″ data and the ″returned″ data to and from the link. At the time of import, the data structure that maps the ″passed″ data should be designated as a ″request″ message by selecting the appropriate radio button on the import dialog. The data structure that maps the ″returned″ data should be designated as a ″response″ message.

Sometimes the ″returned″ data can be moved to alternative data structures depending on a field in the ″returned″ data. These alternative data structures can be designated as ″undefined″ at import time. All data structures that are required to map the data should be deemed critical data structures.

For MQ applications, all data structures that map the user data portion of the MQ messages should be considered critical. Similar to DPL applications, data structures that map ″PUT″ data should be imported as ″request″ messages. Structures that map ″GET″ data should be imported as ″response″ messages.

For FEPI applications, screens constitute the critical data structures. Both screens that contain data to be ″scraped″ and all preliminary screens in the dialog should be imported using the 3270 Importer in the Control Center. In addition to the application screens, any screens that accomplish system functions such as Logon should also be imported.

Similar to data structures imported with the COBOL Importer, screens imported with the 3270 Importer exist in the builder as messages.

Building adapters

The general procedure for developing an adapter is as follows:

1. If you have not done so during the run time installation, customize the build time JCL templates to reflect your local OS/390 server environment:
   - DFHMAXCJ (Compile JCL)
   - DFHMAXPU (DFHMAMPF properties file update)
   - DFXMAX04 (OS/390 server account, IP address and DSN qualifier and deploy information).

   See the *MQSeries Integrator Agent for CICS Transaction Server Run Time User's Guide* for considerations on customizing the build time templates.
2. Setup the required Interaction Specification and Connector Resource specification files.

   See the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* manual for a discussion and examples of the use of the Specification files.
3. Import or create messages (data definitions).

   Messages can be imported via the:
   - 3270 Screen Importer
   - COBOL Importer

   To create or modify messages manually:
   - Create a new message set
   - Create elements
   - Create types
   - Create messages
   - Build transactions
4. Compose a microflow.
5. Perform data mapping (data transformations).
6. Associate the microflow with an adapter
7. Generate adapter code (COBOL source code, COBOL copybooks, and JCL).
8. Move adapter source files to an OS/390 system and compile and test. See "Deploying adapters". If you are satisfied with the adapter that you built, put it into production.

Deploying adapters

After you generate an adapter and associated copybooks, you must deploy the adapter code to test it before you put it into production. When the generated adapter is deployed to the OS/390 server, the adapter code is compiled. You should check that the adapter code compiled successfully.

You will need to define resources to CICS each time a new adapter is deployed. CICS needs to know which resources to use, what their properties are and how they interact with other resources.

To define resources to CICS you must:

- Submit JCL to run the Properties File Update job.

  Note: This is can be done automatically when you generate the adapter from the MQSI Agent for CICS Adapter Builder, or you can submit the JCL (DFHMAMPU) manually. DFHMAMPU run the Properties File Update job (DFHMAMUP)
- Run the CEDA transaction to define the programs, transaction IDs and files used by your adapter to CICS.

Figure 6 depicts how the adapter and associated files are deployed into the run time environment.

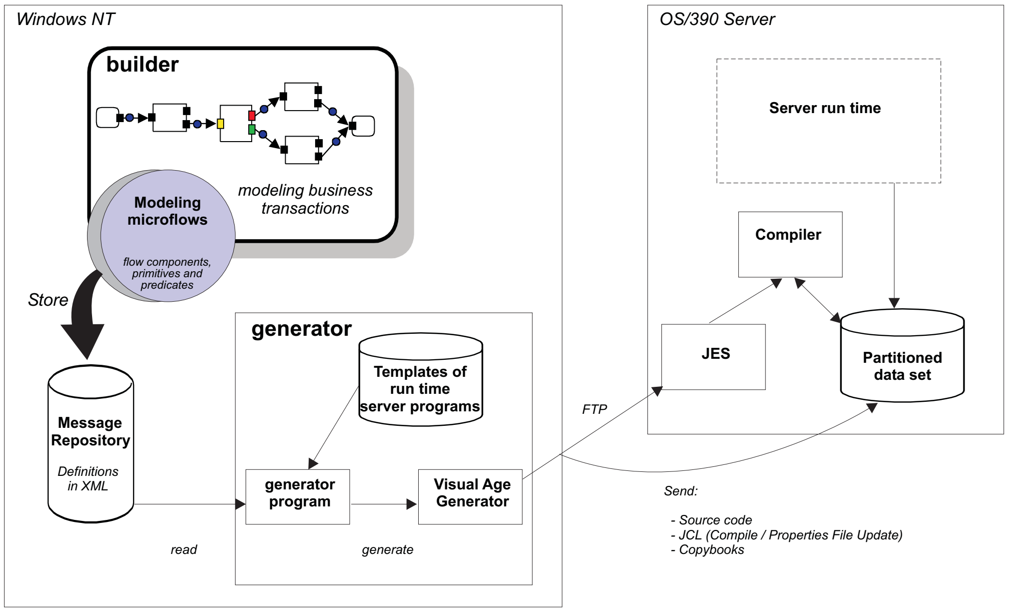


Figure 6. Deploying the adapter from the builder to the OS/390 server

See the section on deploying a new adapter to the run time environment in the Run Time User's Guide for information on how to define adapter resources to CICS and for information on running the Properties File Update Job (DFHMAMPU).

High level control flow of a CICS business transaction at run time

When the CICS adapter is moved to the run time environment, it exists as a process (BTS application) that is organized hierarchically. Data exchange is done through data-containers, named areas of storage, associated with a particular process or activity, and maintained by BTS. See *CICS Business Transaction Services* for information about CICS business transaction services (BTS)

The high level control flow of a typical CICS BTS business transaction is as follows:

1. A CICS transaction makes an initial request to start a process.

2. CICS BTS initiates a process instance - control activity. In MQSI Agent for CICS, this control activity is the Navigator.
3. The top level program associated with the control activity, using the CICS BTS API, creates a child activity or several child activities. In MQSI Agent for CICS, these child activities are the FEPI subflows, DPL adapters or MQ adapters.

   It provides the child activity with some input data (by placing the data in a data-container associated with the child), and requests CICS to start the child activity. If, as is often the case, the child activity is to run asynchronously with the control activity, the control activity returns.
4. The control activity (Navigator) is re-invoked when one of its child activities (FEPI subflow, DPL adapter or MQ adapter) completes. The control activity (Navigator) determines which event caused it to be re-invoked, that is, the completion of the activity that it started earlier. It retrieves, from the completed activity's output data-container, any return data that the completed activity has placed there.
5. Steps 3 and 4 are repeated until all the child activities that make up the CICS business transaction have completed.
6. The control activity then terminates.

Chapter 2. Tutorial overview

If you have not had any exposure to the Adapter Builder component of the MQSeries Integrator Agent for CICS Transaction Server product, you should read the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center*. This book contains information on the concepts of the MQSeries Integrator Agent for CICS Adapter Builder.

The *MQSI Agent for CICS Tutorial and Techniques* manual contains three separate tutorials that provide you with hands-on experience in modeling business transactions and with instructions on how to deploy the modeled transactions as *adapters* to a run time environment. An adapter is the output of the MQSeries Integrator Agent for CICS Adapter Builder. It consists of COBOL source code that is compiled and run in a CICS environment on an OS/390 server. The adapter implements a business transaction.

In order to deploy the adapters that you build from this tutorial, make sure that the MQSI Agent for CICS run time is installed and that your site has completed the installation verification procedure (IVP) as documented in the *MQSeries Integrator Agent for CICS Transaction Server Run Time User's Guide*.

The *MQSI Agent for CICS Tutorial and Techniques* manual also explains how to test and validate the adapters by providing instructions on:

- Defining adapter resources to CICS
- Invoking the adapter so that it can perform the business transaction(s) that you modeled. This is done through a supplied Simulator program.

The tutorials provide instructions on modeling adapters for each of the interface methods (DPL, MQ and FEPI) supported by the MQSeries Integrator Agent for CICS Transaction Server product.

About the business transaction that you will model

The business transaction to be modeled is the same in each tutorial. The transaction is a request for information on a customer. The customer information exists on a back-end system that can be accessed by any of the supported interfaces.

A group of programs and jobs that were provided and used during the IVP (Installation Verification Procedure) will simulate the back-end transactions in this tutorial.

- DFHMABP4, the back-end DPL customer information maintenance program
- DFHMABP6, the back-end MQ customer information maintenance program
- DFHMABP1, the back-end 3270 customer information maintenance program

Note: Modeling the FEPI interface requires capturing screens from the back-end system.

For instructions on modeling the adapter types supported by MQSI Agent for CICS, see

- Chapter 3, “Build an adapter that supports a DPL interface” on page 15

- Chapter 4, “Build an adapter that supports an MQ interface” on page 65
- Chapter 5, “Build an adapter that supports a FEPI interface” on page 111

Accessing the files to perform the tutorials

The files that you need to perform the tutorials are packaged on the Adapter Builder CD and were put on your system during the Adapter Builder installation procedure.

At the beginning of each of the tutorials there is information on how to access the tutorial files.

- See “Accessing the DPL tutorial files” on page 22 for information on how to get to the files needed for the **Building an adapter that supports a DPL interface** tutorial.
- See “Accessing the MQ tutorial files” on page 72 for information on how to get to the files needed for the **Building an adapter that supports a MQ interface** tutorial.
- See “Accessing the FEPI tutorial files” on page 115 for information on how to get to the files needed for the **Building an adapter that supports a FEPI interface** tutorial.

Assumptions

For each of the tutorials contained in *MQSI Agent for CICS Tutorial And Techniques*, the following is assumed:

- Version 1, release 1, modification 1 of MQSeries Integrator Agent for CICS Transaction Server has been installed. This includes both the builder component and the run time component.
- Object Rexx installed for deployment.
- The MQSI Agent for CICS run time installation verification procedure (IVP) has been completed.
- Users will have access (direct access, or indirect access through a CICS administrator) to the CICS region to which adapters will be deployed.
- Users will have access (an account and password) to the OS/390 server on which the adapters will run.
- Users will have access to CICS and OS/390 subject matter experts to help customize JCL templates and to help define adapter resources to CICS.
- That one person at a time will be performing the steps to design, create and deploy the adapter. If multiple persons will perform the tutorial at the same time and if the adapters created will be deployed in the same CICS region, the tutorial participants must make sure that the program names and transaction ids assigned to the generated adapter programs are unique.

Please refer to the product installation sections contained in the MQSI Agent for CICS builder and run time documentation for complete setup and configuration, including information on the IVP programs that will be used for validating the adapters built using these tutorials.

If there will be multiple tutorial participants deploying adapters to the same CICS region, each participant is responsible for managing the tutorial prefixes and specifications files so as to avoid transaction identifier and program name collisions in CICS.

Tutorial directory structure

The install wizard on the tutorial `setup.exe` allows you to pick the directory structure to house the required tutorial files. At installation, you are also allowed to pick and choose which of the three tutorials to install, the default indicating you want to install all 3 tutorials.

Figure 7 and Figure 8 on page 12 illustrate the complete tutorial installation that adheres to the **default settings**.

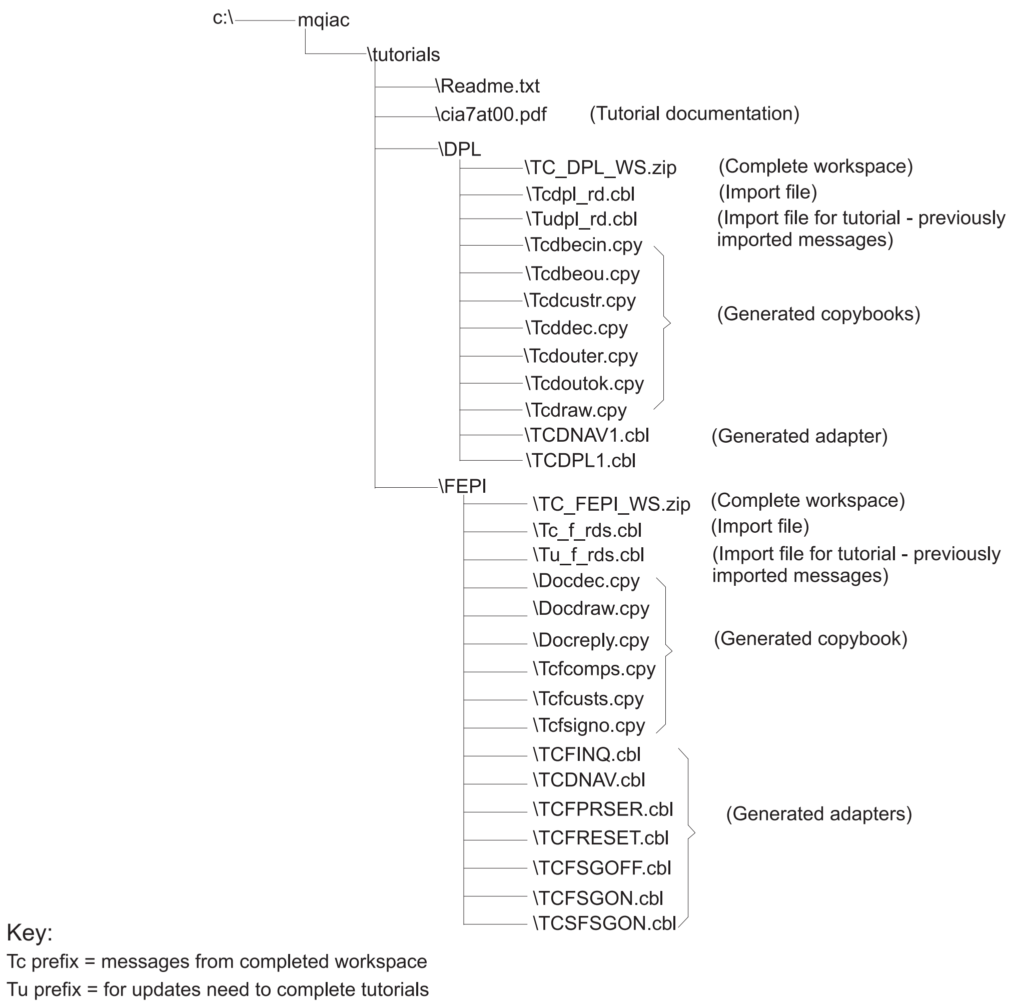


Figure 7. Tutorial installation for default settings (part 1)

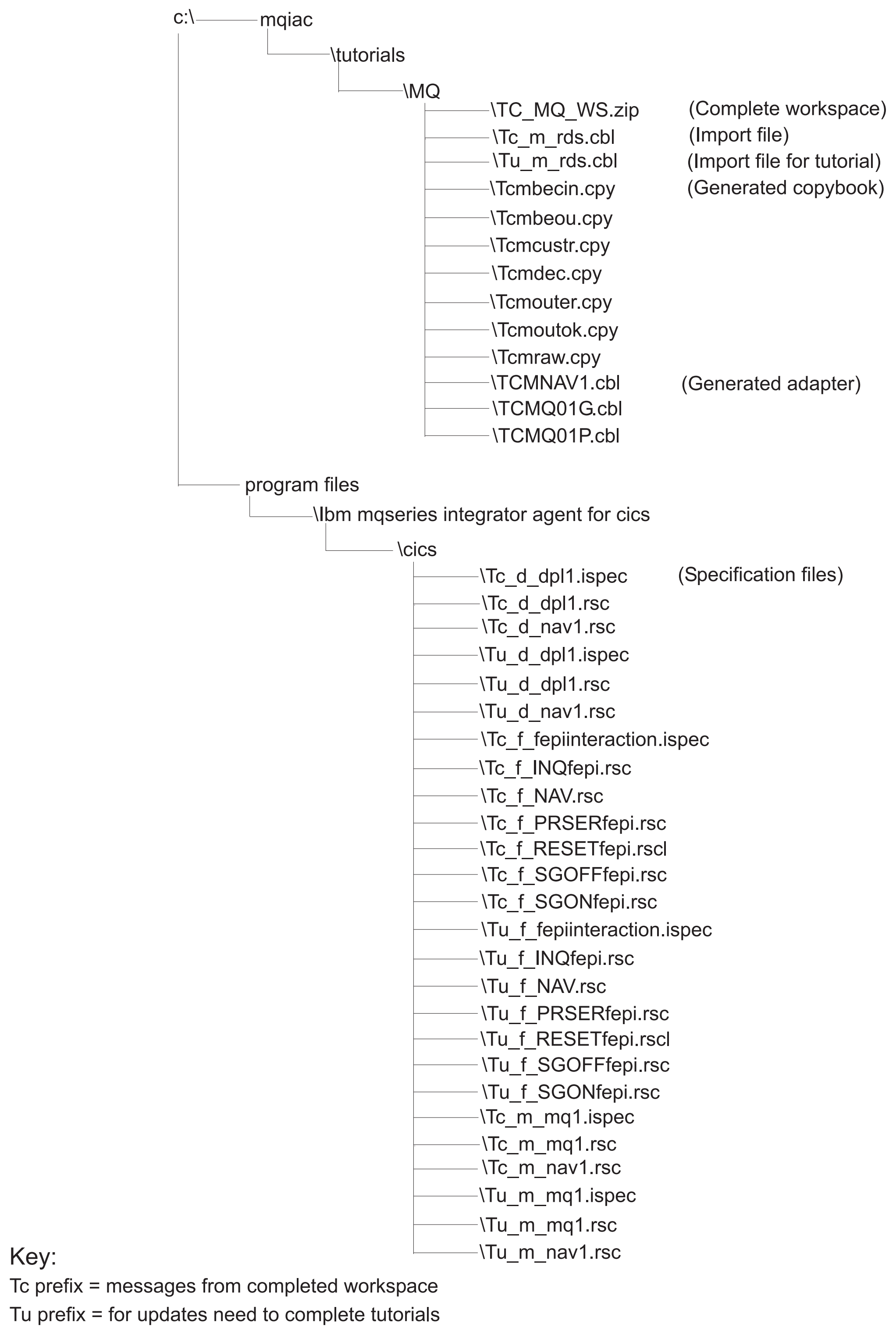


Figure 8. Tutorial installation for default settings (part 2)

Accessing a completed workspace

For each of the tutorials there is a .zip file that contains a sample of a completed workspace. This completed workspace is provided so that you can compare the workspace you build with one that has been proven valid. All the messages in the sample workspace are prefixed with *TC*. If you intend to import the completed workspace contained in .zip files for comparison with the tutorial workspace you create, then please prefix the message sets and messages you create with a different prefix (for example, *TU*) to avoid item duplication in your repository.

To view the provided workspace for any of the tutorials, you must first import the completed workspace into the Builder tool by performing the following steps:

1. Go to **File > Import Workspace** in the builder tool to bring up the Select a File to Import window
2. Find the <*mqiac_tutorials*>\ directory. Double-click on the zip file (for example `TC_DPL_WS.zip`).

   Next you will need to specify a name for the workspace you are importing. Specify a name of your choice (preferably a related name). Notice that the File Name has an .xml extension.
3. Choose **Open Workspace**, and select the .xml file you created on the previous step.

   Now, you will be able to view, and/or modify the workspace
4. To make modifications, once you open the newly renamed workspace, you must check-out individual components. For example, if you want to make changes to the microflow, right-click on the microflow name, and select **Check-Out**).
5. You will have to modify the following:
   - .ispec and .rsc files to point to your CICS environment.

     Note: If you installed the tutorial to a location other than C:\<mqiac_tutorials>, you must point to the location that you installed to when you generate the messages and adapters.

Chapter 3. Build an adapter that supports a DPL interface

Before you begin this tutorial, read Chapter 2, “Tutorial overview” on page 9. The Tutorial Overview section contains important information on the business transaction to be modeled, as well as information on the tutorial’s file structure. The Tutorial Overview also lists the assumed environment requirements that must be adhered to in order to build and deploy the adapter.

From this tutorial, you will learn how to use the MQSI Agent for CICS Adapter Builder tool to model and generate code for an adapter that supports a DPL interface.

You will model an adapter that has the functionality to access an existing CICS program via a Distributed Program Link (DPL). See “About the adapter you will design” on page 16 for a description of the adapter that you will model.

If you have not had any exposure to the Adapter Builder component of the MQSeries Integrator Agent for CICS Transaction Server product, you should read the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center*. This book contains information on the concepts of the MQSeries Integrator Agent for CICS Adapter Builder.

This tutorial provides instructions on:
- “Designing an adapter”
- “Creating an adapter that supports a DPL interface” on page 26
- “Deploying an adapter” on page 59

After completing this tutorial you should be able to:
- Identify required Host based information you need to gather and use.
- Import COBOL copybooks and create message sets.
- Create workspaces to define adapter flow logic.
- Create and generate a COBOL adapter.
- Deploy and test the generated COBOL adapter.

Before you begin this tutorial you should read Chapter 2, “Tutorial overview” on page 9. The Tutorial overview provides important information on the tutorial files, the tutorial directory structure and how to avoid naming conflicts when you create message sets and messages.

Designing an adapter

As was discussed in “Requirements analysis and design considerations” on page 1, before you start to use the MQSI Agent for CICS Adapter Builder, you would spend some time analyzing the business need that the adapter will address and then spend some time considering how you will design the adapter.

Build an adapter that supports a DPL interface

When you finish with requirements analysis and design considerations, you should have a sound understanding of how your adapter will *behave* at run time in order to manage and fulfill a business transaction.

To help you gain a frame of reference for what you will create in this tutorial, you should understand the following:

- The business need to be addressed
- The messages in and out structure
- The CICS resources required

Addressing a business need

An adapter should address a particular business need. In this tutorial, the business need is to provide a controlling application with an interface to a back-end environment for the purpose of accessing an existing CICS application (named DFHMABP4) that performs a customer inquiry.

In this tutorial, the adapter that you build will provide the interface to the back-end environment by way of a Distributed Program Link (DPL).

Note: For the purpose of this tutorial, the back-end environment that you will be accessing is the same back-end environment that was installed and used by the run time installation verification procedure (IVP). For information on the programs used by the IVP (including DFHMABP4), see the chapter on performing post installation tasks in the *MQSI Agent for CICS Run Time User's Guide*.

About the adapter you will design

The adapters that you build using the MQSI Agent for CICS Adapter Builder are visual models of business transactions. They are intended to map out the activities that comprise the entire business transaction, from invocation to completion.

The adapter that you build contains the instructions, logic and code that enable it to run on an OS/390 server, this includes an interface methodology for accessing information on back-end systems. In this tutorial the business transaction on which you will base your adapter is a *customer inquiry request* and the interface method used is a distributed program link (DPL) interface.

In this tutorial you will learn how to design and build an adapter (as displayed in Figure 9 on page 17) that when deployed will perform the following functions:

- Accept the structure TU_D_RAW from the Simulator (in this tutorial the Simulator functions as the controlling application). TU_D_RAW is the input record description from the controlling application. See Input Terminal in Figure 9 on page 17.
- Map individual fields from TU_D_RAW to expected and required DPL program Commarea format. See Map1 in Figure 9 on page 17.
- Execute the DPL link to the CICS application DFHMABP4 via the TU_D_DCUST command node.
- After the completion of the DPL link function, map response message from DPL program to store any customer demographic information that is supplied by the backend CICS application (DFHMABP4) in the data context area named TU_D_CUST_CTX. See Map2 in Figure 9 on page 17.
- Use a decision node to determine the success or failure of the DPL link to the CICS application DFHMABP4. Map the output message (TU_D_OUT_OK or TU_D_OUT_ERR) to be returned to the Simulator and exit.

Before building the adapter, you will need to:

- Define the CICS transaction IDs for the adapter programs that are generated.

  Note: As with any CICS application, the program names, transaction IDs, and CICS region, must adhere to your local site standards and your local site's naming conventions.

  For the purpose of this tutorial, the following programs will be generated.

Table 1. DPL Adapter programs

Program type	**Program name**	**Transaction ID**
Navigator	TUDNAV1	TUDN
DPL adapter	TUDPL1	TUD1

- Determine the DPL program that is invoked, as mentioned previously this is the existing CICS application named DFHMABP4. The request and response messages utilized are TU_BE_C_IN and TU_BE_C_OUT respectively.
- Determine the CICS region where the adapter programs will execute.
- Determine the CICS region where the DPL program will execute to access the back-end system.

When completed, the flow of components that make up your model will look like the following:

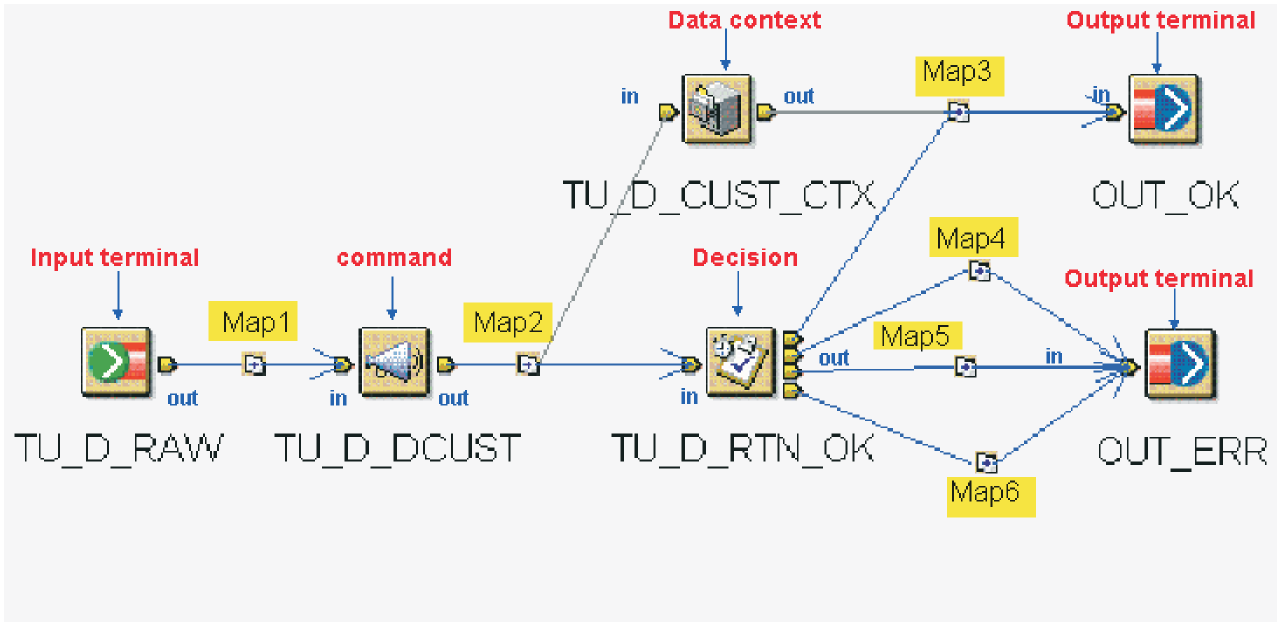


Figure 9. Components that make up the DPL adapter you will build

As stated previously, the business function that you will be modelling will enable a controlling application to invoke an existing CICS Application to retrieve customer data (from a back end application) on behalf of the controlling application.

To better understand the role that each component plays in your model of the business transaction, see Table 2 on page 18.

Build an adapter that supports a DPL interface

Use this table in conjunction with Figure 9 on page 17

Table 2. Component roles in the adapter that supports a DPL interface

Component	Name	Definition	Role / implementation
Input Terminal	TU_D_RAW	A primitive component that is used to represent data types that are input to a microflow. The term *primitive* indicates that inputs and outputs are visible to the user but their internals are not visible. TU_D_RAW contains the record description from the controlling application.	The purpose of this component within the context of modeling the transaction is to provide an entry point for the controlling application to the Navigator To implement this data transformation in your model, you will connect TU_D_RAW to the command node TU_D_DCUST by way of a **control connection**. A control connection provides the sequence relationship between two nodes in a microflow. On the Map node that sits on the control connection wire between TU_D_RAW and TU_D_DCUST, you will program the data transformation – in this case this means you will move the Customer Number provided by the Controlling application and specify the desired action to be performed by the existing CICS application DFHMABP4. By hard coding an **I** in the CUST_ACTION field, you are directing the CICS application DFHMABP4 to perform an inquiry on the customer record that correlates to the customer number.

Table 2. Component roles in the adapter that supports a DPL interface (continued)

Component	Name	Definition	Role / implementation
Command	TU_D_DCUST	A simple component that is used to represent application APIs that you import into the builder. The term *simple* indicates the component does not consist of other components.	The purpose within the context of modeling the transaction is to create a DPL Command type, which will allow the microflow to perform the DPL Link to the server-side DFHMABP4 program. The adapter will generate a COBOL program for every DPL command node. The program that is generated is the vehicle that allows the Navigator to interact with the existing CICS application DFHMABP4. On the map node that sits on the control connection wire between TU_D_DCUST and TU_D_RTN_OK, you will provide code that moves the output data (via a data control connection) provided by DFHMABP4 to a data context node named TU_D_CUST_CTX. The data context node holds data for future use. You will also provide instructions on the map node that moves the DFHMABP4 return code information to the decision node named TU_D_RTN_OK. Based on the return code provided, the decision node makes determination on the success or failure of the existing CICS application DFHMABP4.

Table 2. Component roles in the adapter that supports a DPL interface (continued)

Component	Name	Definition	Role / implementation
Decision	TU_D_RTN_OK	A composed component that is used to test a condition for true or false, to resolve the control flow path. The term *composed* indicates the component consist of other components that are connected by control flow connectors.	In your model of the business transaction, the Decision node will be used to evaluate the message indicator (Good, Warning or Error) upon return from the existing CICS application DFHMABP4. The Decision node will test the return code information that it receives from DFHMABP4. Based on the results of the test, the flow of the transaction will proceed in one of 4 ways (good, warning, error or default (unknown)) as indicated in Figure 9 on page 17. On the 3 map nodes that sit on the control connection wires between TU_D_RTN_OK and the OUT_ERR output terminal node, you will provide instructions that move the appropriate error information (depending on the error). This error information will be returned to the controlling application via the Output terminal. On the map node that sits on the control connection wire between the TU_D_RTN_OK decision node and the OUT_OK output terminal node, you will provide the instructions that move the customer demographic information (stored in TU_D_CUST_TRX data context node) along with the successful response information. This information will be returned to the controlling application via the Output terminal.

Table 2. Component roles in the adapter that supports a DPL interface (continued)

Component	Name	Definition	Role / implementation
Data Context	TU_D_CUST_TRX	A simple component that is used to store data for later access via a data connection.	You will need to create a Data Context type to store customer information. In the model, this Data Context Node is used to store the contents of the customer demographic information fields populated by the CICS application DFHMABP4. This data will then be provided to the controlling application or discarded once the Navigator determines the success or failure of the DPL link.
Output	OUT_OK	A primitive component that is used to represent data types that are output from a microflow.	The purpose of this component within the context of modeling the business transaction is to provide an exit point for the controlling application from the Navigator. In this model, the controlling application has been designed to receive 2 different types of reply messages. A successful reply and an error reply.
Output	OUT_ERR		

After some analysis, we determine that the host environment for the deployed adapter will look like Figure 10 on page 22. In this host environment, the generated adapter programs, TUDNAV1 and TUDPL1 execute in CICS region QAS1. The DPL Server Adapter accesses the back-end program DFHMABP4.

Note: In the following figure, DFHMABP4 resides in the same CICS region. It is also plausible, that the back-end program could reside outside of the CICS region, as illustrated by the dotted line.

Build an adapter that supports a DPL interface

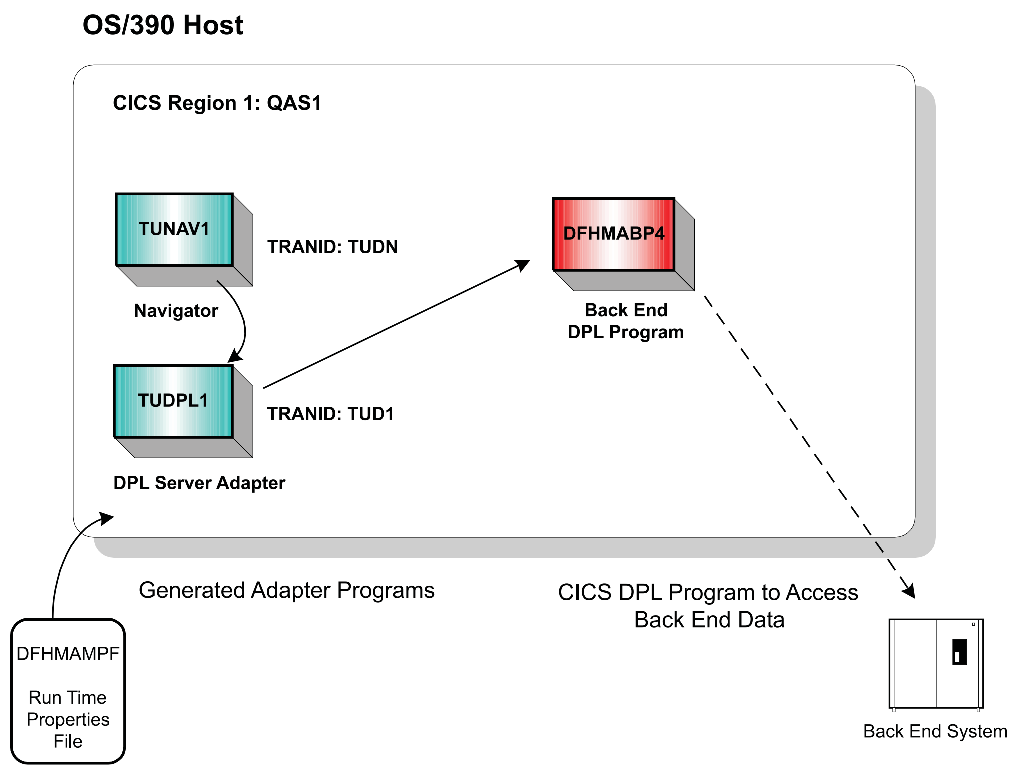


Figure 10. Tutorial run time environment for DPL adapter

Accessing the DPL tutorial files

The files you will need in order to build and deploy an adapter that supports a DPL interface are located in two directories as follows:

- C:\<mqiac_tutorials>\dpl
- C:\<mqiac_base>\cics

In the **C:\<mqiac_tutorials>\dpl** directory you will find the following files:

Table 3. Files to be used in the DPL tutorial

File name	Description	Use
TUDPL_RDS.cbl	COBOL record description.	Used as import for messages. Contains the message structures.
TC_DPL_WS.zip	Completed workspace for DPL adapter.	A completed workspace that you import and use as the basis for the workspace used to create the DPL adapter. See “Accessing a completed workspace” on page 12 for information on using the contents of this file
*.cpy files	Generated copybooks	The generated copybooks for the DPL adapter.

In the **C:\<mqiac_base>\cics** directory you will find the following files:

Table 4. Files in the C:\<mqiac_base>\cics directory

File name	Description	Use
tu_d_dpl1.ispec	Interaction specification file (tutorial version)	Assigns adapter name, DPL command type, CICS TransID, server side program, maximum Commarea length and SYNCONRETURN value.
tu_d_dpl1.rsc	Connector resource file (tutorial version)	Identifies CICS server region where server side program resides.
tu_d_nav1.rsc	Connector resource file (tutorial version)	Specifies synchronous rollback, Navigator type, COBOL program name for the DPL adapter and the CICS TransID.
Note: There is also a version of the Specification files prefixed with *tc_d_* that are used for the completed workspace supplied in the TC_DPL_WS.zip file.		

Configuring the Specification files for a DPL interface

GOAL

In this section you will learn how to configure the physical properties of the DPL adapter. These properties represent the XML definitions that are sent to the Properties file on the host at deployment time.

For information on the Properties file, see the MQSI Agent for CICS run time documentation.

Specification files are XML-format files that provide specific values to certain components created in MQSI Agent for CICS. An *Interaction Specification* file provides unique values for the component to which it is assigned. A *Connector Resource* file provides more general values for the component.

Some of the information in the Interaction Specification file and Connector Resource file maps to a run time properties file, DFHMAMPF. Other information in the Interaction Specification file is incorporated in generated Command and Navigator programs. The DFHMAMPF file stores data that is needed to run the generated adapter code programs on the host.

The DPL adapter requires specification files for its Command type and its Microflow type. The specification files for this tutorial are located in the `<mqiac_base>/cics` directory.

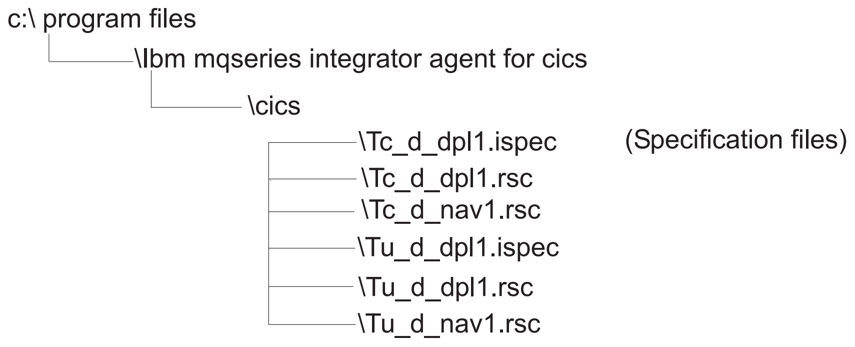


Figure 11. Directory structure for locating specification files for the DPL interface

You **must** configure the settings in the specification files used for the tutorial. The DPL Command type uses a Connector Resource file and an Interaction Specification file. In the Connector Resource file (`tu_d_dpl1.rsc`), the MAT_SYSID variable is used to specify the name of the CICS server region where the server side program resides. In the example, the MAT_SYSID variable has a value of QAS1. You can modify this to correspond to the CICS server region you are using.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
   <Attribute xmi.label="MAT_SYSID" type="" xmi.uuid="" valueMandatory="false"
      value="QAS1" encoded="false"/>
</AttributeGroup>
```

The Interaction Specification file for the DPL Command type used in the tutorial is `tu_d_dpl1.ispec`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Interaction Specification">
   <Attribute xmi.label="MAT_CMDTYPE" type="MAT_DPL MAT_MQ MAT_FEPI" xmi.uuid=""
         valueMandatory="true" value="MAT_DPL" encoded="false"/>
   <Attribute xmi.label="MAT_PROGID" type="" xmi.uuid="" valueMandatory="false"
         value="TUDPL1" encoded="false"/>
   <Attribute xmi.label="MAT_TRANID" type="" xmi.uuid="" valueMandatory="false"
         value="TUD1" encoded="false"/>
   <Attribute xmi.label="MAT_LINKNAME" type="" xmi.uuid="" valueMandatory="false"
         value="DFHMABP4" encoded="false"/>
   <Attribute xmi.label="MAT_LINKTRAN" type="" xmi.uuid="" valueMandatory="false"
         value="" encoded="false"/>
   <Attribute xmi.label="MAT_MAXCALEN" type="" xmi.uuid="" valueMandatory="false"
         value="401" encoded="false"/>
   <Attribute xmi.label="MAT_SYNCONRETURN" type="" xmi.uuid="" valueMandatory="false"
         value="N" encoded="false"/>
</AttributeGroup>
```

Table 5. Keyword values used for DPL Interaction Specification file

Keyword Symbolic	Description / Use	Example Value
MAT_CMDTYPE	Identifies the type of command	MAT_DPL
MAT_PROGID	The name of the COBOL program generated for the DPL command.	TUDPL1
MAT_TRANID	The CICS TransID for the server command program generated on the server.	TUD1
MAT_LINKNAME	The server side program to which a DPL-type command will link.	DFHMABP4

Table 5. Keyword values used for DPL Interaction Specification file (continued)

Keyword Symbolic	Description / Use	Example Value
MAT_LINKTRAN	The server side Transaction ID parameter (TRANSID) to specify on the DPL Link.	Blank (indicates TRANSID will not be used on the DPL Link)
MAT_MAXCALEN	Specifies the maximum Commarea length for the MAT_LINKNAME program (maximum 24576).	401
MAT_SYNCONRETURN	Specifies whether the SYNCONRETURN parameter is included on the DPL Link Y or N (default).	N

The Resource Connection file for the DPL Microflow type used in the tutorial is `tu_d_nav1.rsc`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Connector Resource">
   <Attribute xmi.label="MAT_REQTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="0" encoded="false"/>
   <Attribute xmi.label="MAT_NAVTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="R" encoded="false"/>
   <Attribute xmi.label="MAT_PROGID" type="" xmi.uuid="" valueMandatory="false"
         value="TUDNAV1" encoded="false"/>
   <Attribute xmi.label="MAT_TRANID" type="" xmi.uuid="" valueMandatory="false"
         value="TUDN" encoded="false"/>
</AttributeGroup>
```

Table 6. Keyword values used for DPL Microflow Connector Resource file

Keyword Symbolic	Description / Use	Example Value
MAT_REQTYPE	Specifies whether the request is run on the server in asynchronous, synchronous or synchronous rollback mode 0 (asynchronous) 1 (synchronous) 2 (synchronous rollback)	0
MAT_NAVTYPE	Specifies whether the Microflow Type is a base Navigator (R) or a FEPI Navigator (F)	R
MAT_PROGID	The name of the COBOL program generated for the DPL microflow.	TUDNAV1
MAT_TRANID	The CICS TransID for the server command program generated on the server.	TUDN

Build an adapter that supports a DPL interface

You have just configured the `tu_d_dpl1.ispec` file and the `tu_d_nav1.rsc` file. You are ready to create the adapter that supports a DPL interface.

Creating an adapter that supports a DPL interface

GOAL

In this section you will learn how to use the adapter builder to create the model of the business transaction.

Specifically, you will learn how to import the necessary COBOL record descriptions and system interfaces for the DPL adapter. These are stored in the logical message model in the Adapter Builder for use in the DPL microflow.

Follow these instructions to begin the process of building an adapter that supports a DPL interface:

__ Step 1. **Start the builder and create a new workspace**.

To start the builder, go to the **Start** > **Programs** > **IBM MQSI Agent for CICS** >**IBM MQSI Agent for CICS**. This will launch the tool as shown below, in Figure 12 on page 27.

You should begin the tutorial with a new workspace. A workspace is a view of what you can work with at one time. A workspace is displayed as the graphical space in the builder where you will build the adapter to support the DPL interface.

From the **File** pull-down menu, select **New Workspace**.

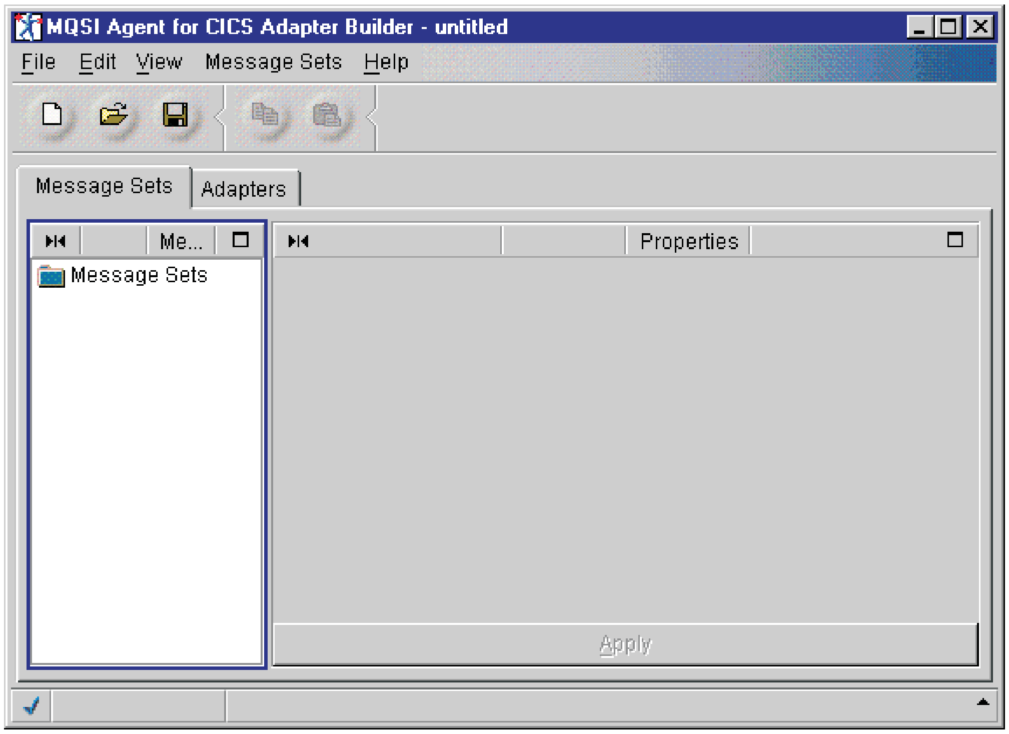

MQSI Agent for CICS Adapter Builder - untitled
File Edit View Message Sets Help
Message Sets
Adapters
Me...
Message Sets
Properties
Apply

Figure 12. Initial panel of the MQSI Agent for CICS Adapter Builder

__ Step 2. **Name your tutorial workspace and save it to the repository**.

From the **File** pull-down menu, select **Save Workspace**. Enter a name for the workspace, such as TU_DPL_WS, and click **Save**.

Note: Be sure to use under_scores and not dashes "-" when naming the workspace.

__ Step 3. **Import a message set**.

A message set is a collection of structured XML-based data types that are stored in the message repository.

When you import a message set, what you are really doing is bringing in the COBOL structured data type definitions from existing CICS transactions on the host system, into the Adapter Builder's control center. The imported data type definitions contain the record descriptions of the messages.

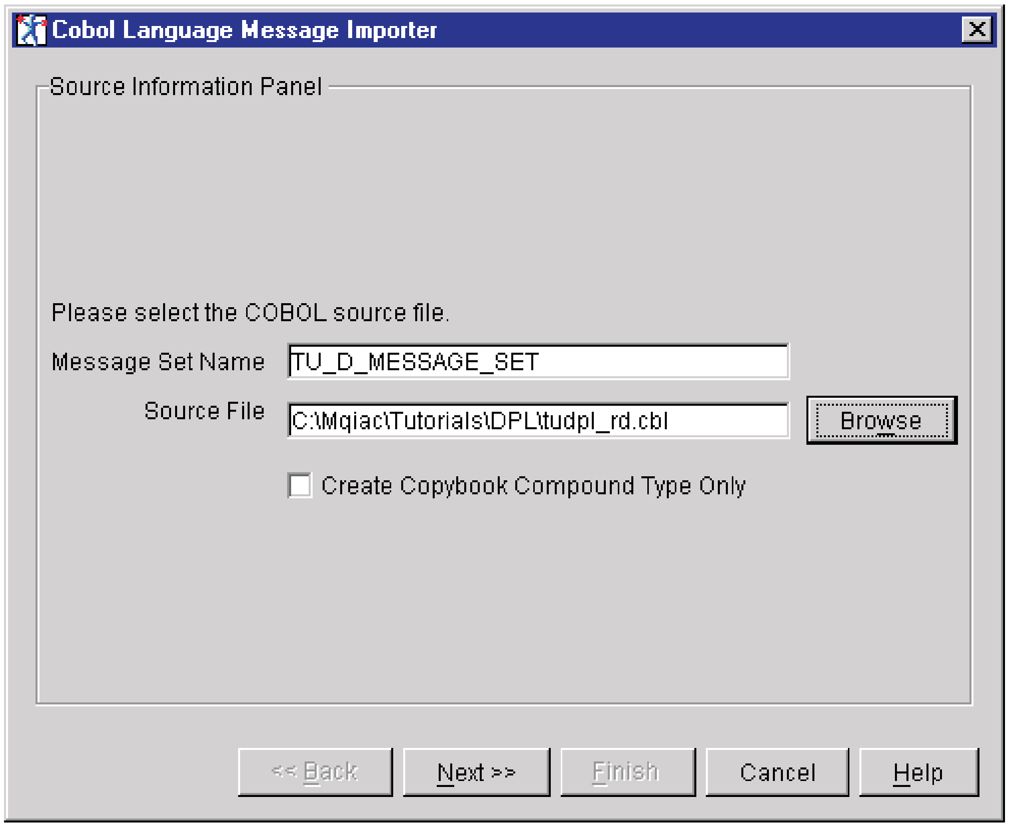
The control center utilizes the message set as an interface between the adapter builder tool and the business transaction to be modelled.

Note: You cannot import the COBOL structured data type definitions directly from CICS. You must first FTP the structured data type definitions from the host to a workstation. You can then import the message set from the workstation.

After importing a message, you can modify and store it.

Note: It is much easier to import a COBOL structured data type definition than it is to build the message set. If there is no record description, create one with a text editor and import it.

a. Right click on the Message Sets folder, select **Import to New Message Set > COBOL**.



Please select the COBOL source file.

Message Set Name: TU_D_MESSAGE_SET

Source File: C:\Mqiac\Tutorials\DPL\tudpl_rd.cbl

☐ Create Copybook Compound Type Only

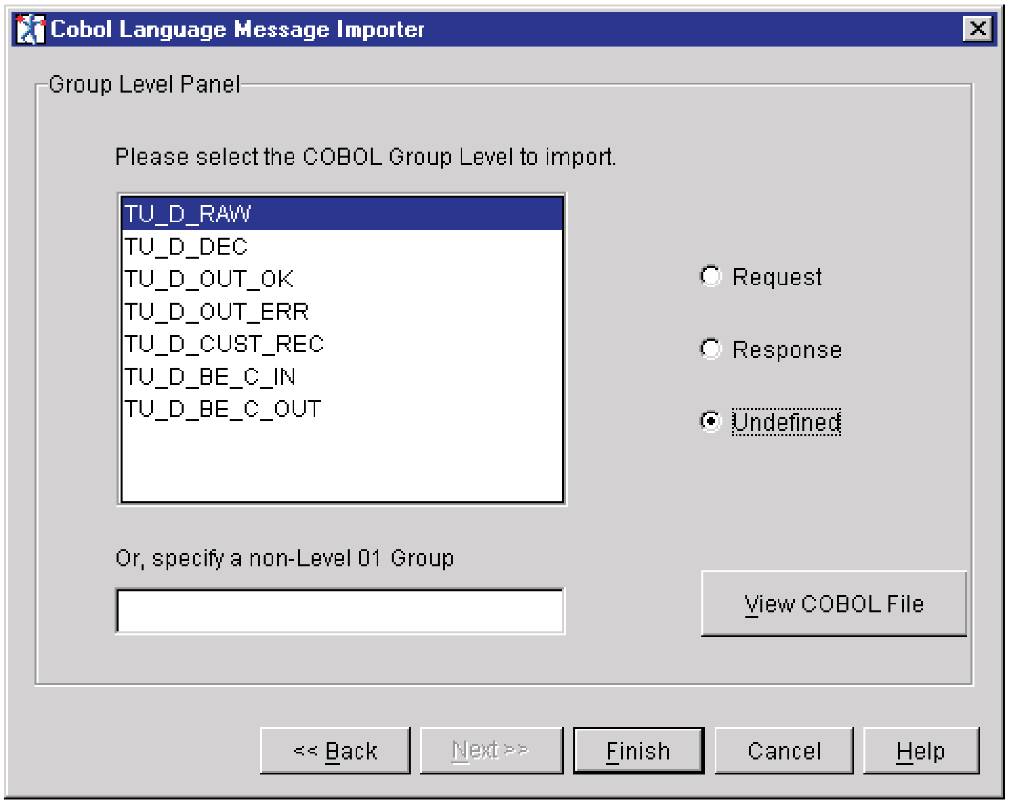
Figure 13. Import a message set (source information)

On the COBOL Language Message Importer dialog (Source Information Panel), enter the Message Set Name (in the tutorial, TU_D_MESSAGE_SET) and the directory path where you installed the tutorial (<mqiac_tutorials>\dpl) and the name and location of the copybook from which you will be importing the message set (**<mqiac_tutorials>\dpl\tudpl_rd.cbl**).

For the purposes of this tutorial, leave the **Create Copybook Compound Type Only** box unchecked. This box is an option that controls *how* copybooks can be imported.

Click **Next**.

b. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import:


Cobol Language Message Importer

Group Level Panel

Please select the COBOL Group Level to import.

- TU_D_RAW
- TU_D_DEC
- TU_D_OUT_OK
- TU_D_OUT_ERR
- TU_D_CUST_REC
- TU_D_BE_C_IN
- TU_D_BE_C_OUT

Request [✓ Undefined] Response

Or, specify a non-Level 01 Group

Figure 14. Import a message set (group level)

The radio button selections are as follows:

Request
Use if the message is going to be used as an input message in a transaction.

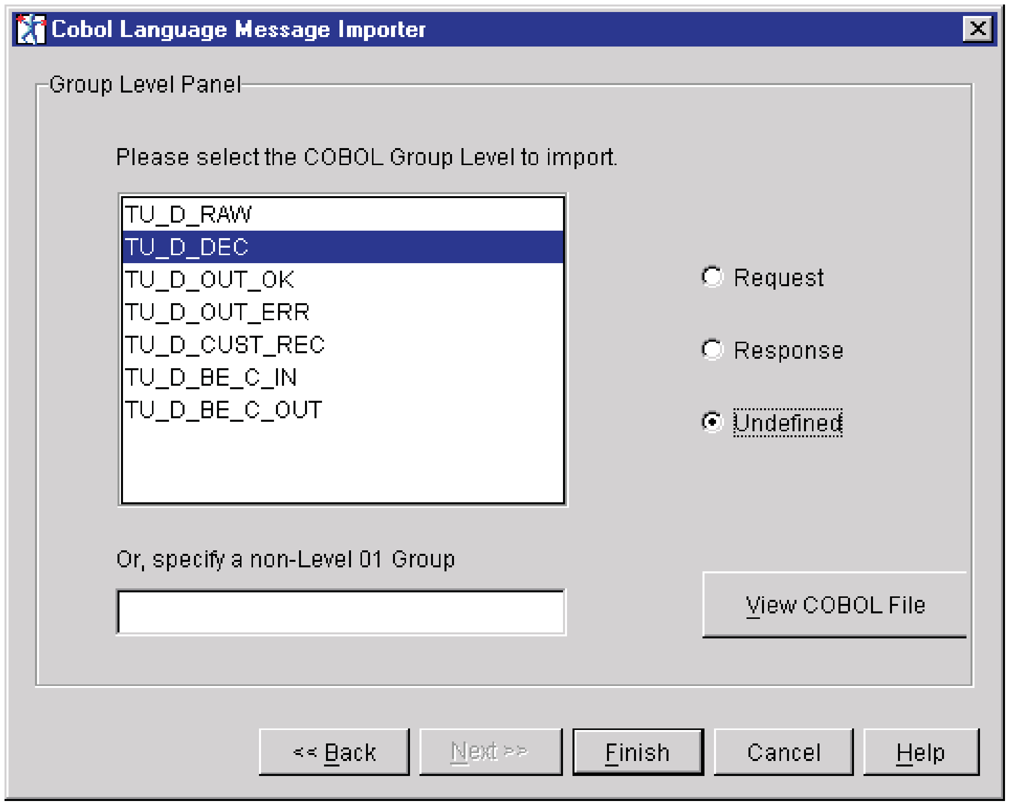
Response
Use if the message is going to be used as an output message in a transaction.

Undefined
Can be used for messages that are not used in a transaction.

For this tutorial, select TU_D_RAW and select the Undefined message type radio button. Click **Finish** to complete the import.

c. Right click on the newly created TU_D_MESSAGE_SET folder and select **Import to Message Set > COBOL**. On the COBOL Language Message Importer dialog (Source Information Panel), enter the directory path where the Source Files for the copybooks are located (see Figure 13 on page 28). Click **Next**.

d. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import (for the tutorial, select TU_D_DEC) and select the Undefined message type radio button. Click **Finish** to complete the import.



Selected: TU_D_DEC; Request [✓ Undefined] Response

Figure 15. Import a message set (group level)

e. Click **Next** to return to the COBOL Language Message Importer dialog (Group Level Panel).

f. Repeat the procedure in step d until all of the following messages (with the specified message types) are imported as follows:

Table 7. Messages to add to the workspace

Message	Message Type	Purpose
TU_D_RAW	Undefined	Input message used by the navigator to receive information from the controlling application.
TU_D_DEC	Undefined	Decision node message used by the navigator to determine how to flow logically within the flow. This message provides a series of fields, the context of which are evaluated by the Navigator to control logical flow.

Table 7. Messages to add to the workspace (continued)

Message	Message Type	Purpose
TU_D_OUT_OK	Undefined	The output message used by the Navigator to provide customer demographic information to the controlling application in the event that the customer inquiry request was executed successfully.
TU_D_OUT_ERR	Undefined	Error output message used by the Navigator to provide error information to the controlling application in the event that the customer inquiry request was not executed successfully due to the fact that the customer inquiry failed.
TU_D_CUST_REC	Undefined	Customer record layout utilized by the navigator to store customer demographic information supplied by the existing CICS application DFHMABP4.
TU_D_BE_C_IN	Request	DPL Commarea Input message used to supply the existing CICS application DFHMABP4 with the information that it requires to execute properly.
TU_D_BE_C_OUT	Response	DPL Commarea output message used to receive customer demographic information as provided by the existing CICS application DFHMABP4.

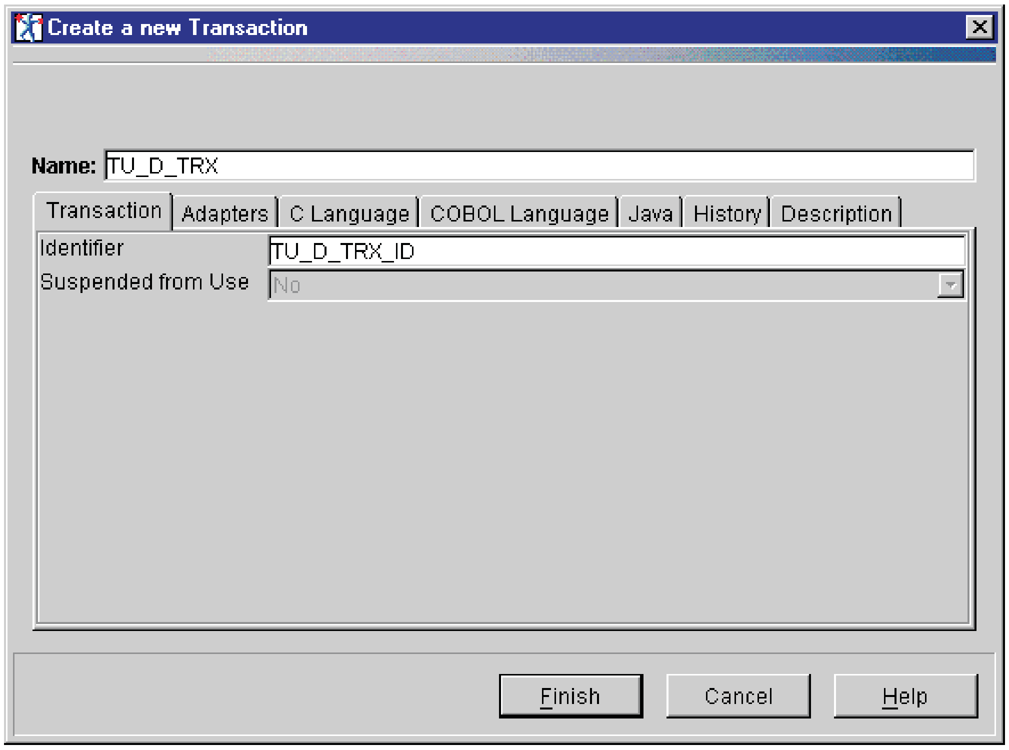
g. When you have completed importing the COBOL structured data type definitions listed in Table 7 on page 30, click **Cancel** to return to the workspace.

You just completed importing the COBOL structured data type definitions needed to model the DPL adapter. These data type definitions now reside as messages in the Control Center of the Adapter Builder.

__ Step 4. **Create Transactions.**

A transaction represents the message and data flowing to and from the back-end DPL program to be accessed by the adapter. In order to create a DPL command node, you need to associate the command node with a transaction. The messages associated with the transaction are defined as Input and Output representing the expected format of the input message (and identified as input terminal in the node) and the expected format of the output message (and identified as the output terminal in the node).

a. Create a transaction for the customer information. Right click on the Transaction folder. Select **Create > Transaction**. On the Create a new Transaction dialog, enter TU_D_TRX in the Name field and TU_D_TRX_ID in the Identifier field. Click **Finish**.



Name: TU_D_TRX

Identifier: TU_D_TRX_ID

Suspended from Use: No

Figure 16. Create a TU_D_TRX transaction

b. Add messages to the TU_D_TRX transaction. Right click on the TU_D_TRX transaction and select **Add > Message**. On the Add an Existing Message dialog (the messages exist in the message set, from when you imported them) select the TU_D_BE_C_IN and TU_D_BE_C_OUT messages (press the CTRL key and highlight both messages) and click **Finish**.

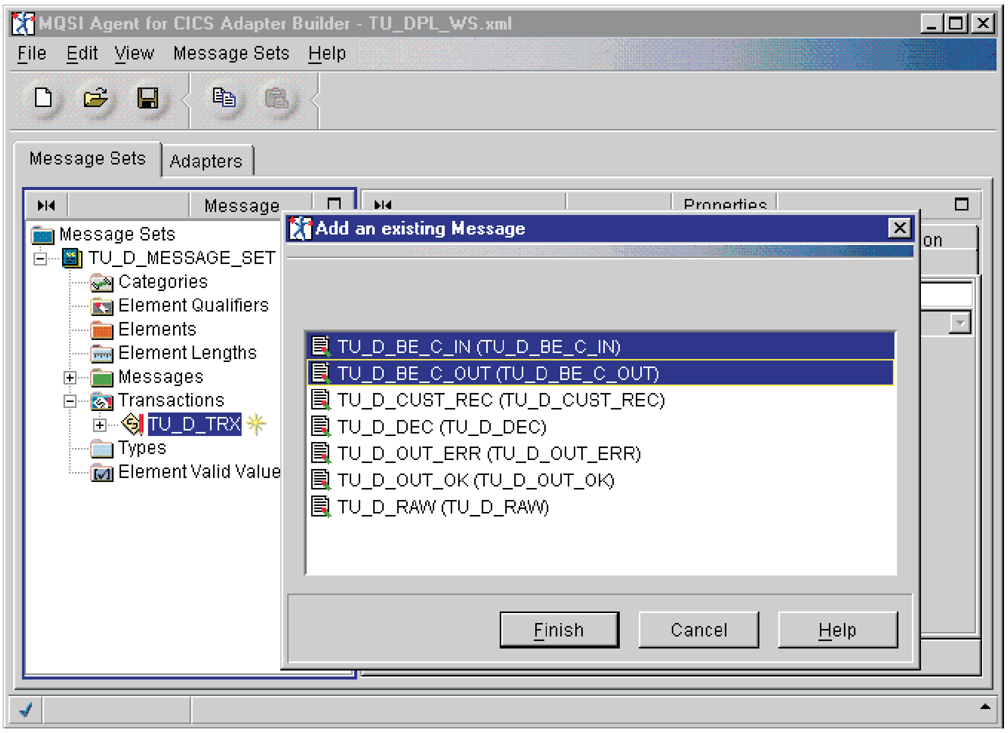


Figure 17. Add messages to the TU_D_TRX transaction

At this point, after adding messages to transactions the Message Sets view will appear as shown in Figure 18.

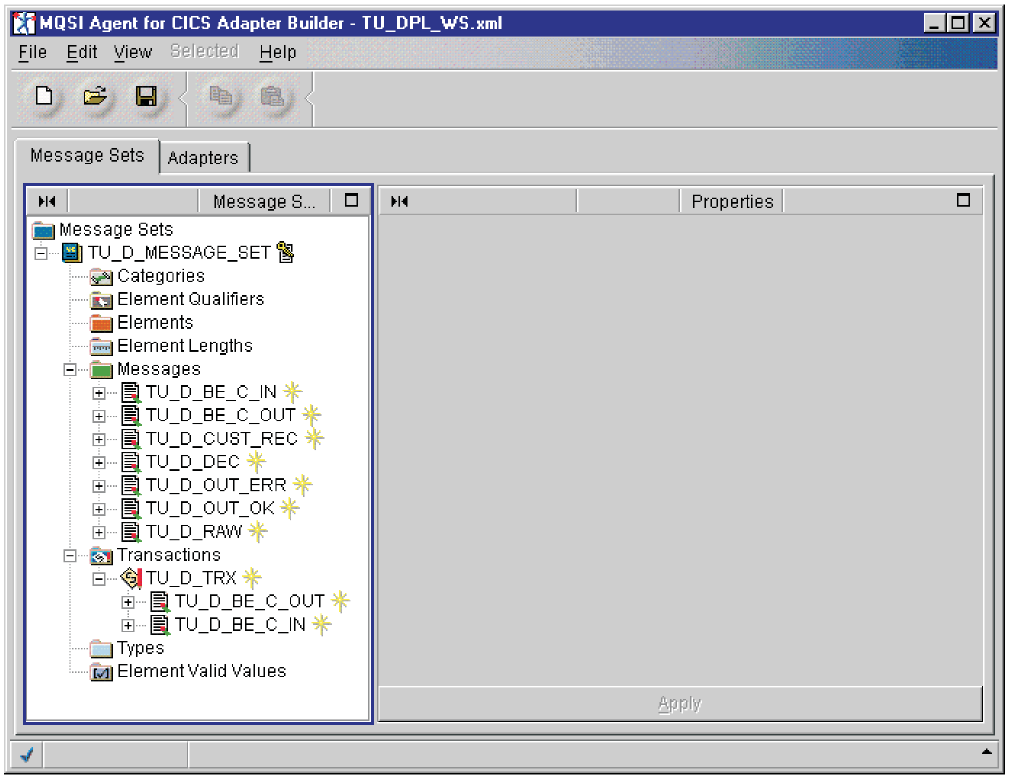


Figure 18. Messages Sets view

Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created the DPL transaction and associated the input and output messages to the transaction.

You are now ready to create the component types. In this next step, you will associate the command component type with the transaction that you just created.

__ Step 5. **Create the component types for use in the microflow**

A component type represents a template that can be used as a building block in modeling the microflow.

When you complete the tasks in this step, you will have all the necessary component types required to model the adapter's functionality. The component types will display in the Adapter Tree View. From the Adapter Tree view you will be able to drag a component type onto the Microflow Definition pane and begin the process of constructing the flow.

This step is made up of the following tasks:

- Create a Decision Type
- Create a Command Type
- Create a Data Context Type
- Create a Microflow Type

See the section *Composing microflows* in the *MQSI Agent for CICS Using the Control Center* documentation for descriptions of the component types.

a. **Create a Decision Type**.

A decision type is necessary to test a condition for true or false, to resolve the control flow path.

You will use this type to create a Decision node for the microflow. The Decision node will be used to evaluate the message indicator (Good, Warning or Error) upon return from the DPL program, (DFHMABP4) and it will *decide* how processing will continue.

1) Click on the Adapters tab to switch to the Adapters view.

2) Create the TU_D_RTN_OK Decision type that will be used to determine whether the data returned from the back-end host is valid. Right click on the Decision Types folder and select **Create > Decision Type**. Enter *TU_D_RTN_OK* in the Name field and click **Finish**.

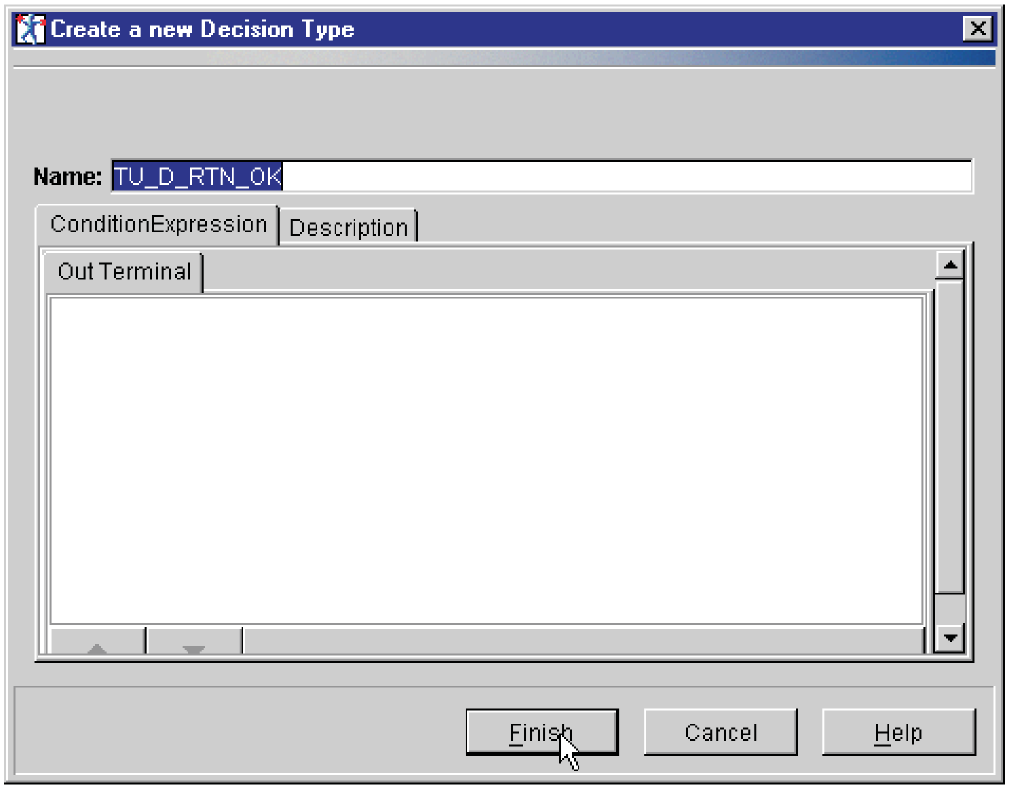
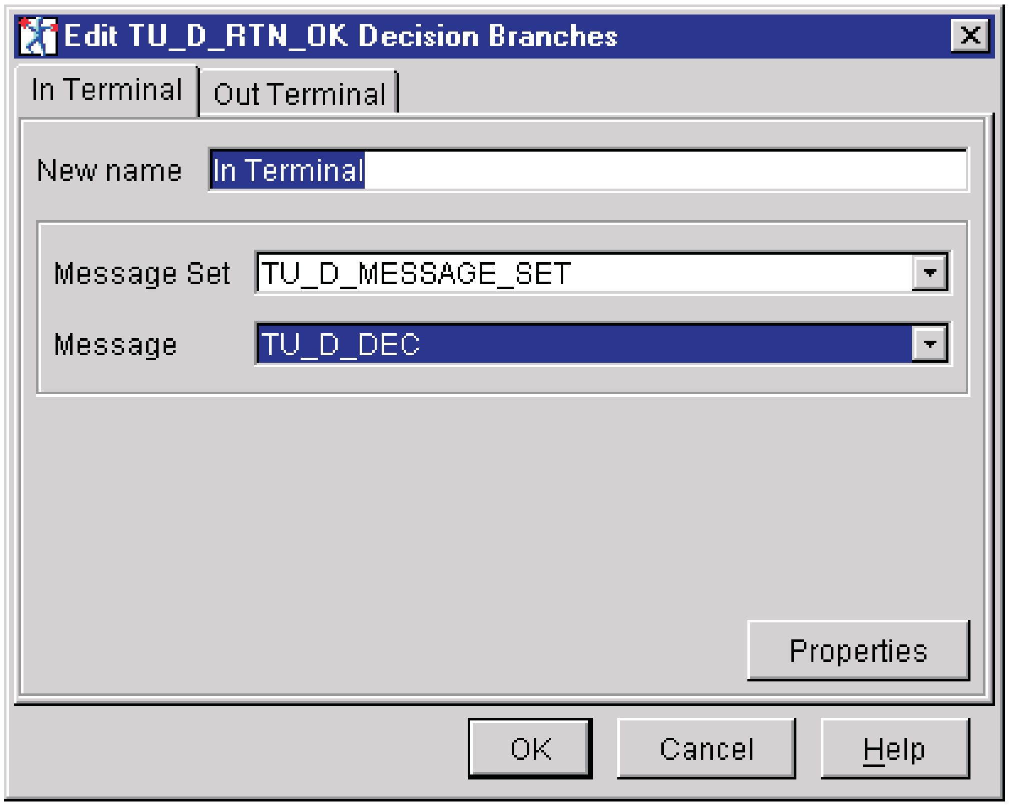


Figure 19. Creating a Decision type for the DPL Adapter

3) Associate a message set and message with the In Terminal on the TU_D_RTN_OK Decision type.

Right click on the TU_D_RTN_OK Decision type under the Decision Types folder and select **Decision Branch**.

Make sure the In Terminal tab is selected. Using the drop down menus, select TU_D_MESSAGE_SET for the Message Sets field and TU_D_DEC for the Message field.



New name: In Terminal

Message Set: TU_D_MESSAGE_SET

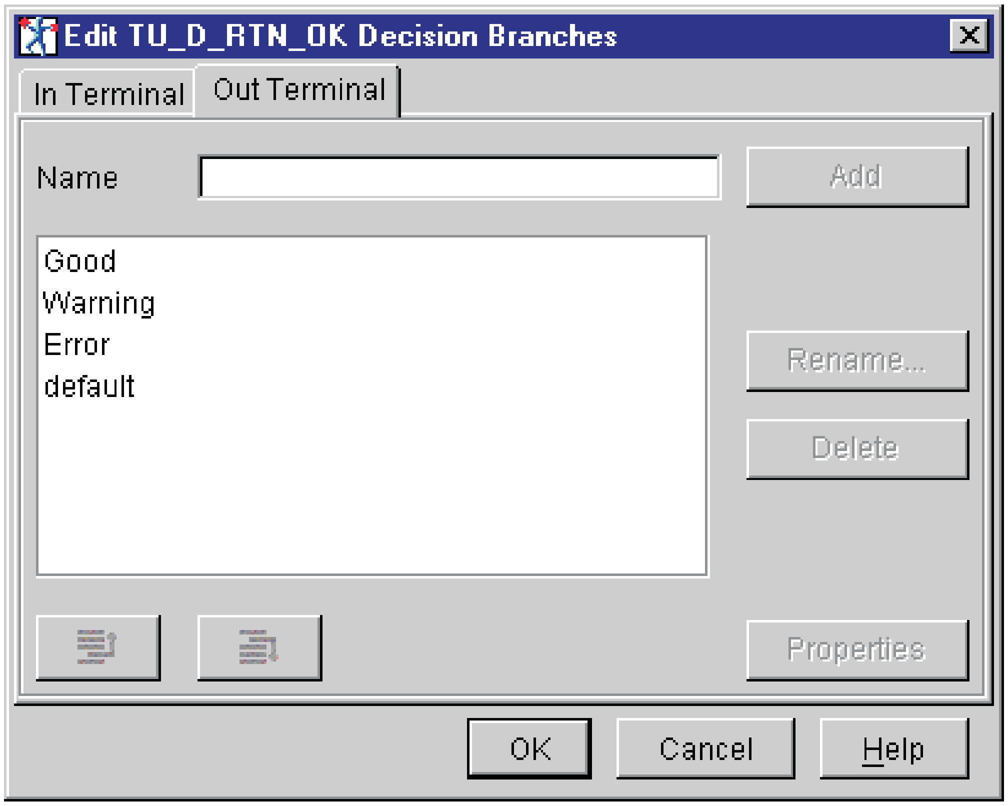
Message: TU_D_DEC

Figure 20. Editing the In Terminal on the Decision type

4) Create Out Terminals for the Good, Warning, and Error decisions.

   The TU_D_RTN_OK Decision type will determine which of these actions to take based on the MSG_IND_D field in the decision message (TU_D_DEC).

   a) Make sure the Out Terminal tab is selected. Click Out Terminal in the terminal list box and click **Rename**. Enter *Good* in the New name field and click **Finish**.
   b) Enter *Warning* in the Name field and click **Add**.
   c) Enter *Error* in the Name field and click **Add**. Click **OK**.



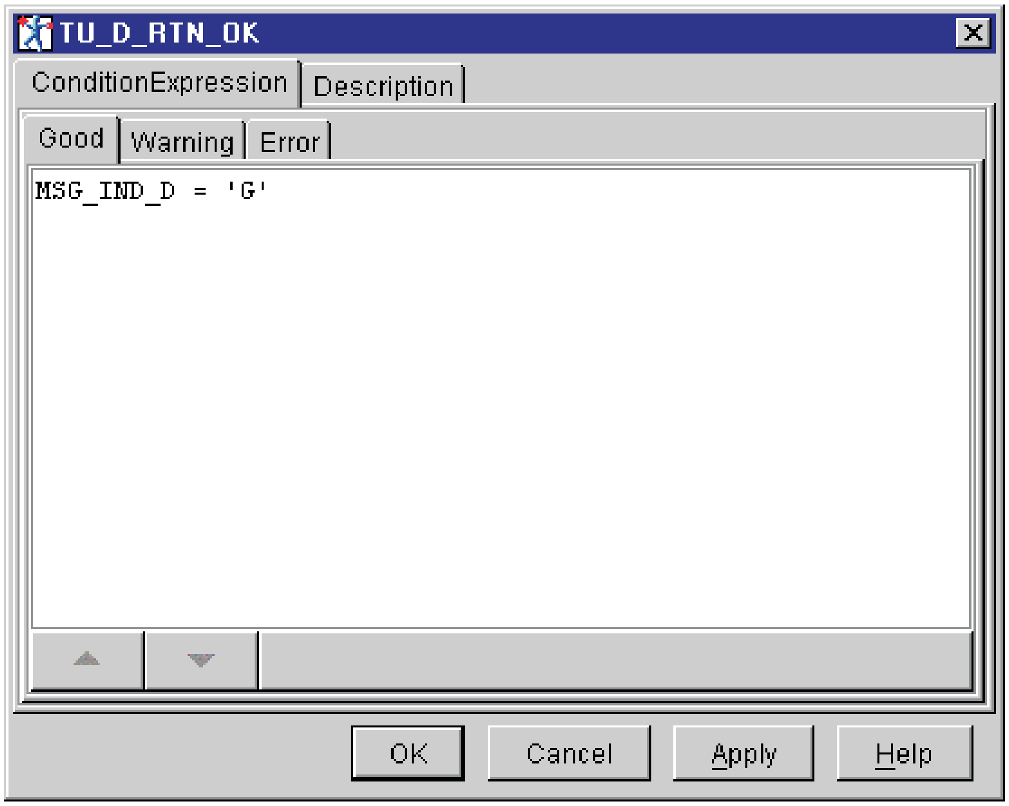
Good
Warning
Error
default

Figure 21. Editing the Out Terminal on the Decision type

5) Create conditional expressions in simple SQL

Right click on the TU_D_RTN_OK Decision type and select **Properties** on the pop up menu. Make sure the ConditionExpression tab is selected and the Good tab is selected.

Click in the Good test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_D_DEC message that we associated with the TU_D_RTN_OK Decision type). Select the *MSG_IND_D* field to add this to the Input area of the ConditionExpression tab.



```
MSG_IND_D = 'G'
```

Figure 22. Code for the Good Terminal

You should add the code shown in Figure 22 for the Good terminal test condition. The letter ’G’ for the MSG_IND_D field is based on the message indicator action codes that are defined for the decision message (TU_D_DEC).

6) In a similar manner, add the test condition code for the remaining terminals: Warning and Error. When finished, click **OK**.

Table 8. Code for the Out Terminal actions for the TU_D_RTN_OK Decision type

Terminal	Code	Description
Good	MSG_IND_D = ’G’	G - Good - request processed
Warning	MSG_IND_D = ’A’	A - Application warning (e.g. ’Record Not Found’)
Error	MSG_IND_D = ’E’	E - System error (e.g. ’File Closed’)

b. **Create a DPL Command Type.**

A command type is a simple adapter component which, depending on how its properties are set, can be used to represent a server adapter program (DPL, MQ) or FEPI command (3270 screen interaction).

In this step you need to create a DPL Command type which will allow the microflow to perform the DPL Link to the server-side DFHMABP4 program.

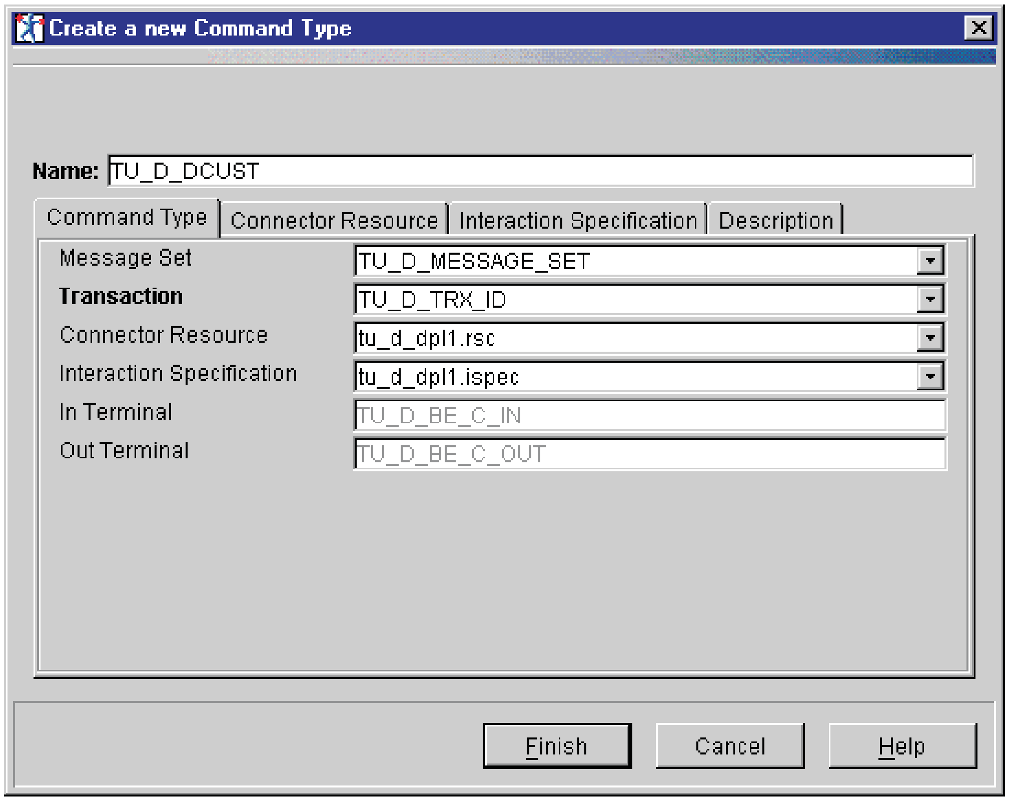
1) Create the TU_D_DCUST Command type for the DPL Command.

a) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_D_DCUST* in the Name field.

b) Using the drop down menus, set the following field property values:

Table 9. DCUST Command property values

Field	Value
Message Set	TU_D_MESSAGE_SET
Transaction	TU_D_TRX_ID
Connector Resource	tu_d_dpl1.rsc
Interaction Specification	tu_d_dpl1.ispec


Create a new Command Type

Name: TU_D_DCUST

Message Set	TU_D_MESSAGE_SET
Transaction	TU_D_TRX_ID
Connector Resource	tu_d_dpl1.rsc
Interaction Specification	tu_d_dpl1.ispec
In Terminal	TU_D_BE_C_IN
Out Terminal	TU_D_BE_C_OUT

Figure 23. Creating a TU_D_DCUST Command type

Click **Finish** to apply the property values.

c. **Create a Data Context Type.**

A data context type is a simple adapter component that is used to store data for later access through a data flow.

In this step you will need to create a Data Context type to store customer information. This data can be accessed later from a connector data flow.

This Data Context Node is used to store the contents of the customer demographic information fields populated by the CICS application DFHMABP4. This data will then be provided to the controlling application or discarded, once the Navigator (TUNAV1) determines the success or failure of the DPL link.

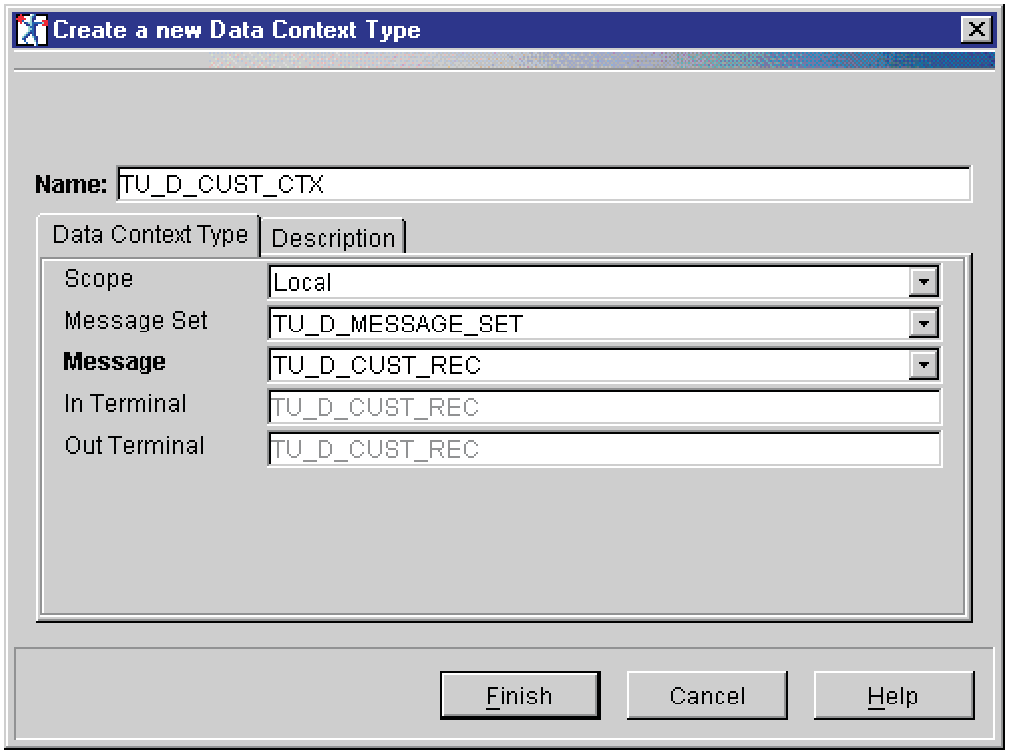
1) Create the TU_D_CUST_CTX Data Context type.

a) Right click on the Data Context Types folder and select **Create > Data Context Type**. Enter *TU_D_CUST_CTX* in the Name field.

b) Using the drop down menus, set the following field property values:

Table 10. TU_D_CUST_CTX Data Context property values

Field	Value
Scope	Local
Message Sets	TU_D_MESSAGE_SET
Message	TU_D_CUST_REC



Name: TU_D_CUST_CTX

Scope	Local
Message Set	TU_D_MESSAGE_SET
Message	TU_D_CUST_REC
In Terminal	TU_D_CUST_REC
Out Terminal	TU_D_CUST_REC

Figure 24. Creating a TU_D_CUST_CTX Data Context type

Click **Finish** to apply the property values.

d. **Create a microflow type**.

A microflow type is a collection of adapter components that models all or part of the message processing. In your adapter, this is the Navigator that calls the transaction and is responsible for controlling adapter request processing and managing states during the microflow processing.

A navigator invokes server adapter programs.

In this step you will create a microflow that will model the processing of the customer data request.

1) Right click on the Microflow Types folder and select **Create > Microflow Type**.

2) Enter TUDPL01 in the Name field.

3) Use the drop down menu in the Connector Resource field to select tu_d_nav1.rsc as the Connector Resource file and then click **Finish**.

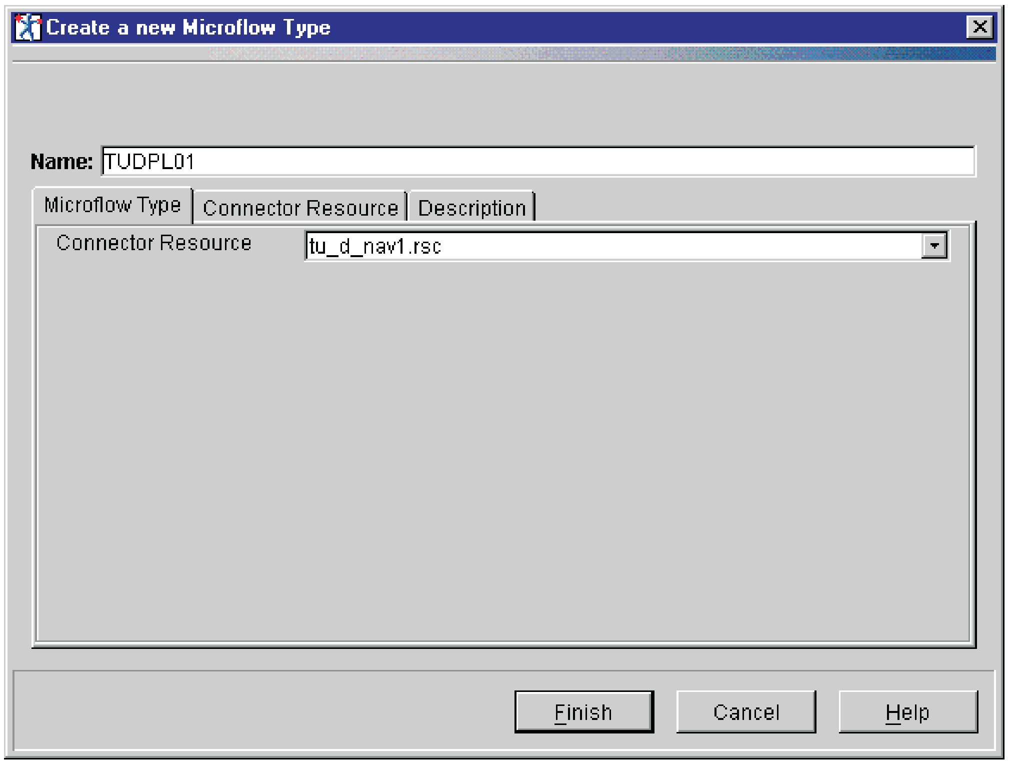


Figure 25. Creating a TUDPL01 Microflow Type

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created all of the component types that you will need to model your adapter.

__ Step 6. **Model the adapter**

In this step you will perform a set of tasks to model the adapter.

When you model an adapter you are indicating how the adapter will function at run time. Within the context of the business flow, the adapter model is of the navigation of the server application with the back end systems. The adapter represents the behavior you need to access data from the existing back end applications.

Within the builder, the model of the adapter is represented as a *microflow*, a sequence of nodes and connections. The microflow models the processing of a message as it passes from the input of the adapter to the output of the adapter.

Build an adapter that supports a DPL interface

This step is made up of the following tasks:

- Adding microflow nodes
- Connecting the microflow nodes
- Defining the mappings

a. **Add the microflow nodes** .

In this task, you will drag all the component types that you created in step 5 on page 34, onto the Microflow Definition pane. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

1) **Add the Input Terminal node**

An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

a) Drag the node on to the Microflow Definition pane.

In the Microflow Types folder, select the TUDPL01 microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

Drag an Input Terminal type from the Adapter Tree View to the Microflow Definition pane.

Left click and hold on the Input Terminal to drag it to the Microflow Definition pane:

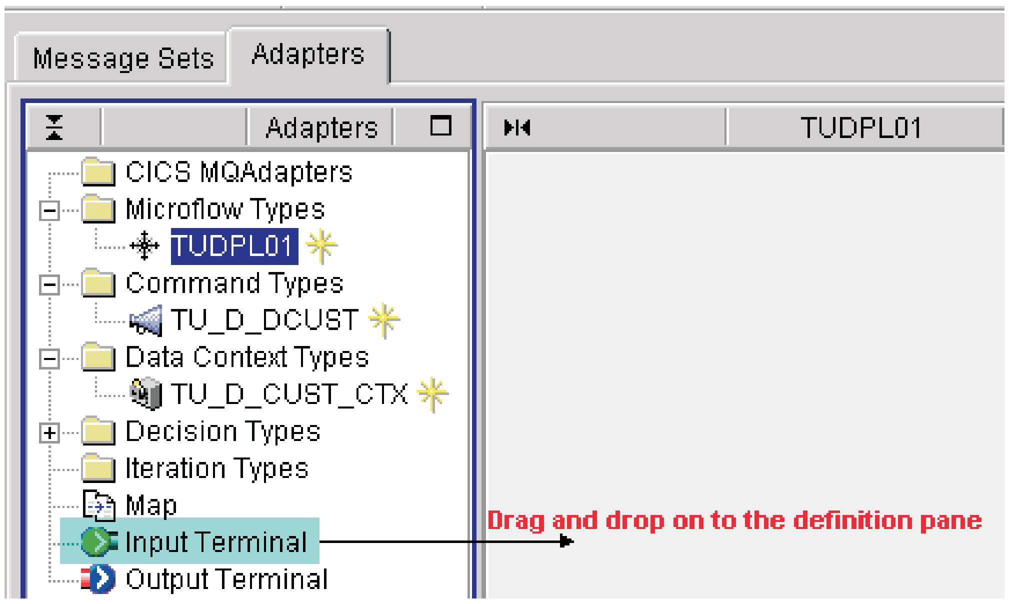


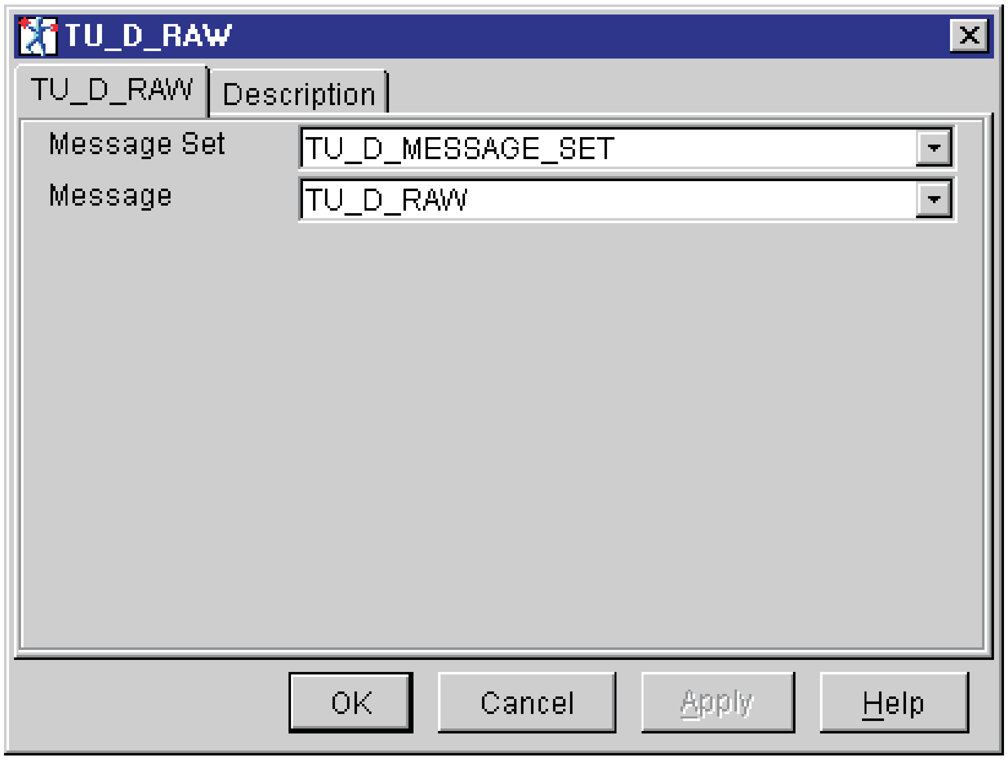
Figure 26. Dragging an Input Terminal on to the Microflow Definition pane

b) Rename the node

Right click on the Input Terminal located in the definition pane and select **Rename**. Rename the Input Terminal node to TU_D_RAW and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. Make sure the TU_D_RAW tab is selected. From the drop down menus, select TU_D_MESSAGE_SET in the Message Set field and select TU_D_RAW in the Message field. Click **OK**.



Message Set TU_D_MESSAGE_SET
Message TU_D_RAW

Figure 27. Configuring the TU_D_RAW Input Terminal node properties

2) **Add the Command node**
   a) Drag the node on to the Microflow Definition pane

   From the Command Types folder in the Adapter Tree View, select a TU_D_DCUST Command type.

   Left click and hold on the TU_D_DCUST Command type to drag it to the Microflow Definition pane. Place the node to the right of the TU_D_RAW Input Terminal node.
   b) Rename the node

   Right click on the TU_D_DCUST1 Command node and select **Rename**. Modify TU_D_DCUST1 in the New name field to the name TU_D_DCUST and click **Finish**.
3) **Add the Decision node**
   a) Drag the node on to the Microflow Definition pane

   Drag a TU_D_RTN_OK Decision type from the Adapter Tree View to the Microflow Definition pane. Place the node to the right of the TU_D_DCUST Command node.
   b) Rename the node

Right click on the TU_D_RTN_OK1 Decision node and select **Rename**. Modify TU_D_RTN_OK1 in the New name field to the name TU_D_RTN_OK and click **Finish**.

4) **Add the Data context node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_D_CUST_CTX Data Context type from the Adapter Tree View to the workspace. Place the node above the TU_D_RTN_OK Decision node.
   b) Rename the node

      Right click on the TU_D_CUST_CTX1 Data Context node and select **Rename**. Modify TU_D_CUST_CTX1 in the New name field to the name TU_D_CUST_CTX and click **Finish**.
5) **Add the Output terminal node**

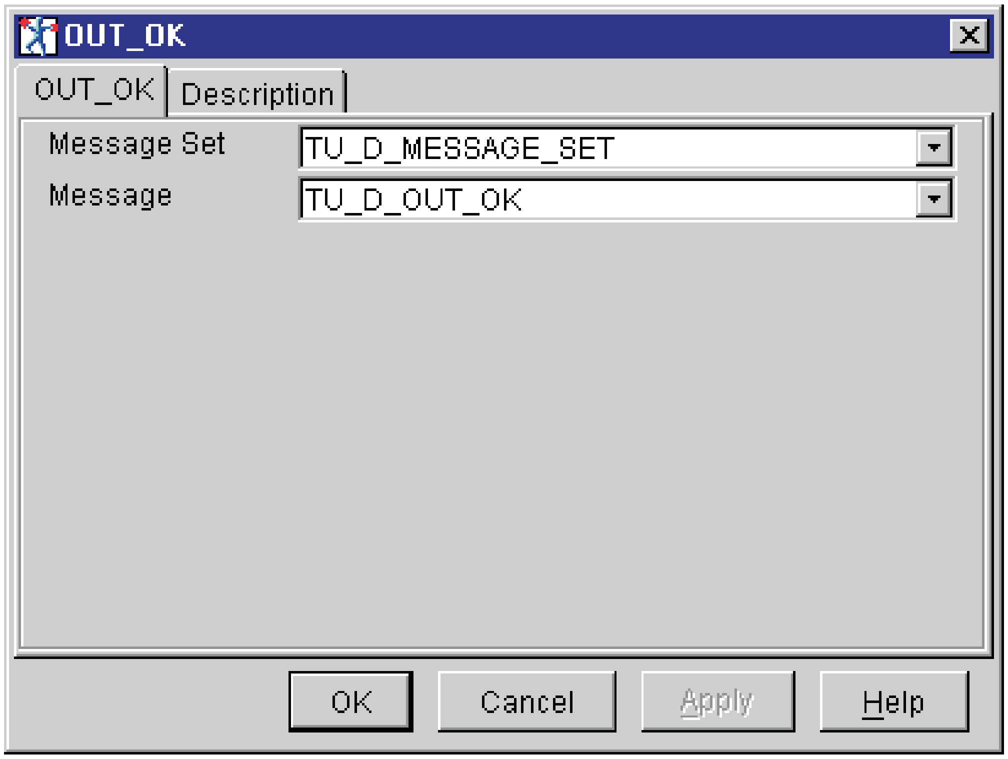
   An Output Terminal serves as an exit point for the microflow. The Output Terminal can receive connections only. It can never start a connection. A microflow can have multiple Output Terminals (as in the DPL example). A developer must design the controlling application to recognize the possible reply messages provided by multiple Output Terminals.
   a) Drag the node on to the Microflow Definition pane

      Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_D_CUST_CTX Data Context node.
   b) Rename the node

      Rename the Output Terminal node to OUT_OK.
   c) Flip the node

      Right click on the OUT_OK and select **Flip node**
   d) Set the properties for the node

      Right click on the OUT_OK node and select **Properties**. Make sure the OUT_OK tab is selected. From the drop down menus, select TU_D_MESSAGE_SET in the Message Set field and select TU_D_OUT_OK in the Message field. Click **OK**



Message Set TU_D_MESSAGE_SET
Message TU_D_OUT_OK

Figure 28. Configuring the OUT_OK Output Terminal properties

6) **Add the Error Output terminal node**
   a) Drag the node on to the Microflow Definition pane

      Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_D_RTN_OK node.
   b) Rename the node

      Rename the Output Terminal node to OUT_ERR.
   c) Flip the node

      Right click on the OUT_ERR and select **Flip node**
   d) Set the properties for the node

      Right click on the Output Terminal and select **Properties**. Make sure the OUT_ERR tab is selected. From the drop down menus, select TU_D_MESSAGE_SET in the Message Sets field and select TU_D_OUT_ERR in the Messages field. Click **OK**.
7) Save your workspace by selecting **File > Save Workspace** from the menubar.

Your Microflow Definition panel should look something like this:

Build an adapter that supports a DPL interface

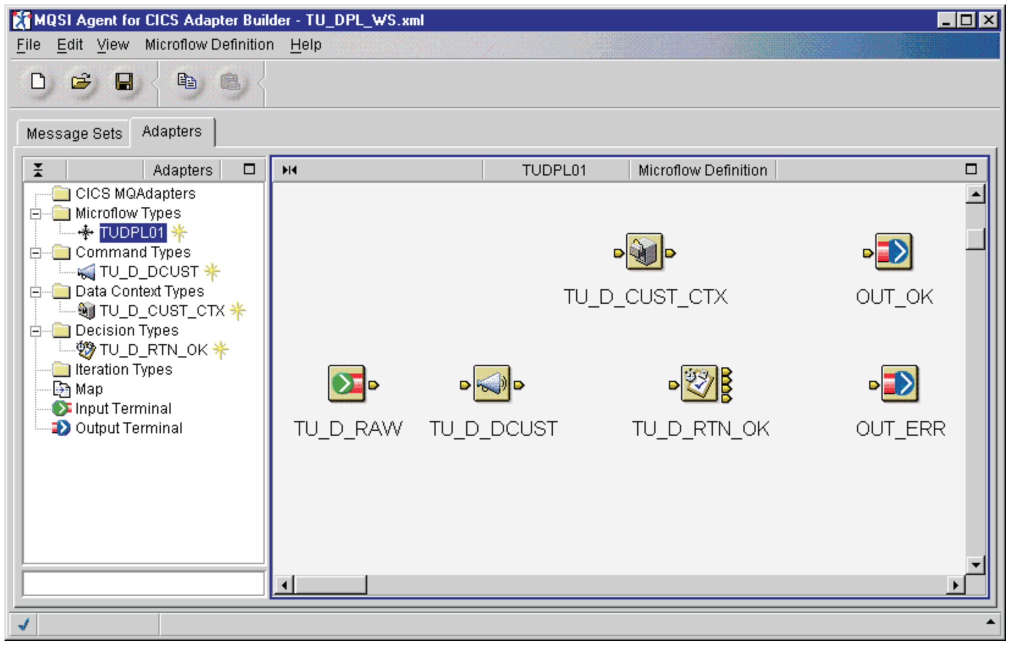


Figure 29. Nodes for the DPL adapter

b. **Connect the microflow nodes**.

In this task you will connect the microflow nodes that are on the Microflow Definition pane so as to define the flow of processing. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the TU_D_RAW Input Terminal node and select **Connect > Out**. Move the connection line to the TU_D_DCUST Command node and left click. This adds a control connection and a map (Map1 node) between the two nodes.

A control connection provides a sequential relationship between 2 nodes in a microflow.

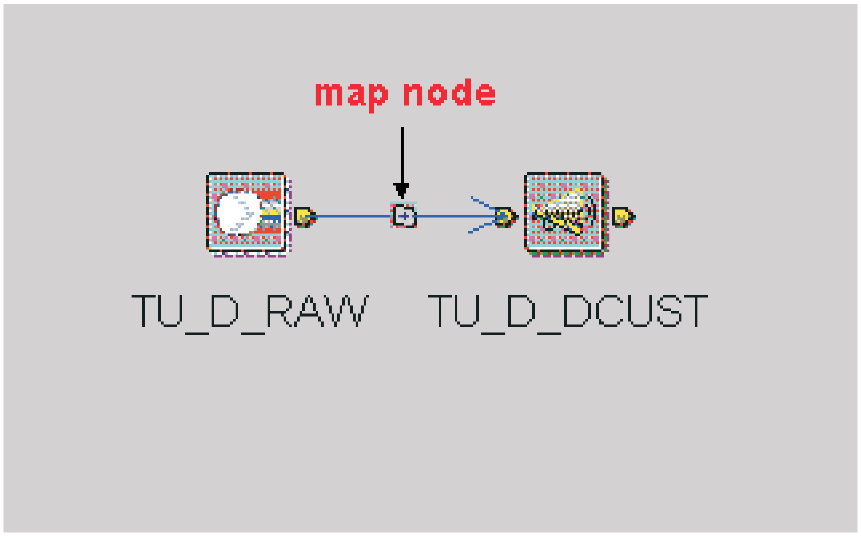


Figure 30. Connecting the TU_D_RAW Input Terminal and TU_D_DCUST Command node .

2) Add a control connection from the out terminal on the TU_D_DCUST Command node to the TU_D_RTN_OK Decision node. This auto-adds a Map2 node on the control connection line.
3) Add a Map node (Map3) between the TU_D_RTN_OK Decision node and the OUT_OK Output Terminal node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel).
4) Add a control connection from the first out terminal (labeled Good) on the TU_D_RTN_OK Decision node to the Map3 node and from the Map3 node to the OUT_OK Output Terminal node.
5) Add a series of three Map nodes (Map 4–Map 6) between the TU_D_RTN_OK Decision node and the OUT_ERR node.

   Note: See Figure 31 on page 48 to see the placement of the four Map nodes (Map 3–Map 6). The Map node labels are for annotation only and will not appear in your workspace.
6) Add control connections from the second out terminal (labeled Warning) on the TU_D_RTN_OK Decision node to the Map4 node and from the Map4 node to the OUT_ERR Output Terminal node.
7) Add control connections from the third out terminal (labeled Error) on the TU_D_RTN_OK Decision node to the Map5 node and from the Map5 node to the OUT_ERR Output Terminal node.
8) Add control connections from the fourth out terminal (labeled default) on the TU_D_RTN_OK node to the Map6 node and from the Map6 node to the OUT_ERR Output Terminal node.
9) Add a data connection from the Map2 node to the TU_D_CUST_CTX Data Context node and from the out terminal of the TU_D_CUST_CTX node to the Map3 node.

Refer to Figure 31 to see all of the node connections in the microflow.

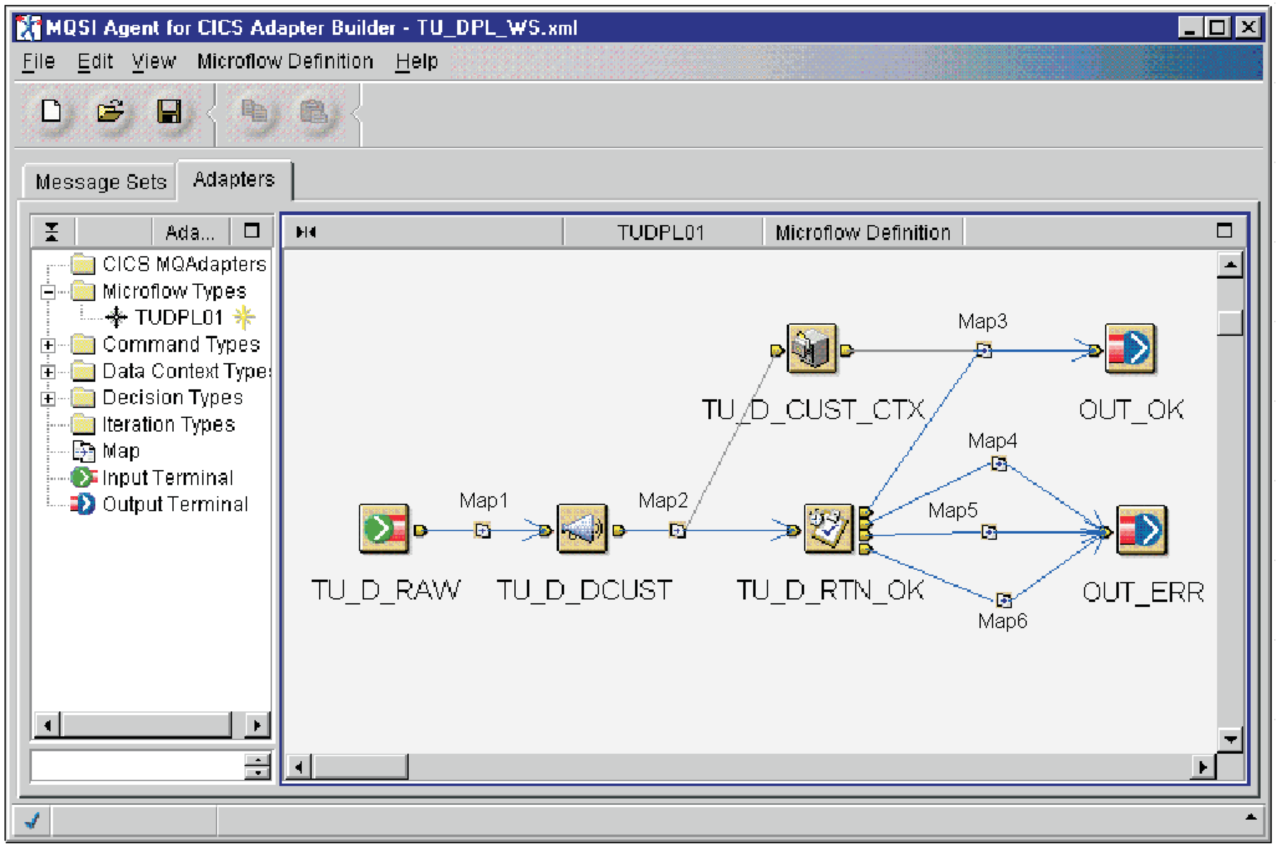


Figure 31. The TUDPL01 microflow

10) Save your workspace by selecting **File > Save Workspace** from the menubar.

c. **Map your adapter**.

You are now ready to *map* your adapter. The act of *mapping* refers to the modeling of data transformation via a Map node, between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 11 on page 49 and shown in Figure 32 on page 50. This map passes the customer number and inquire action indicator 'I' to the DPL program DFHMABP4.

a) Right click on the Map1 node (the Map node that appears between the TU_D_RAW and TU_D_DCUST nodes) and select **Properties**. Click the DataMappingExpression tab.

b) Left click on the CUST_NO field under the TU_D_RAW message (view input message on right of panel) and drag the mouse cursor to the CUST_DATA_I field under the TU_D_BE_C_IN message (view output message on left of panel). This will create a mapping between the two fields (see Table 11 and Figure 32 on page 50).

c) The second mapping (’I’ to CUST_ACTION_I) is a literal mapping. To perform a literal mapping, display the Map node’s properties and make sure the DataMappingExpression tab is selected.

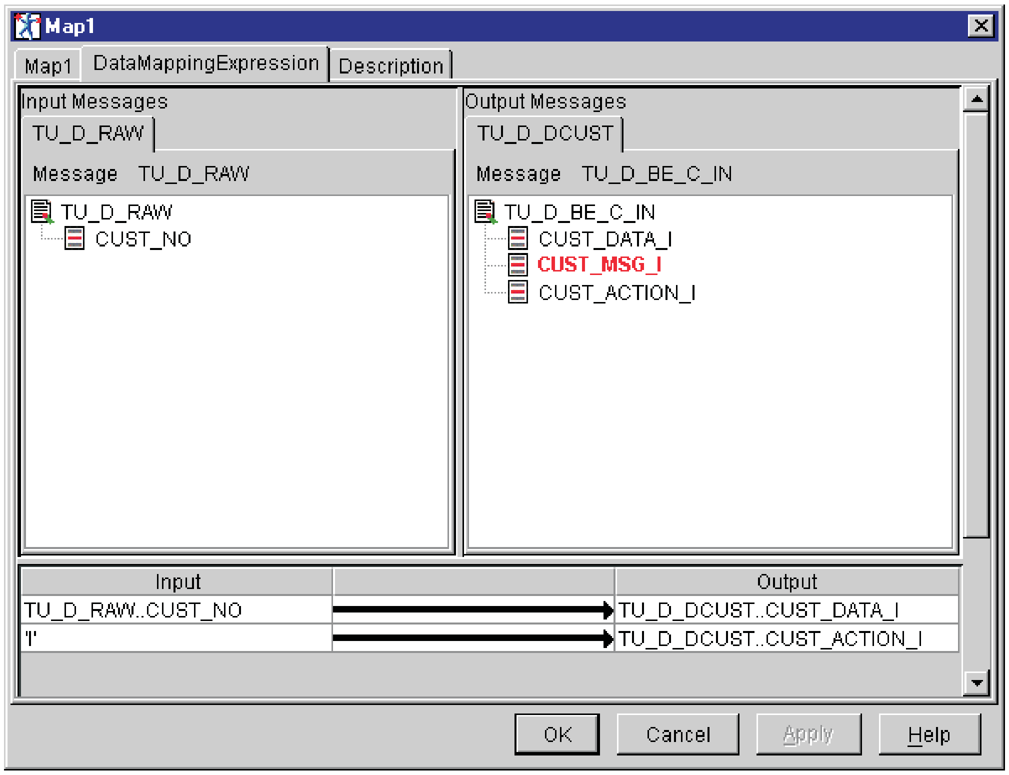
   i. Right click on the destination field for the literal (the CUST_ACTION_I field in the TU_D_DCUST Output Message) and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

   ii. Double click on LITERAL field and rename it to ’I’ (quotes must be used). Click **OK**.

d) Click **OK** when the mappings are completed.

Table 11. Mapping fields for Map1 node (TU_D_RAW message to TU_D_BE_C_IN message)

Input Field	Output Field	Description
CUST_DATA	CUST_DATA_I	Used to pass customer data through the flow
’I’	CUST_ACTION_I	Type of action — Inquiry



Input		Output
TU_D_RAW..CUST_NO	→	TU_D_DCUST..CUST_DATA_I
'I'	→	TU_D_DCUST..CUST_ACTION_I

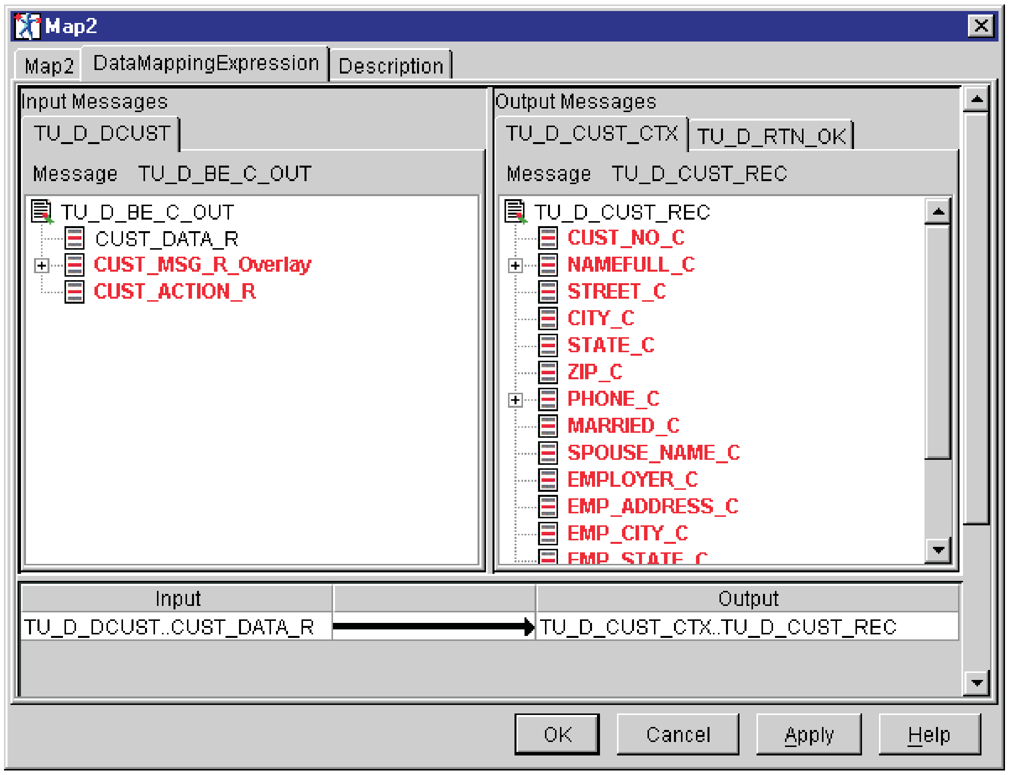
Figure 32. Mapping for Map1 node

2) Perform the mapping for the Map2 node as listed in Table 12 and Table 13 on page 51. The mappings are shown in Figure 33 on page 51 an Figure 34 on page 52.

This map passes the customer data from the DPL program DFHMABP4 (Back-end DPL Customer Information Maintenance test transaction) to the TU_D_RTN_OK Decision node. Click **OK** when the mappings are completed.

Table 12. Mapping fields for Map2 node (TU_D_BE_C_OUT message to TU_D_CUST_REC message)

Input Field	Output Field	Description
CUST_DATA_R	TU_D_CUST_REC	Store customer data received from the back-end system in a data context



Input Messages

TU_D_DCUST

Message TU_D_BE_C_OUT

- TU_D_BE_C_OUT
  - CUST_DATA_R
  - CUST_MSG_R_Overlay
  - CUST_ACTION_R

Output Messages

TU_D_CUST_CTX | TU_D_RTN_OK

Message TU_D_CUST_REC

- TU_D_CUST_REC
  - CUST_NO_C
  - NAMEFULL_C
  - STREET_C
  - CITY_C
  - STATE_C
  - ZIP_C
  - PHONE_C
  - MARRIED_C
  - SPOUSE_NAME_C
  - EMPLOYER_C
  - EMP_ADDRESS_C
  - EMP_CITY_C
  - EMP_STATE_C

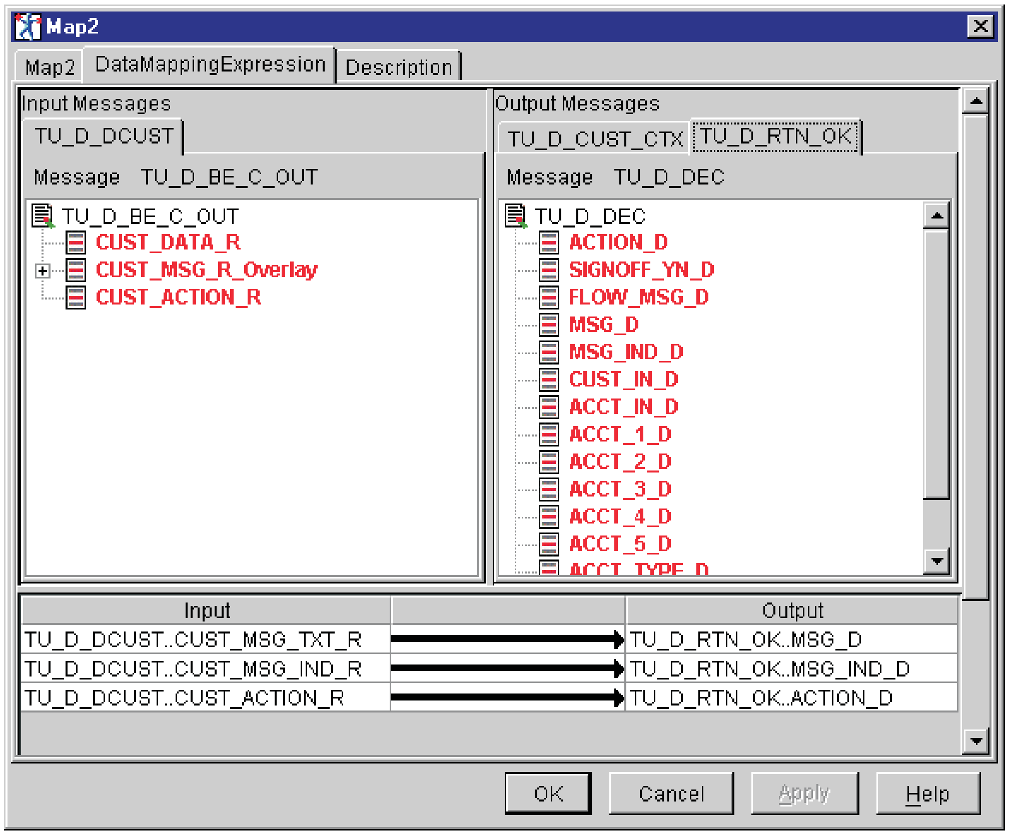
Input		Output
TU_D_DCUST..CUST_DATA_R	→	TU_D_CUST_CTX..TU_D_CUST_REC

Figure 33. Mapping for Map2 node

Note: To perform the mapping in Table 13, you will have to select the TU_D_RTN_OK tab in the Output Messages section of the Map2 node. You may have to expand the TU_D_BE_C_OUT and the TU_D_DEC records to access the underlying fields for mapping.

Table 13. Mapping fields for Map2 node (TU_D_BE_C_OUT message to TU_D_DEC message)

Input Field	Output Field	Description
CUST_MSG_TXT_R	MSG_D	Message ID number
CUST_MSG_IND_R	MSG_IND_D	Output message
CUST_ACTION_R	ACTION_D	Requested action



Input		Output
TU_D_DCUST..CUST_MSG_TXT_R	→	TU_D_RTN_OK..MSG_D
TU_D_DCUST..CUST_MSG_IND_R	→	TU_D_RTN_OK..MSG_IND_D
TU_D_DCUST..CUST_ACTION_R	→	TU_D_RTN_OK..ACTION_D

Figure 34. Mapping for Map2 node

3) Perform the mapping for Map3 node as listed in Table 14 and Table 15 and shown in Figure 35 on page 53. This map passes customer data and good response messages to the DPL Good response Output Terminal (OUT_OK). Click **OK** when the mappings are completed.

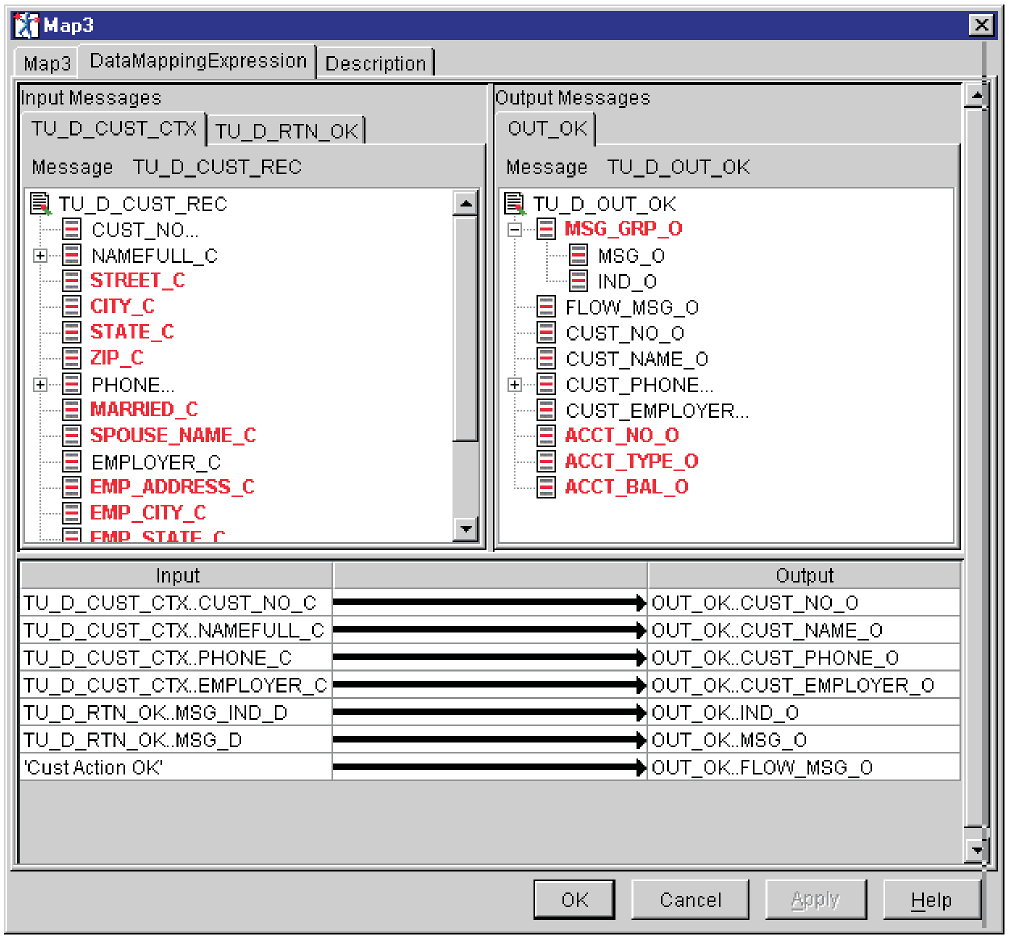
Table 14. Mapping fields for Map3 node (TU_D_CUST_REC message to OUT_OK message)

Input Field	Output Field	Description
CUST_NO_C	CUST_NO_O	Customer ID number
NAMEFULL_C	CUST_NAME_O	Customer name
PHONE_C	CUST_PHONE_O	Customer phone number
EMPLOYER_C	CUST_EMPLOYER_O	Customer employer

Table 15. Mapping fields for Map3 node (TU_D_RTN_OK message to OUT_OK message)

Input Field	Output Field	Description
MSG_D	MSG_O	Message ID number
MSG_IND_D	IND_O	Output OK message
’Cust Action OK’	FLOW_MSG_O	Response message for customer action

Note: The MSG_O and IND_O output fields can be located by expanding the MSG_GRP_O element. Click on the + sign.



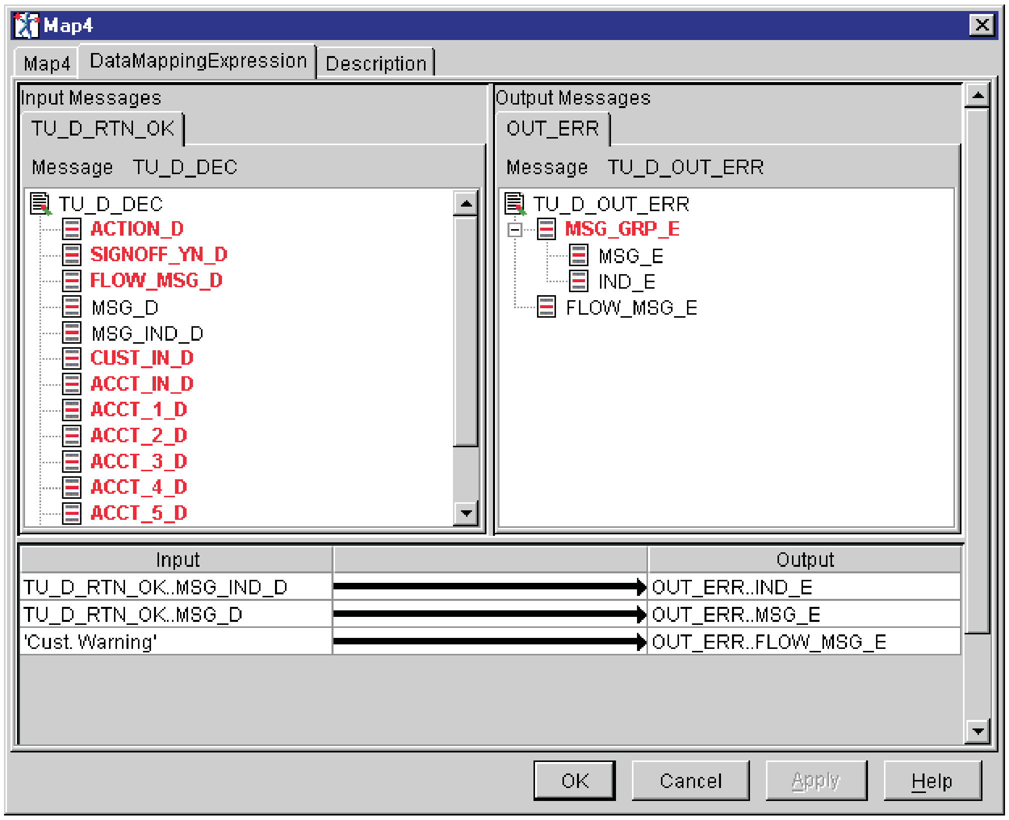
Input		Output
TU_D_CUST_CTX..CUST_NO_C	→	OUT_OK..CUST_NO_O
TU_D_CUST_CTX..NAMEFULL_C	→	OUT_OK..CUST_NAME_O
TU_D_CUST_CTX..PHONE_C	→	OUT_OK..CUST_PHONE_O
TU_D_CUST_CTX..EMPLOYER_C	→	OUT_OK..CUST_EMPLOYER_O
TU_D_RTN_OK..MSG_IND_D	→	OUT_OK..IND_O
TU_D_RTN_OK..MSG_D	→	OUT_OK..MSG_O
'Cust Action OK'	→	OUT_OK..FLOW_MSG_O

Figure 35. Mapping for Map3 node (TU_D_CUST_CTX and TU_D_RTN_OK messages to OUT_OK message)

4) Perform the mapping for Map4 node as listed in Table 16 and shown in Figure 36 on page 54. This map passes warning response messages to the DPL Error response Output Terminal (OUT_ERR). Click **OK** when the mappings are completed.

Table 16. Mapping fields for Map4 node (TU_D_DEC message to TU_D_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
'Cust. Warning'	FLOW_MSG_E	Warning message text



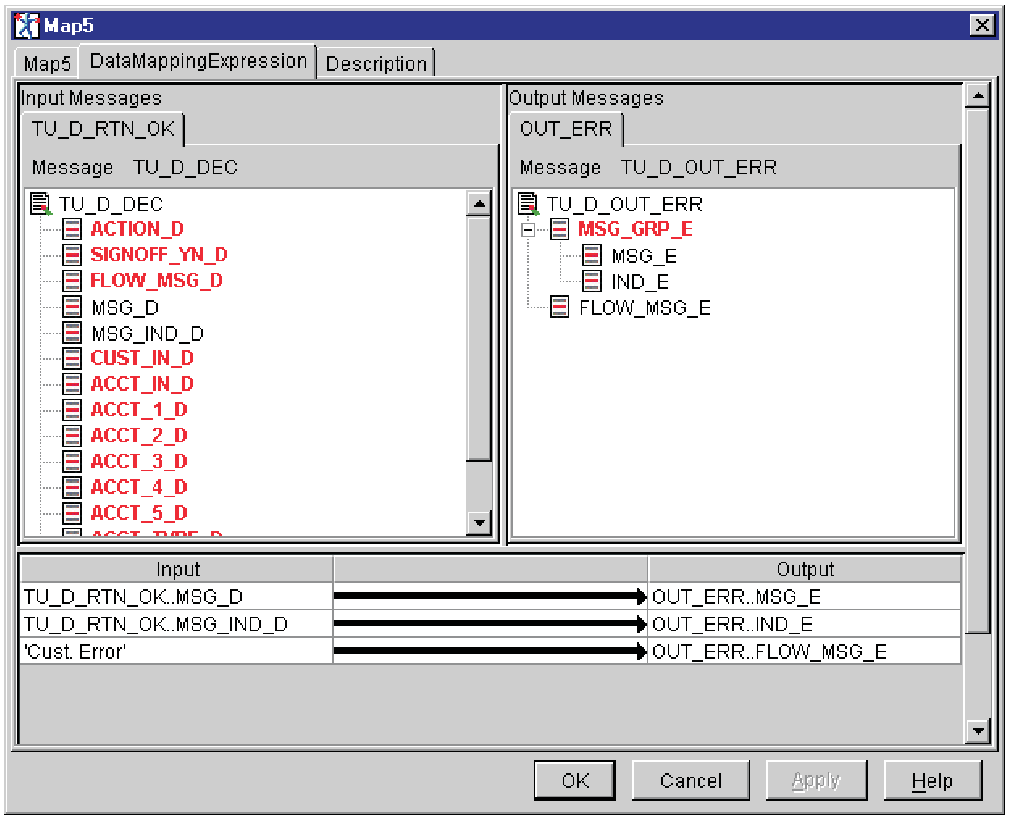
Input		Output
TU_D_RTN_OK..MSG_IND_D	→	OUT_ERR..IND_E
TU_D_RTN_OK..MSG_D	→	OUT_ERR..MSG_E
'Cust. Warning'	→	OUT_ERR..FLOW_MSG_E

Figure 36. Mapping for Map4 node

5) Perform the mapping for Map5 node as listed in Table 17 and shown in Figure 37 on page 55. This map passes error response messages to the DPL Error response Output Terminal (OUT_ERR). Click **OK** when the mappings are completed.

Table 17. Mapping fields for Map5 node (TU_D_DEC message to TU_D_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
'Cust. Error'	FLOW_MSG_E	Error message text



Input		Output
TU_D_RTN_OK..MSG_D	→	OUT_ERR..MSG_E
TU_D_RTN_OK..MSG_IND_D	→	OUT_ERR..IND_E
'Cust. Error'	→	OUT_ERR..FLOW_MSG_E

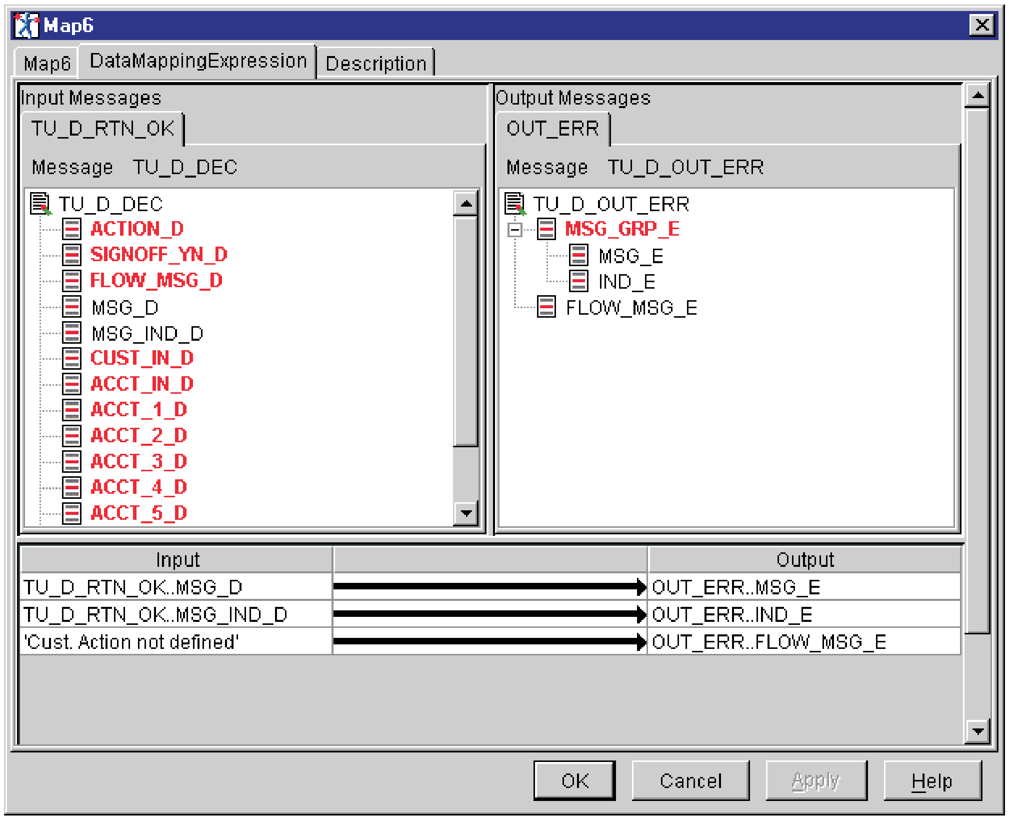
Figure 37. Mapping for Map5 node

6) Perform the mapping for Map6 node as listed in Table 18 and shown in Figure 38 on page 56. This map passes error response messages to the DPL Error response Output Terminal (OUT_ERR).

Table 18. Mapping fields for Map6 node (TU_D_DEC message to TU_D_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
'Cust. Action not defined'	FLOW_MSG_E	Error message text

Build an adapter that supports a DPL interface



Input		Output
TU_D_RTN_OK..MSG_D	→	OUT_ERR..MSG_E
TU_D_RTN_OK..MSG_IND_D	→	OUT_ERR..IND_E
'Cust. Action not defined'	→	OUT_ERR..FLOW_MSG_E

Figure 38. Mapping for Map6 node

You just completed the modelling stage in the process of building your adapter.

In your model, you have coded the instructions on how the adapter is supposed to behave at run time. You are now ready to create the adapter.

__ Step 7. **Assign the model to a CICS MQAdapter.**

In this step you will associate the microflow (the model that you just completed), with a CICS MQAdapter.

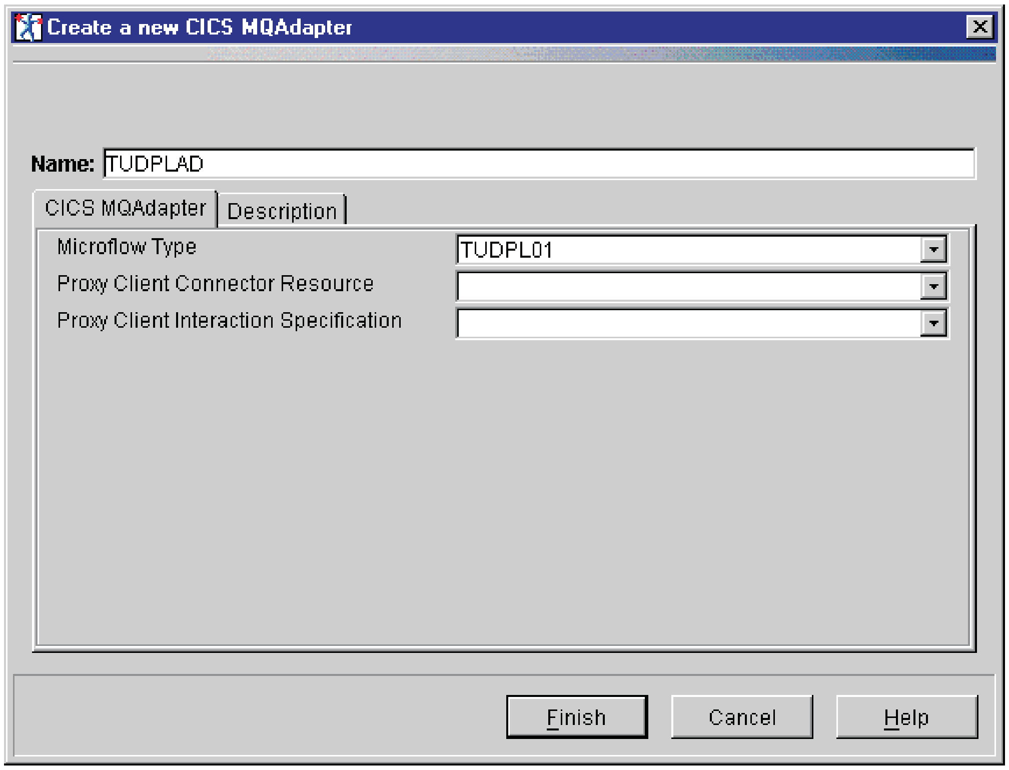
The CICS MQAdapter provides the actual implementation of the adapter request processing.

The adapter will enable the controlling application by invoking TUDNAV1 and TUDPL1 to access the back-end DPL Program (DFHMABP4) to retrieve customer information.

a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

b. On the Create a new CICS MQAdapter dialog, enter TUDPLAD for the Name and use the drop down menu to select TUDPL01 for the

Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TUDPLAD

Microflow Type: TUDPL01

Proxy Client Connector Resource:

Proxy Client Interaction Specification:

Figure 39. Creating an CICS MQAdapter

You have completed the microflow and setup your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

__ Step 8. **Generate the adapter**

Now you are ready to generate your adapter. The adapter code files will be generated in the output directory that you specify.

Adapter code generation is a two-step process:

a. Generate copybooks from message definitions (in Message Sets view).

b. Generate the adapter run time code from the modeled microflow (in Adapters view).

- **Generate the Copybooks**.

  You will generate to COBOL copybooks for the following messages:

  - TU_D_RAW
  - TU_D_DEC
  - TU_D_OUT_OK
  - TU_D_OUT_ERR
  - TU_D_CUST_REC
  - TU_D_BE_C_IN
  - TU_D_BE_C_OUT

Note: All must be generated to the same directory.

Note: To generate a copybook for a message, the message must be checked out or newly created.

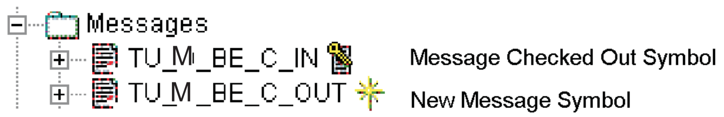


Figure 40. Messages Sets folder showing checked out message and newly created message

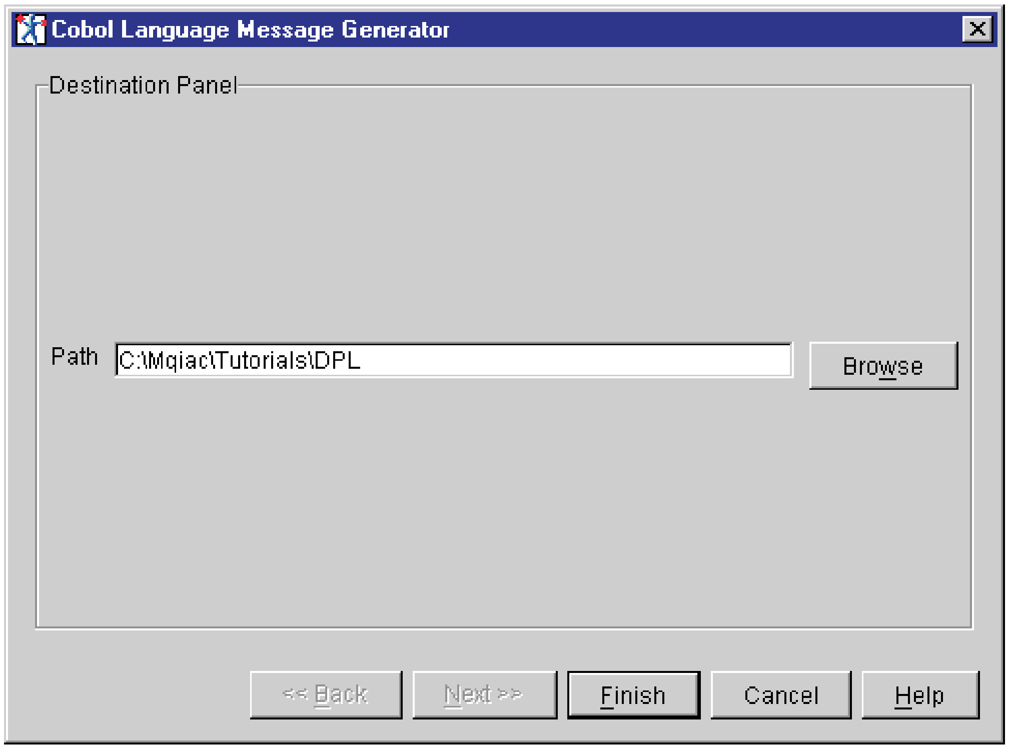
To generate copybooks, make sure that you are in the Message Sets view and then, follow this procedure:

a. Make sure the list of messages is visible under the Messages folder for the TU_D_MESSAGE_SET. To view the messages, click on the + sign in front of the Messages folder to display the list of messages.

b. Right click on the message for which you want to generate a copybook (for example, TU_D_RAW) and select **Generate > COBOL**.

c. Enter the output destination in the Path field and click Finish.

CAUTION:
The copybook generate removes underscores from the message names and only uses the first eight characters of the filename to generate the new copybook name. Be aware of potential naming conflicts.



Path C:\Mqiac\Tutorials\DPL

Figure 41. Specifying pathname for copybook generation output

d. Repeat the process to generate copybooks for the remaining messages in the list.

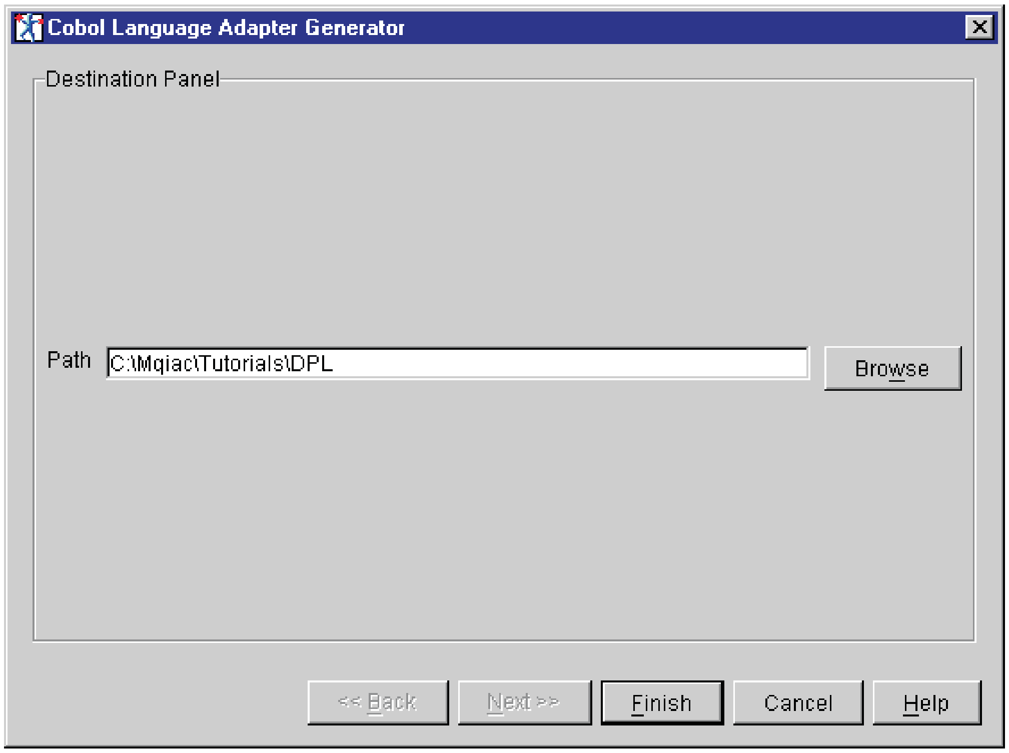
- **Generate adapter Code**.

  To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

  Note: You must generate the adapter code in the same directory where you generated the copybooks.

  a. Right click on TUDPLAD adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination in the PATH field (the example uses C:\Mqiac\Tutorials\DPL). Click **Finish**.

     The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\Tutorials\DPL

Figure 42. Specifying pathname for adapter code generation output

Deploying an adapter

GOAL

In the following section you will learn how to deploy the adapter that you created. The deploy operation sends the copybooks, source code, JCL and the configuration parameters for each microflow that you generated, to the host system, for source code configuration, object code build and parameter update operations.

Build an adapter that supports a DPL interface

You will need an account and password to the OS/390 environment that will host the adapter you are deploying.

Make sure that you have customized the build time JCL templates to your site standards. See "Building adapters" on page 6 for information on the JCL you need to customize.

You must have Object REXX installed on the workstation for FTP deployment processing.

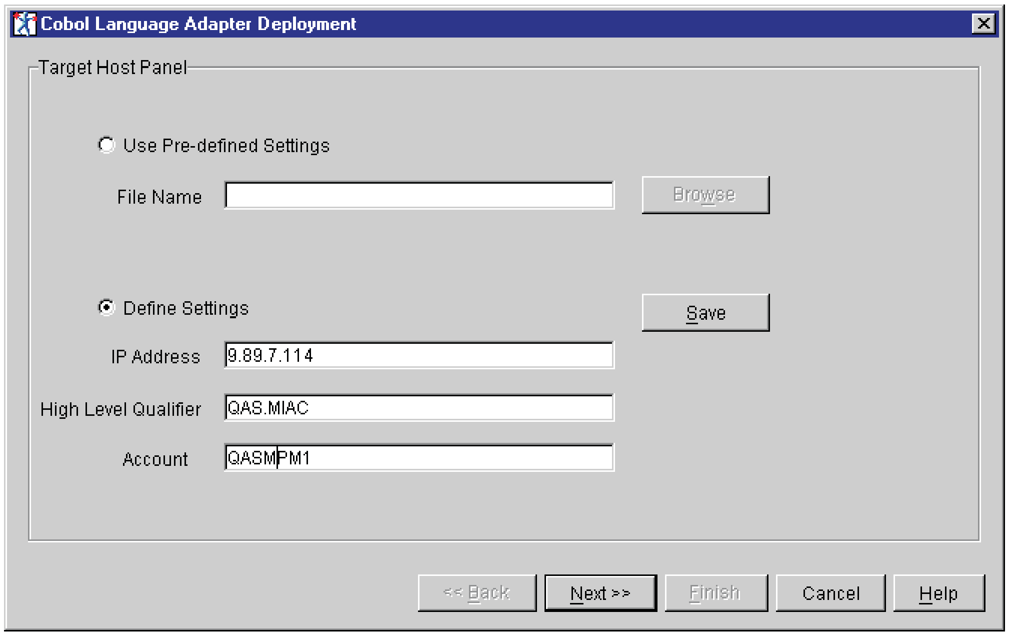
To deploy an adapter, make sure that you are in the Adapters view and then, follow this procedure:

1. Right click on TUDPLAD adapter (listed under the CICS MQAdapters folder) and select **Generate > Deploy COBOL Adapter**. Click the Define Settings radio button and enter the following information:
   - IP Address — IP Address - The host system IP address (for example, 9.89.7.114)
   - High Level Qualifier — The high level qualifier for the partition data set (PDS)

     Note: The tutorial uses QAS.MIAC as the high level qualifier.
   - Account — The account under which JCL submits a job for compilation.

   Note: If you wish to save these settings for reuse, then click **Save**. You will be prompted to specify an output location and filename to store the setting information. The next time you deploy adapter code you can click the **Use Pre-defined Settings** radio button and enter the saved filename.

   Click **Next**.


Cobol Language Adapter Deployment

Target Host Panel

Use Pre-defined Settings [ ]

File Name:

[✓ Define Settings]

IP Address: 9.89.7.114

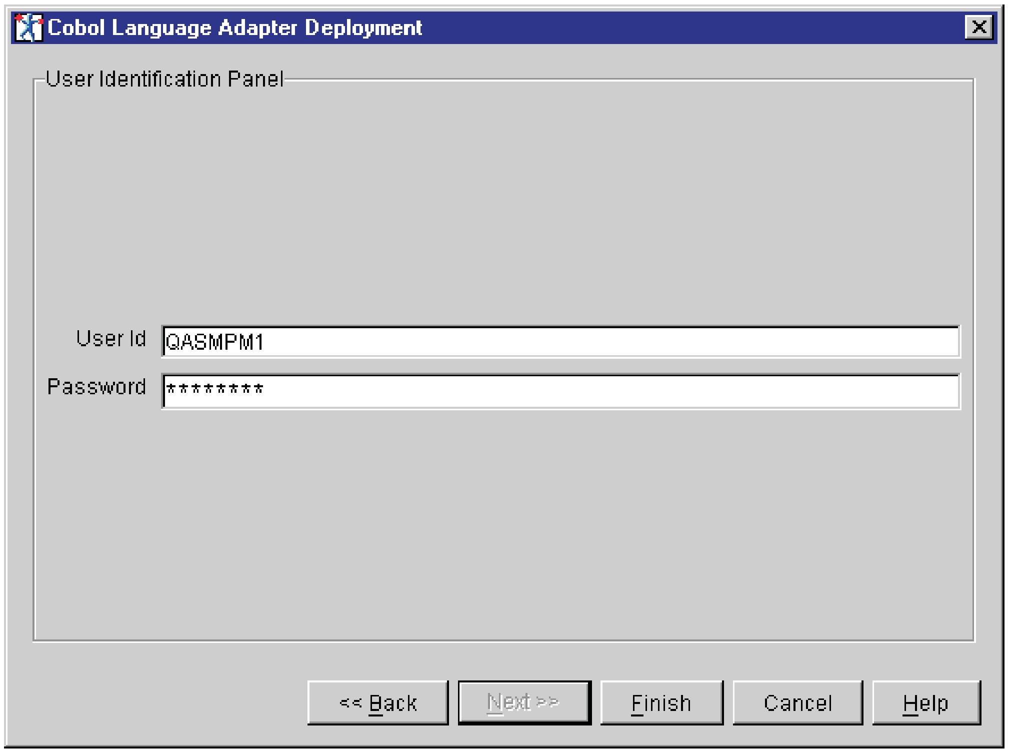
High Level Qualifier: QAS.MIAC

Account: QASMPM1

Figure 43. Specifying the target host

2. On the User Identification panel enter your user ID and password. Click **Finish**.

The generated adapter code, copybooks, and JCL (Compile / Properties File Update) files will be moved to the OS/390 server.



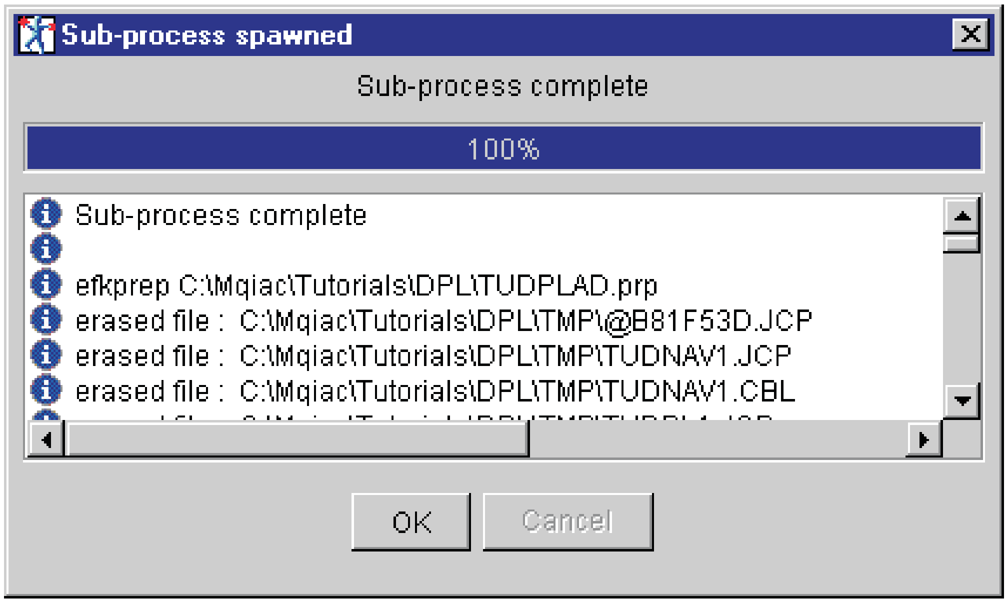
User Id QASMPM1

Password ********

Figure 44. Logon to the host

3. The *Sub-process dialog* appears and provides a status of the deploy process as it happens. When the deploy is complete the generated adapter code, copybooks, and JCL (Compile / Properties File Update) files will be moved to the OS/390 server.

   Note: You should scroll through the output listing in the Sub-process dialog window to see if any errors occurred.



Sub-process complete

100%

```
Sub-process complete

efkprep C:\Mqiac\Tutorials\DPL\TUDPLAD.prp
erased file :  C:\Mqiac\Tutorials\DPL\TMP\@B81F53D.JCP
erased file :  C:\Mqiac\Tutorials\DPL\TMP\TUDNAV1.JCP
erased file :  C:\Mqiac\Tutorials\DPL\TMP\TUDNAV1.CBL
```

Figure 45. Sub-process dialog indicating status of the deploy process

4. Select **OK** to close the dialog.

The adapter now resides on the OS/390 server and is ready to be tested. See Chapter 6, "Validating the adapters" on page 231 for instructions on how to test the adapter.

Check to see that the adapter compiled in CICS

After you have deployed the adapter to the OS/390 server, you need to make sure that it compiled with no errors. Consult with your CICS systems administrator for assistance with this procedure.

Defining the adapter resources to CICS

If you do not have access to CICS at your site, you will need to ask your CICS administrator to perform the necessary CEDA and CEMT functions. You will need to provide the CICS administrator with the following information as it relates to the adapter that you deployed:

- Program names
- Group name
- Transaction Identifiers

For the DPL adapter, the following values apply:

Table 19. Values for the Define Transactions screen

Program	Group	Transid
TUNAV1	MIACUSER	TUDN
TUDPL1	MIACUSER	TUD1

A CICS administrator needs to define the adapter resources to CICS each time a new adapter is deployed. CICS needs to know which resources to use, what their properties are and how they interact with other resources.

To define resources to CICS, the CICS administrator must:

- Run the CEDA transaction to define programs and any files to CICS.
- Submit JCL to run the Properties File Update job.

  This is necessary only if you did not automatically submit JCL using the builder's generator facility.

  If you were not allowed to submit JCL automatically, you can manually submit JCL (DFHMAMPU) to run the Properties File Update job (DFHMAMUP). See the MQSI Agent for CICS Run Time User's Guide for information on the Properties file update JCL (DFHMAMPU).

The CICS administrator must NEWCOPY any server adapter programs that were modified.

For an example of defining CICS resources to CICS, See "Example procedure for defining adapter resources to CICS" on page 239.

Chapter 4. Build an adapter that supports an MQ interface

Before you begin this tutorial, read Chapter 2, "Tutorial overview" on page 9. The Tutorial Overview section contains important information on the business transaction to be modeled, as well as information on the tutorial's file structure. The Tutorial Overview also lists the assumed environment requirements that must be adhered to in order to build and deploy the adapter.

From this tutorial, you will learn how to use the MQSI Agent for CICS Adapter Builder tool to model and generate code for an adapter that supports an MQ interface.

You will model an adapter that has the functionality to access an existing CICS program via an MQ Interface. See "About the adapter that you will design" on page 66for a description of the adapter that you will model.

If you have not had any exposure to the Adapter Builder component of the MQSeries Integrator Agent for CICS Transaction Server product, you should read the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center*. This book contains information on the concepts of the MQSeries Integrator Agent for CICS Adapter Builder.

This tutorial consists of:
- "Designing an adapter"
- "Creating an adapter that supports an MQ interface" on page 76
- "Deploying an adapter" on page 107

After completing this tutorial you should be able to:
- Identify required Host based information you need to gather and use.
- Import COBOL copybooks into a workspace and create message sets.
- Create workspaces to define adapter flow logic.
- Create and generate a COBOL adapter.
- Deploy and test the generated COBOL adapter.

Before you begin this tutorial you should read Chapter 2, "Tutorial overview" on page 9. The Tutorial overview provides important information on the tutorial files, the tutorial directory structure and how to avoid naming conflicts when you create message sets and messages.

Designing an adapter

As was discussed in "Requirements analysis and design considerations" on page 1, before you start to use the MQSI Agent for CICS Adapter Builder, you would spend some time analyzing the business need that the adapter will address and then spend some time considering how you will design the adapter.

Build an adapter that supports an MQ interface

When you finish with requirements analysis and design considerations, you should have a sound understanding of how your adapter will behave at run time in order to manage and fulfill a business transaction.

To help you gain a frame of reference for what you will create in this tutorial, you should understand the following:

- The business need to be addressed
- The messages in and out structure
- The CICS resources required

Addressing a business need

An adapter should address a particular business need. In this tutorial, the business need is to provide a controlling application with an interface to a back-end environment for the purpose of accessing an existing CICS application (named DFHMABP6) that performs a customer inquiry.

In this tutorial you will be accessing the same back-end environment that was installed and used by the run time installation verification procedure (IVP). For information on the programs used by the IVP, see the chapter on performing post installation tasks in the MQSI Agent for CICS Run Time User's Guide.

Note: For the purpose of this tutorial, the back-end environment that you will be accessing is the same back-end environment that was installed and used by the run time installation verification procedure (IVP). For information on the programs used by the IVP (including DFHMABP6), see the chapter on performing post installation tasks in the MQSI Agent for CICS Run Time User's Guide.

About the adapter that you will design

The adapters that you build using the MQSI Agent for CICS Adapter Builder are visual models of business transactions. They are intended to map out the activities that comprise the entire business transaction, from invocation to completion.

The adapter that you build contains the instructions, logic and code that enable it to run on an OS/390 server, this includes an interface methodology for accessing information on back-end systems. In this tutorial the business transaction on which you will base your adapter is a *customer inquiry request* and the interface method used is an MQ interface.

Your adapter design will include instructions on accessing a back-end application to retrieve customer information and will include instructions on where to put the information so that it can be returned to the controlling application.

In this tutorial you will learn how to design and build an adapter that when deployed will perform the following functions:

1. Accept the structure TU_M_RAW from the Simulator. TU_M_RAW is the input record description from the controlling application.
2. Map individual fields from TU_M_RAW to the expected and required back-end program commarea format.
3. After completion of the MQ Put function, map response message from MQ Get program.
4. Determine success - map output message (OUT_OK or OUT_ERR) to be returned to the Simulator and exit.

Identify the components of the run time environment

Before building the adapter we need to:

- Define the CICS transaction IDs for the adapter programs that are generated.

  Note: As with any CICS application, the program names, transaction IDs, and CICS region must adhere to your local site standards and your local sites naming conventions.

Table 20. MQ Adapter programs

Program type	Program name	Transaction ID
Navigator	TUMNAV1	TUMN
MQ Adapter — Put	TUMQ01P	TUMP
MQAdapter — Get	TUMQ01G	TUMG

- Determine the program that is invoked, as mentioned previously this is the existing CICS application named DFHMABP6. The request and response messages utilized are TU_M_BE_C_IN and TU_M_BE_C_OUT respectively.
- Determine the CICS region where the adapter programs will execute.
- Determine the MQ request and reply queues utilized for access to the back-end system.

After some analysis, we determine that adapter execution on the host environment will involve the sequence documented in Figure 46. In this host environment, the generated MQ Adapter programs, TUMNAV1, TUMQ01P and TUMQ01G execute in CICS region QAS1. DFHMABP6 acts as the back-end program. In this scenario it executes in the same CICS region (QAS1).

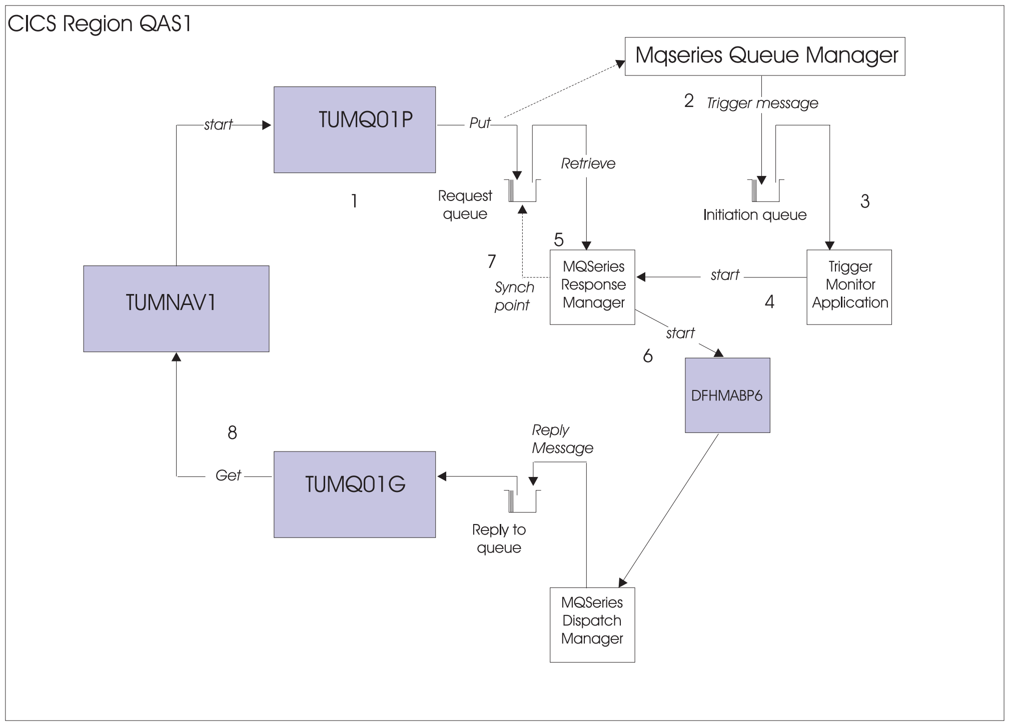


Figure 46. Tutorial run time environment for MQ adapter (DFHMABP6).

Build an adapter that supports an MQ interface

1. A customer inquiry business request invokes TUMNAV1, which in turn invokes the Put program (TUMQ01P) to *put* the request message in the Request Queue on the host.

   Note: The request queue's characteristics (that is, whether it is a first-in-first-out queue, and so on) and its message priority scheme are controlled by the configuration of the MQSeries licensed software.
2. The MQSeries Queue Manager generates a trigger event, causing a trigger message to be placed on the initiation queue associated with the Request Queue. The trigger message contains information from the associated MQSeries process definition object.
3. The trigger monitor application retrieves the trigger message from the initiation queue.
4. The trigger monitor application starts the MQSeries Response Manager.

   Note: It is also possible to start the MQSeries Response Manager as a long running process instead of having it started by a trigger, but this may result in slower performance.
5. The MQSeries Response Manager opens the Request Queue and retrieves the message.
6. The MQSeries Response Manager starts the DFHMABP6 program, passing to it the User Data from the message.
7. After successfully starting the DFHMABP6 program, the MQSeries Response Manager issues a sync point, enabling the message to be deleted from the request queue.
8. DFHMABP6 *Puts* to the reply queue, invoking the ***Get*** adapter program (TUMQ01G) that *gets* the message from the reply queue.
9. TUMNAV1 sends the reply back.

When completed, the flow of components that make up your model will look like the following:

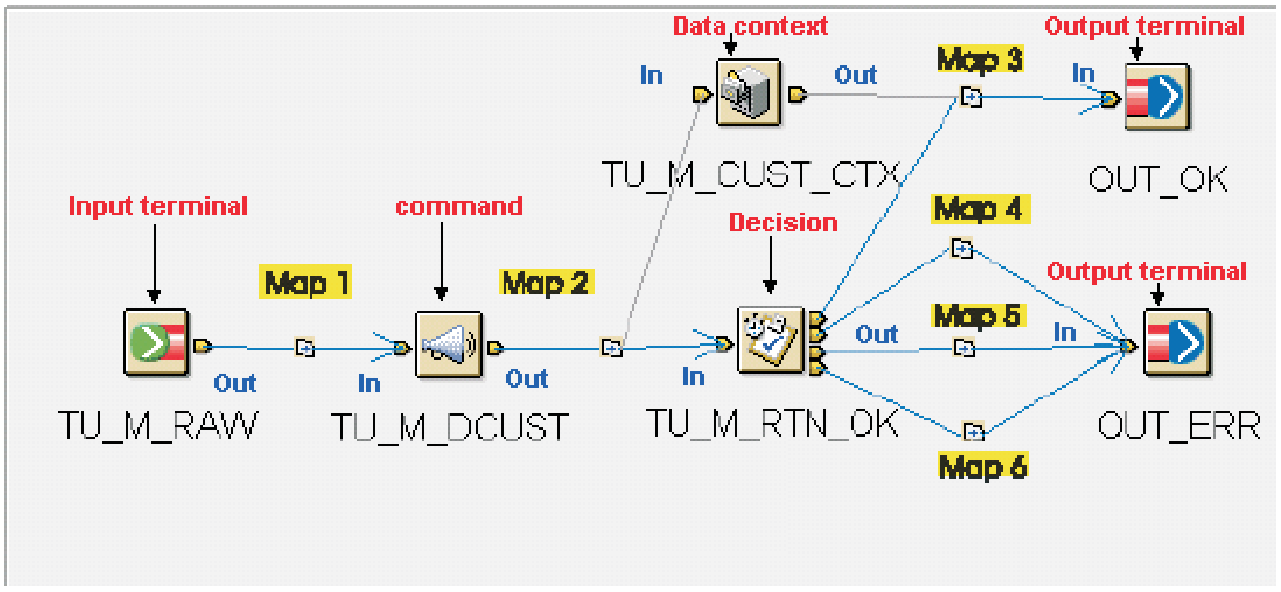


Figure 47. Components that make up the MQ adapter you will build

To better understand the role that each component plays in your model of the business transaction, see Table 21.

Use this table in conjunction with Figure 47 on page 68

Table 21. Component roles in the adapter that supports an MQ interface

Component	Name	Definition	Role / implementation
Input Terminal	TU_M_RAW	A primitive component that is used to represent data types that are input to a microflow. The term *primitive* indicates that inputs and outputs are visible to the user but their internals are not visible. TU_M_RAW contains the record description from the controlling application.	The purpose of this component within the context of modeling the transaction is to provide an entry point for the controlling application to the Navigator To implement this data transformation in your model, you will connect TU_M_RAW to the command node TU_M_DCUST by way of a **control connection**. A control connection provides the sequence relationship between two nodes in a microflow. On the Map node that sits on the control connection wire between TU_M_RAW and TU_M_DCUST, you will program the data transformation – in this case this means you will move the Customer Number provided by the Controlling application and specify the desired action to be performed by the existing CICS application DFHMABP6. By hard coding an **I** in the CUST_ACTION field, you are directing the CICS application DFHMABP6 to perform an inquiry on the customer record that correlates to the customer number.

Table 21. Component roles in the adapter that supports an MQ interface (continued)

Component	Name	Definition	Role / implementation
Command	TU_M_DCUST	A simple component that is used to represent application APIs that you import into the builder. The term *simple* indicates the component does not consist of other components.	The purpose within the context of modeling the transaction is to create an MQ Command type, which will allow the microflow to initiate the execution of the server-side DFHMABP6 program. The adapter will generate 2 COBOL programs (a GET and a PUT) for every MQ command node. The programs that are generated are the vehicle that allows the Navigator to interact with the existing CICS application DFHMABP6. On the map node that sits on the control connection wire between TU_M_DCUST and TU_M_RTN_OK, you will provide code that moves the output data (via a data control connection) provided by DFHMABP6 to a data context node named TU_M_CUST_CTX. The data context node holds data for future use. You will also provide instructions on the map node that moves the DFHMABP6 return code information to the decision node named TU_M_RTN_OK. Based on the return code provided, the decision node makes determination on the success or failure of the existing CICS application DFHMABP6.

Table 21. Component roles in the adapter that supports an MQ interface (continued)

Component	Name	Definition	Role / implementation
Decision	TU_M_RTN_OK	A composed component that is used to test a condition for true or false, to resolve the control flow path. The term *composed* indicates the component consist of other components that are connected by control flow connectors.	In your model of the business transaction, the Decision node will be used to evaluate the message indicator (Good, Warning or Error) upon return from the existing CICS application DFHMABP6. The Decision node will test the return code information that it receives from DFHMABP6. Based on the results of the test, the flow of the transaction will proceed in one of 4 ways (good, warning, error or default (unknown)) as indicated in Figure 47 on page 68. On the 3 map nodes that sit on the control connection wires between TU_M_RTN_OK and the OUT_ERR output terminal node, you will provide instructions that move the appropriate error information (depending on the error). This error information will be returned to the controlling application via the Output terminal. On the map node that sits on the control connection wire between the TU_M_RTN_OK decision node and the OUT_OK output terminal node, you will provide the instructions that move the customer demographic information (stored in TU_M_CUST_TRX data context node) along with the successful response information. This information will be returned to the controlling application via the Output terminal.

Table 21. Component roles in the adapter that supports an MQ interface (continued)

Component	Name	Definition	Role / implementation
Data Context	TU_M_CUST_TRX	A simple component that is used to store data for later access via a data connection.	You will need to create a Data Context type to store customer information. In the model, this Data Context Node is used to store the contents of the customer demographic information fields populated by the CICS application DFHMABP6. This data will then be provided to the controlling application or discarded once the Navigator determines the success or failure of the MQ Adapters.
Output	OUT_OK	A primitive component that is used to represent data types that are output from a microflow.	The purpose of this component within the context of modeling the business transaction is to provide an exit point for the controlling application from the Navigator. In this model, the controlling application has been designed to receive 2 different types of reply messages. A successful reply and an error reply.
Output	OUT_ERR		

Accessing the MQ tutorial files

The files you will need in order to build and deploy an adapter that supports an MQ interface are located in two directories as follows:

- C:\<mqiac_tutorials>\mq
- C:\<mqiac_base>\cics

In the **C:\<mqiac_tutorials>mq** directory you will find the following files:

Table 22. Files to be used in the MQ tutorial

File name	Description	Use
TU_M_RDS.cbl	COBOL record description.	Used as import for messages. Contains message structures.
TC_MQ_WS.zip	Completed workspace for the MQ adapter.	A completed workspace that you import and use as the basis for the workspace used to create the MQ adapter. See “Accessing a completed workspace” on page 12 for information on using the contents of this file
*.cpy	Generated copybooks	The generated copybooks for the MQ adapter

In the **C:\<mqiac_base>\cics** directory you will find the following files:

Table 23. Files in the C:\<mqiac_base>\cics directory

File name	Description	Use
tu_m_mq1.ispec	Interaction specification file (tutorial version)	Assigns MQ command type, COBOL generated program name, and CICS TransID.
tu_m_mq1.rsc	Connector resource file (tutorial version)	Specifies the MQ message maximum length, MQ message type, Queue Manager name, MQ Request Queue name, MQ Reply Queue name and wait interval.
tu_m_nav1.rsc	Connector resource file (tutorial version)	Specifies synchronous rollback, Navigator type, COBOL program name for the MQ adapter and the CICS TransID.

Note: There is also a version of the Specification files prefixed with *tc_m_* that are used for the completed workspace supplied in the TC_MQ_WS.zip file.

Configuring the Specification files for an MQ interface

In this section you will learn how to configure the physical properties of MQ adapter. These properties represent the XML definitions that are sent to the Properties file on the host at deployment time.

For information on the Properties file, see the MQSI Agent for CICS run time documentation.

Specification files are XML-format files that provide specific values to certain components created in MQSI Agent for CICS. An *Interaction Specification* file provides unique values for the component to which it is assigned. A *Connector Resource* file provides more general values for the component.

Some of the information in the Interaction Specification file and Connector Resource file maps to a run time properties file, DFHMAMPF. Other information in the Interaction Specification file is incorporated in generated Command and Navigator programs. The DFHMAMPF file stores data that is needed to run the generated adapter code programs on the host.

The MQ adapter requires specification files for its Command type and its Microflow type. The specification files are located in the `<mqiac_base>/cics` directory.

Build an adapter that supports an MQ interface

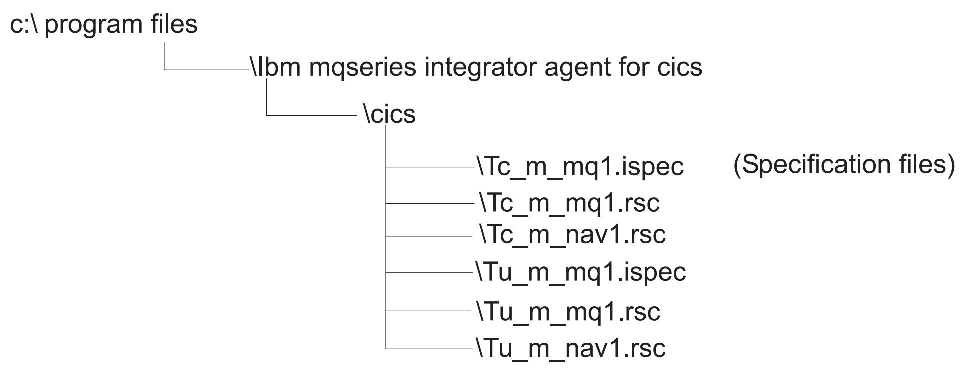


Figure 48. Directory structure for locating specification files for the MQ interface

You **must** configure the settings in the specification files used for the tutorial. The MQ Command type uses a Connector Resource file and an Interaction Specification file. In the Connector Resource file (**`tu_m_mq1.rsc`**), you need to define the **tu_m_nav1.rsc** file.

The Interaction Specification file for the MQ Command type used in the tutorial is **`tu_m_mq1.ispec`**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Interaction Specification">
   <Attribute xmi.label="MAT_CMDTYPE" type="MAT_DPL MAT_MQ MAT_FEPI" xmi.uuid=""
         valueMandatory="true" value="MAT_MQ" encoded="false"/>
   <Attribute xmi.label="MAT_PROGID" type="" xmi.uuid="" valueMandatory="false"
         value="TUMQ01" encoded="false"/>
   <Attribute xmi.label="MAT_TRANID" type="" xmi.uuid="" valueMandatory="false"
         value="TUM" encoded="false"/>
</AttributeGroup>
```

Table 24. Keyword values used for MQ Interaction Specification file

Keyword Symbolic	Description / Use	Example Value
MAT_CMDTYPE	Identifies the type of command	MAT_MQ
MAT_PROGID	The name of the COBOL program generated for the MQ command.	TUMQ01
MAT_TRANID	The CICS TransID for the server command program generated on the server.	TUM TUM (generated as TUMP and TUMG)
Note: Two sets of programs (Put and Get) and TransIDs are generated for the MQ interface adapter. For MQ type commands, TransIDs are truncated to the first three characters and a ″P″ or ″G″ is appended for the transaction id of the generated Put and Get programs, respectively. For MQ type Commands, Program IDs are truncated to the first seven characters and a ″P″ or ″G″ will be appended for the Program ID of the generated Put and Get programs, respectively. There are instances where only a Put program is generated, for example, when no reply is required. This scenario is controlled by the MAT_MQMSGTYPE field in the Resource Connection file. A value of 8 in MAT_MQMSGTYPE field indicates that the message does not require a reply.		

The Resource Connection file for the MQ Microflow type used in the tutorial is `tu_m_mq1.rsc`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Connector Resource">
   <Attribute xmi.label="MAT_REQUEST_QNAME" type="" xmi.uuid="" valueMandatory="false"
         value="QAS1.SAMP.CMAY.REQUEST.QUEUE" encoded="false"/>
   <Attribute xmi.label="MAT_REPLY_QNAME" type="" xmi.uuid="" valueMandatory="false"
         value="QAS1.SAMP.CMAY.REPLY.QUEUE" encoded="false"/>
   <Attribute xmi.label="MAT_REPLY_QMGR" type="" xmi.uuid="" valueMandatory="false"
         value="""" encoded="false"/>
   <Attribute xmi.label="MAT_MAXOUTMSGLEN" type="" xmi.uuid="" valueMandatory="false"
         value="401" encoded="false"/>
   <Attribute xmi.label="MAT_WAIT_INTERVAL" type="" xmi.uuid="" valueMandatory="false"
         value=""030"" encoded="false"/>
   <Attribute xmi.label="MAT_MQMSGTYPE" type="" xmi.uuid="" valueMandatory="false"
         value=""1"" encoded="false"/>
</AttributeGroup>
```

Table 25. Keyword values used for MQ Microflow Connector Resource file

Keyword Symbolic	Description / Use	Example Value
MAT_REQUEST_QNAME	The request queue definition identifies the queue on which the server adapter will put the request message for an MQ back-end transaction.	QAS1.SAMP.CMAY.REQUEST.QUEUE
MAT_REPLY_QNAME	The reply to queue definition identifies the queue from which the server adapter will get a reply message from the back-end transaction.	QAS1.SAMP.CMAY.REPLY.QUEUE
MAT_REPLY_QMGR	This is the name of the queue manager to which the MQSI Agent for CICS run time should send reply messages.	Blank value indicates the use of the default Queue Manager.
MAT_MAXOUTMSGLEN	Specifies the maximum length of the MQ message placed on MAT_REPLY_QNAME.	401
MAT_WAIT_INTERVAL	This is a 3 byte field that indicates the approximate time, expressed in seconds, that the MQGET call waits for a suitable message to arrive.	030
MAT_MQMSGTYPE	Indicates the MQ message type of the defined request. The value is placed in the MQMD Message Descriptor structure field MsgType on the MQPUT1 call.	1 (Put followed by a waited Get)

You have just completed the steps necessary to configure the Properties file. You are ready to create the adapter that supports an MQ interface.

Now that you have configured the specification files,

Creating an adapter that supports an MQ interface

GOAL

In this section you will learn how to use the adapter builder to create the model of the business transaction.

Specifically, you will learn how to import the necessary COBOL record descriptions and system interfaces for the MQ adapter. These are stored in the logical message model in the Adapter Builder for use in the microflow.

Follow these instructions to begin the process of building an adapter that supports an MQ interface:

__ Step 1. **Start the builder and create a new workspace**.

To start the builder, go to the **Start** > **Programs** > **IBM MQSI Agent for CICS** >**IBM MQSI Agent for CICS**. This will launch the tool as shown below, in Figure 49.

You should begin the tutorial with a new workspace. A workspace is a view of what you can work with at one time. A workspace is displayed as the graphical space in the builder where you will build the adapter to support the MQ interface.

From the **File** pull-down menu, select **New Workspace**.

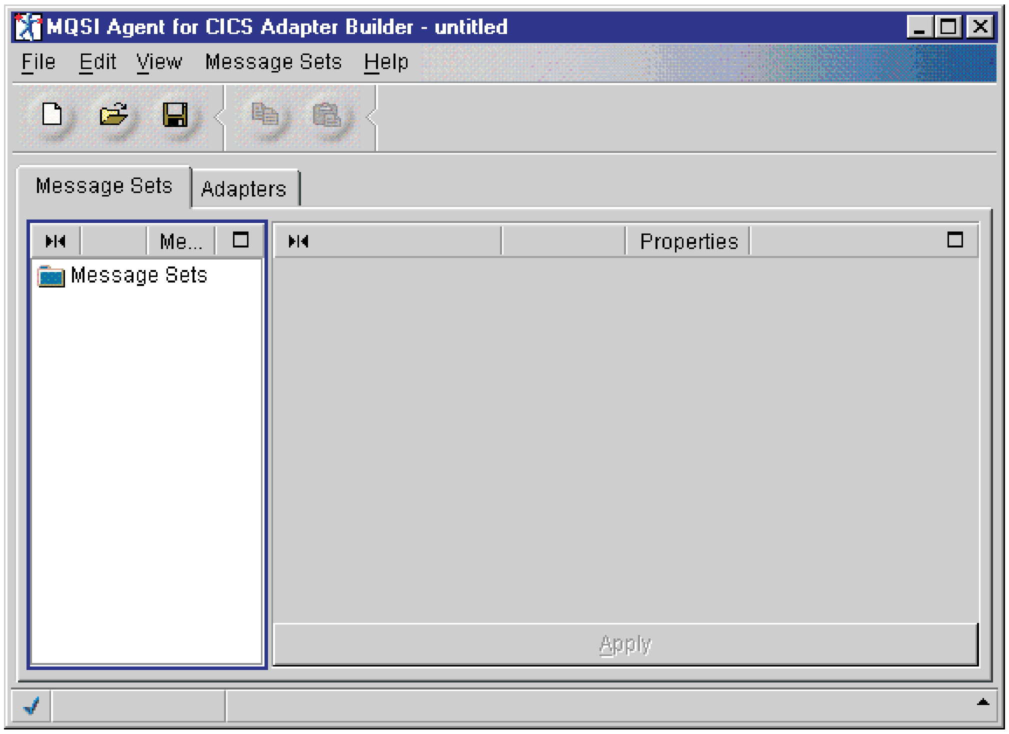

MQSI Agent for CICS Adapter Builder - untitled
File Edit View Message Sets Help
Message Sets
Adapters
Me...
Message Sets
Properties
Apply

Figure 49. Initial panel of the MQSI Agent for CICS Adapter Builder

__ Step 2. **Name your tutorial workspace and save it to the repository**.

You should begin the tutorial with a new workspace. Select **File > New Workspace**. From the **File** pull-down menu, select **Save Workspace**. Enter a name for the workspace, such as TU_MQ_WS, and click **Save**.

Note: Be sure to use under_scores and not dashes ″-″ when naming the workspace.

__ Step 3. **Import a message set**.

A message set is a collection of structured XML-based data types that are stored in the message repository.

When you import a message set, what you are really doing is bringing in the COBOL structured data type definitions from existing CICS transactions on the host system, into the Adapter Builder's control center. The imported data type definitions contain the record descriptions of the messages. The control center utilizes the message set as an interface between the adapter builder tool and the business transaction to be modelled.

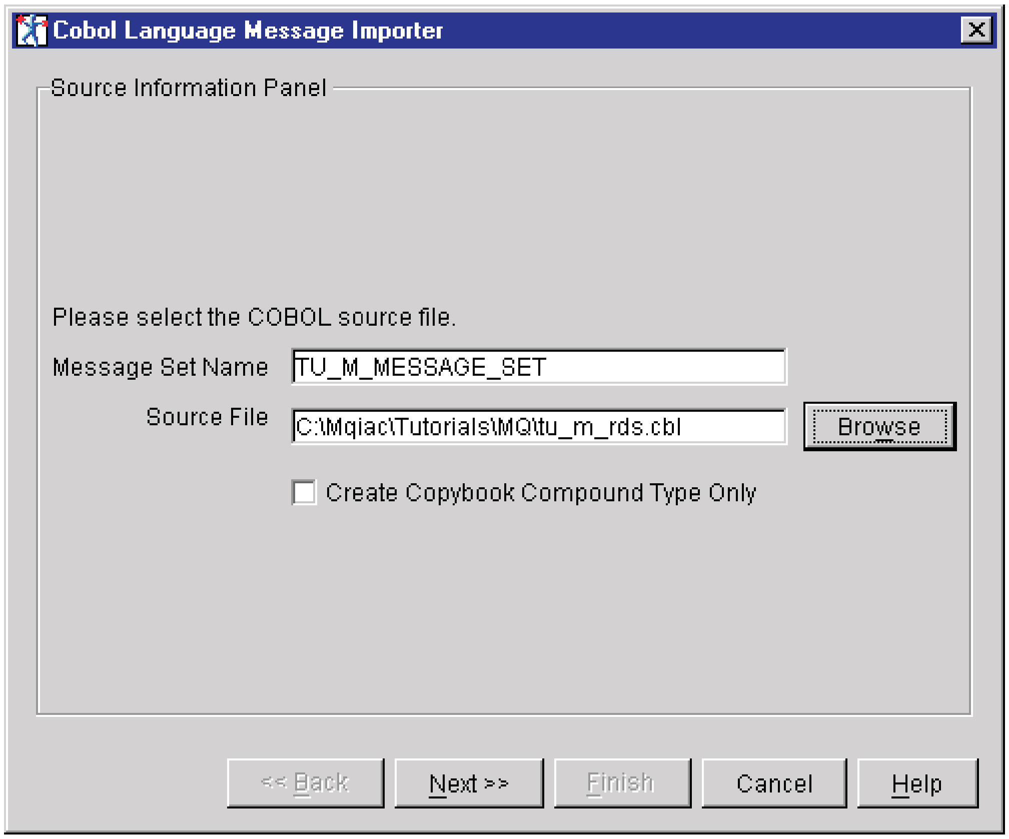
Note: You cannot import the COBOL structured data type definitions directly from CICS. You must first FTP the structured data type definitions from the host to a workstation. You can then import the message set from the workstation.

After importing a message, you can modify and store it.

Note: It is much easier to import a COBOL structured data type definition than it is to build the message set. If there is no record description, create one with a text editor and import it.

a. Right click on the Message Sets folder, select **Import to New Message Set > COBOL**. .

   On the COBOL Language Message Importer dialog (Source Information Panel), enter the Message Set Name (in the tutorial, TU_M_MESSAGE_SET) and the directory path where the Source Files for the copybooks are located (<mqiac_tutorials>\mq).



Please select the COBOL source file.

Message Set Name: TU_M_MESSAGE_SET

Source File: C:\Mqiac\Tutorials\MQ\tu_m_rds.cbl

☐ Create Copybook Compound Type Only

Figure 50. Import a message set (source information)

For the purposes of this tutorial, leave the **Create Copybook Compound Type Only** box unchecked. This box is an option that controls *how* copybooks can be imported.

Click **Next** to go to the Group Level Panel.

The radio button selections are as follows:

Request
: Use if the message is going to be used as an input message in a transaction.

Response
: Use if the message is going to be used as an output message in a transaction.

Undefined
: Can be used for messages that are not used in a transaction.

b. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import (for the tutorial, select TU_M_RAW) and select the Undefined message type radio button. Click **Finish** to complete the import.

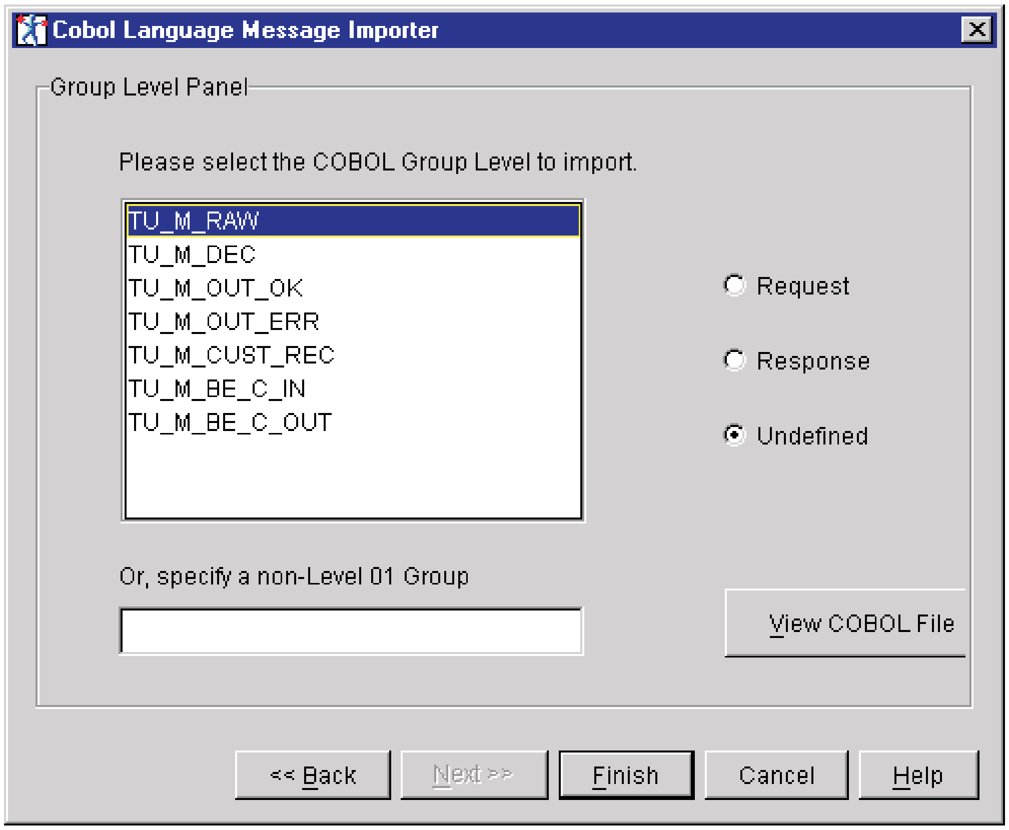


Figure 51. COBOL Language Message Importer — group level panel

c. Right click on the newly created TU_M_MESSAGE_SET folder and select **Import to Message Set > COBOL**. On the COBOL Language Message Importer dialog (Source Information Panel), enter the directory path where the Source Files for the copybooks are located (see Figure 50 on page 78). Click **Next**.

d. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import (for the tutorial, select TU_M_DEC) and select the Undefined message type radio button.

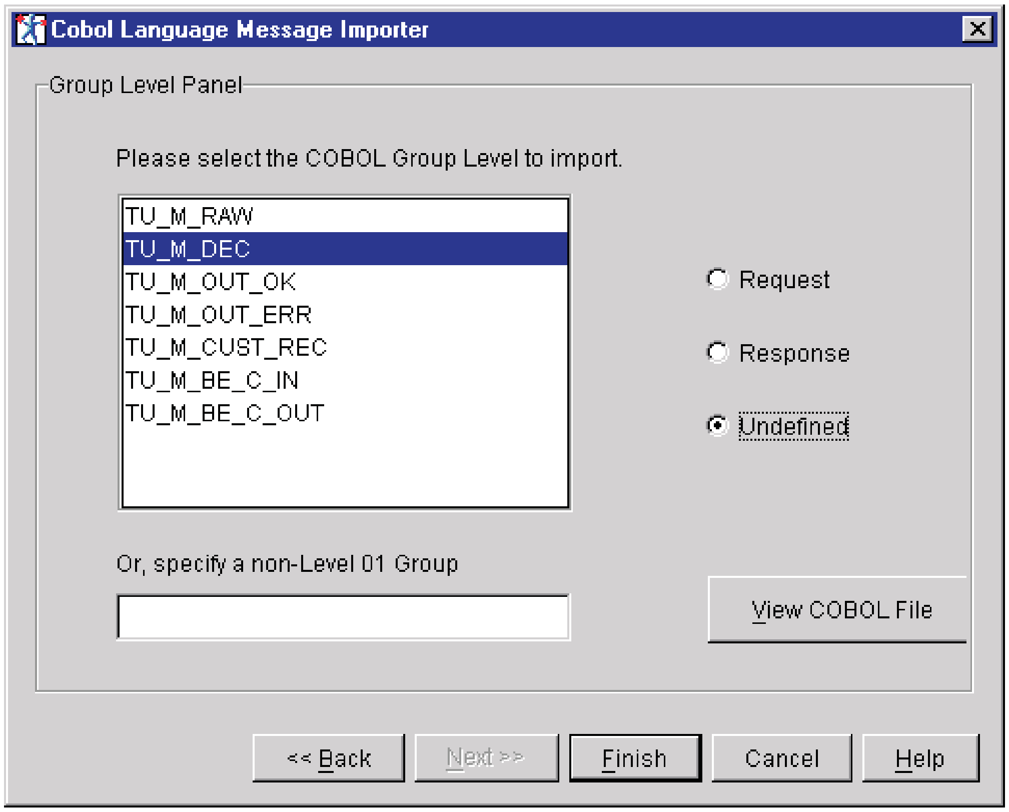


Figure 52. Import a message set (group level)

e. Click **Finish** to complete the import.

f. Click **Next** to go back to the Group level panel. Repeat the import procedure until the remaining messages (with the specified message types) are imported:

Table 26. Messages to add to the workspace

Message	Message Type	Purpose
TU_M_RAW	Undefined	Input message used by the navigator to receive information from the controlling application.
TU_M_DEC	Undefined	Decision node message used by the navigator to determine how to flow logically within the flow. This message provides a series of fields, the context of which are evaluated by the Navigator to control logical flow.

Table 26. Messages to add to the workspace (continued)

Message	Message Type	Purpose
TU_M_OUT_OK	Undefined	The output message used by the Navigator to provide customer demographic information to the controlling application in the event that the customer inquiry request was executed successfully.
TU_M_OUT_ERR	Undefined	Error output message used by the Navigator to provide error information to the controlling application in the event that the customer inquiry request was not executed successfully, due to the fact that the customer inquiry failed.
TU_M_CUST_REC	Undefined	Customer record layout utilized by the navigator to store customer demographic information supplied by the existing CICS application DFHMABP6.
TU_M_BE_C_IN	Request	Commarea Input message used to supply the existing CICS application DFHMABP6 with the information that it requires to execute properly.
TU_M_BE_C_OUT	Response	Commarea output message used to receive customer demographic information as provided by the existing CICS application DFHMABP6.

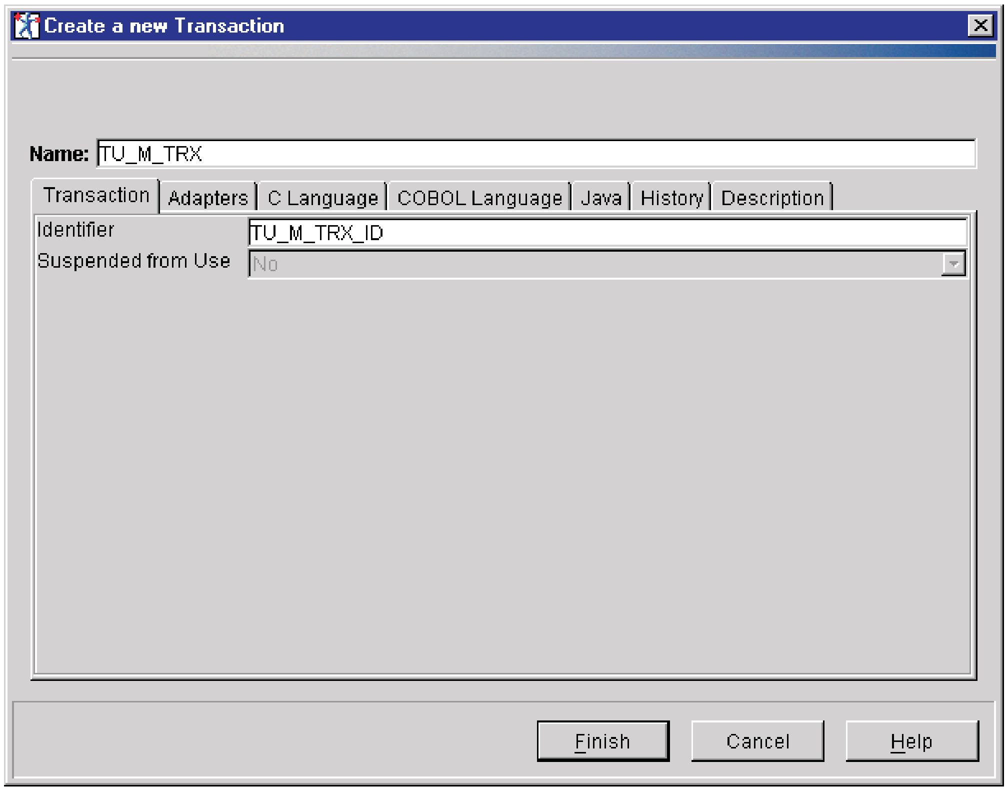
g. When you have completed importing the COBOL structured data type definitions listed in Table 26 on page 80, click **Cancel** to return to the workspace.

You just completed importing the COBOL structured data type definitions needed to model the MQ adapter. These data type definitions now reside as messages in the Control Center of the Adapter Builder.

__ Step 4. **Create Transactions.**

A transaction represents the message and data flowing to and from the back-end MQ program to be accessed by the adapter. In order to create an MQ command node, you need to associate the command node with a transaction. The messages associated with the transaction are defined as Input and Output representing the expected format of the input message (and identified as input terminal in the node) and the expected format of the output message (and identified as the output terminal in the node).

a. Create a transaction for the customer information. Right click on the Transaction folder. Select **Create > Transaction**. On the Create a new Transaction dialog, enter TU_M_TRX in the Name field and TU_M_TRX_ID in the Identifier field. Click **Finish**.



Name: TU_M_TRX

Identifier: TU_M_TRX_ID

Suspended from Use: No

Figure 53. Create a TU_M_TRX transaction

b. Add messages to the TU_M_TRX transaction. Right click on the TU_M_TRX transaction and select **Add > Message**. Messages allow for input and output from the transaction. On the Add an Existing Message dialog, select the TU_M_BE_C_IN and TU_M_BE_C_OUT messages (press the CTRL key and highlight both messages) and click **Finish**

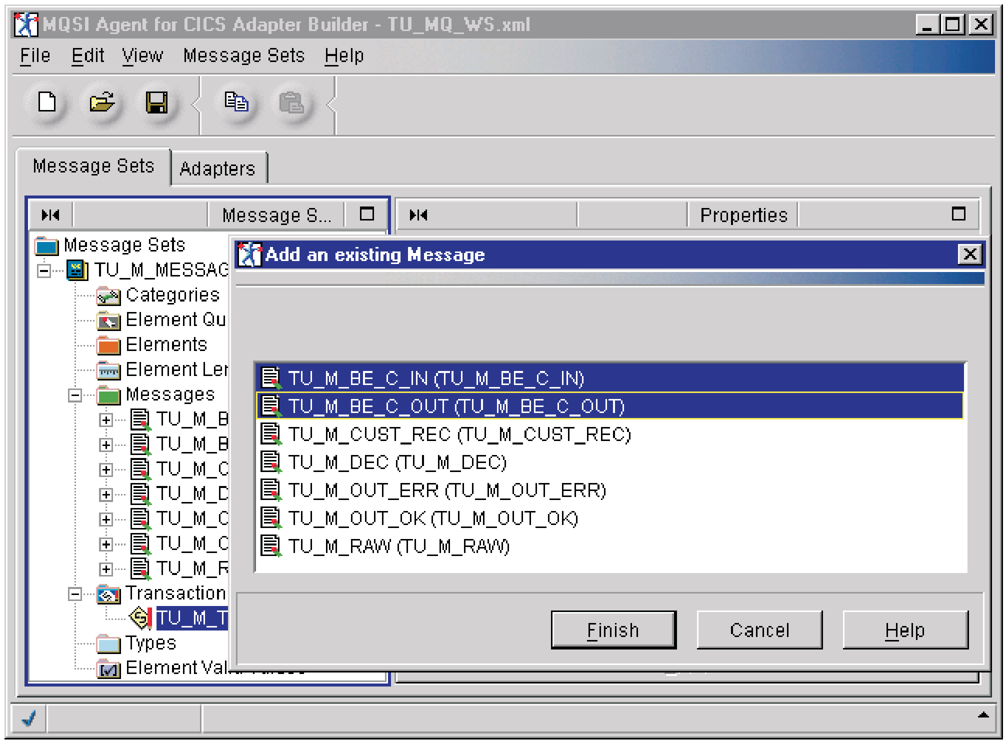
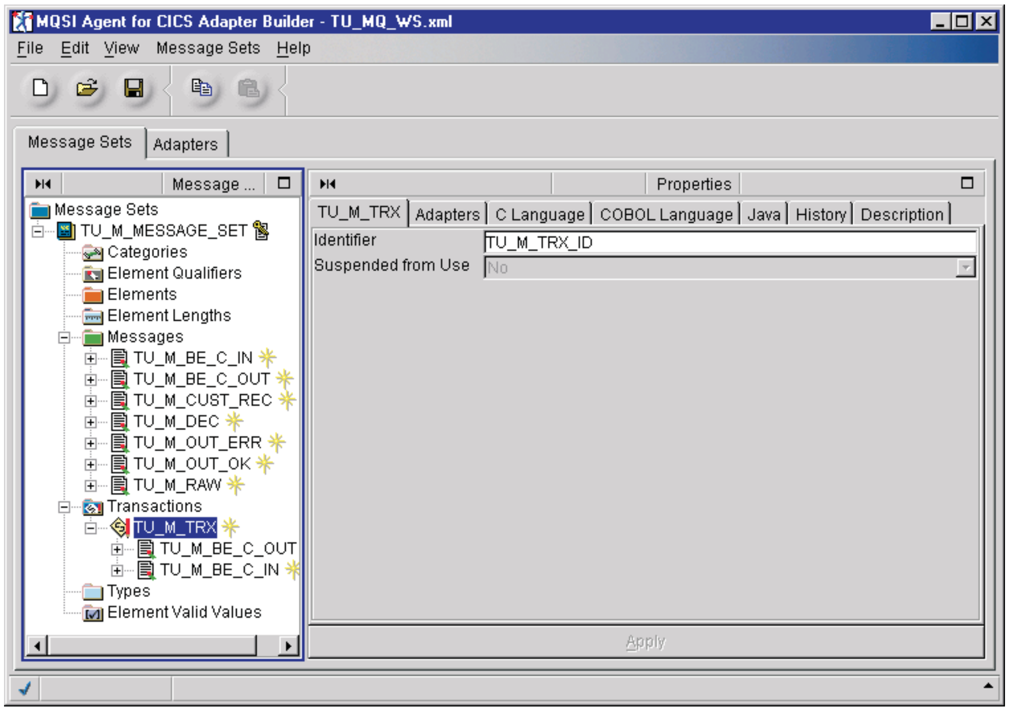


Figure 54. Add messages to the TU_M_TRX transaction

At this point, after adding messages to transactions the Message Sets view will appear as shown in Figure 55. Save your workspace by selecting **File > Save Workspace** from the menubar.



Identifier: TU_M_TRX_ID
Suspended from Use: No

Figure 55. Messages Sets view

You just created the MQ transaction and associated the input and output messages to the transaction.

You are now ready to create the component types. In this next step, you will associate the command component type with the transaction that you just created.

__ Step 5. **Create the component types for use in the microflow**

A component type represents a template that can be used as a building block in modeling the microflow.

When you complete the tasks in this step, you will have all the necessary component types required to model the adapter's functionality. The component types will display in the Adapter Tree View. From the Adapter Tree view you will be able to drag a component type onto the Microflow Definition pane and begin the process of constructing the flow.

This step is made of the following tasks:

- Create a Decision Type
- Create a Command Type
- Create a Data Context Type
- Create a Microflow Type

See the section *Composing microflows* in the **MQSI Agent for CICS Using the Control Center** documentation for descriptions of the component types.

a. **Create a Decision Type**.

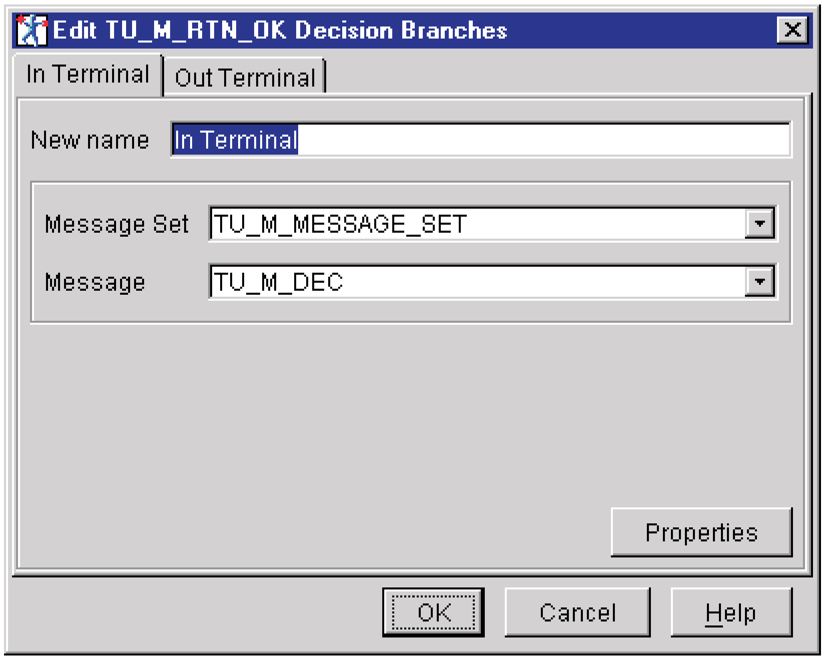
A decision type is necessary to test a condition for true or false, to resolve the control flow path.

You will use this type to create a Decision node for the microflow. The Decision node will be used to evaluate the message indicator (Good, Warning or Error) upon return from the program, DFHMABP6, and it will *decide* how processing will continue.

Click on the Adapters tab to switch to the Adapters view.

1) Create the TU_M_RTN_OK Decision type that will be used to determine whether the data returned from the back-end host is valid. Right click on the Decision Types folder and select **Create > Decision Type**. Enter *TU_M_RTN_OK* in the Name field and click **Finish**.

2) Associate a message set and message with the In Terminal on the TU_M_RTN_OK Decision type. Right click on the TU_M_RTN_OK Decision type under the Decision Types folder and select **Decision Branch**. Make sure the In Terminal tab is selected. Using the drop down menus, select TU_M_MESSAGE_SET for the Message Sets field and TU_M_DEC for the Messages field. Click **OK**.



New name: In Terminal
Message Set: TU_M_MESSAGE_SET
Message: TU_M_DEC

Figure 56. Editing the In Terminal on the Decision type

3) Create Out Terminals for the Good, Warning, and Error decisions. The TU_M_RTN_OK Decision type will determine which of these actions to take based on the MSG_IND_D field in the decision message (TU_M_DEC).
   a) Right click on the TU_M_RTN_OK Decision type under the Decision Types folder and select **Decision Branch**. Make sure the Out Terminal tab is selected. Click Out Terminal in the terminal list box and click **Rename**. Enter *Good* in the New name field and click **Finish**.
   b) Enter *Warning* in the Name field and click **Add**.
   c) Enter *Error* in the Name field and click **Add**. Click **OK**.

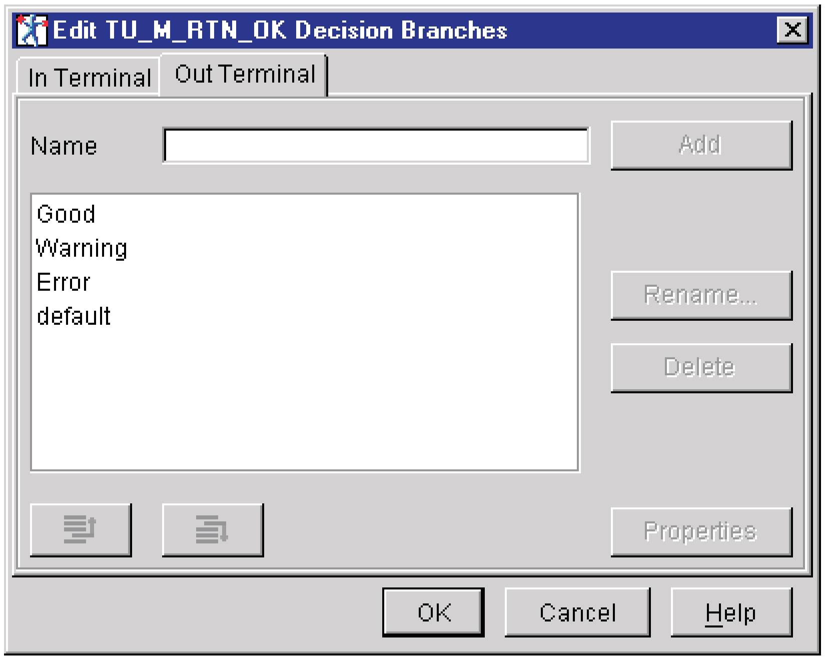
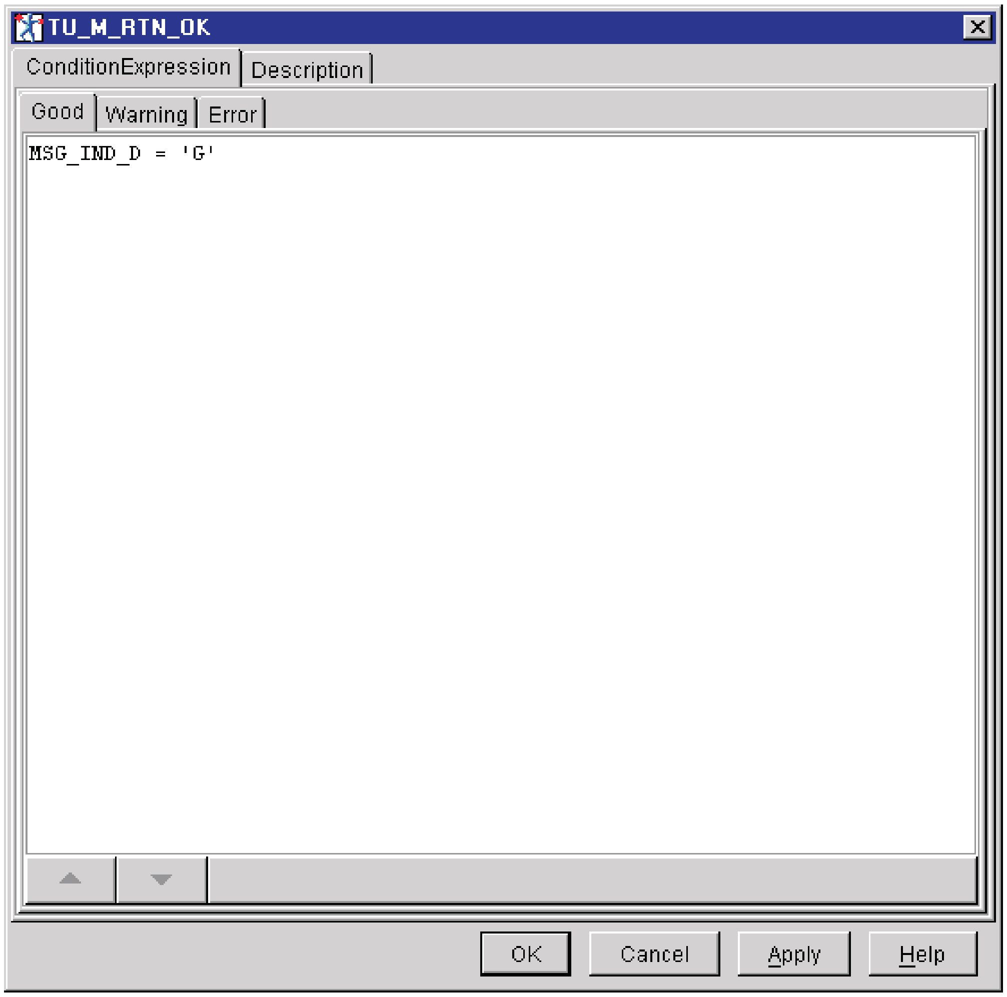


Figure 57. Editing the Out Terminal on the Decision type

4) Right click on the TU_M_RTN_OK Decision type and select **Properties** on the pop up menu. Make sure the ConditionExpression tab is selected and the Good tab is selected. Click in the Good test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_M_DEC message that we associated with the TU_M_RTN_OK Decision type). Select the *MSG_IND_D* field to add this to the ConditionExpression area.

   You should add the code shown in Figure 58 on page 87 for the Good terminal test condition. The letter 'G' for the MSG_IND_D field is based on the message indicator action codes that are defined for the decision message (TU_M_DEC). This screen capture shows code entered in the Good Condition Expression tab for the TU_M_RTN_OK Decision type.



```
MSG_IND_D = 'G'
```

Figure 58. Code for the Good Terminal

5) In a similar manner, add the test condition code for the remaining terminals: Warning and Error. When finished, click **OK**.

Table 27. Code for the Out Terminal actions for the TU_M_RTN_OK Decision type

Terminal	Code	Description
Good	MSG_IND_D = ’G’	G - Good - request processed
Warning	MSG_IND_D = ’A’	A - Application warning (e.g. ’Record Not Found’)
Error	MSG_IND_D = ’E’	E - System error (e.g. ’File Closed’)

b. **Create an MQ Command Type.**

A command type is a simple adapter component which, depending on how its properties are set, can be used to represent a server adapter program (DPL, MQ) or FEPI command (3270 screen interaction).

In this step, you will need to create an MQ Command type that will allow the microflow to perform the ’PUT’ that will trigger the server-side DFHMABP6 program.

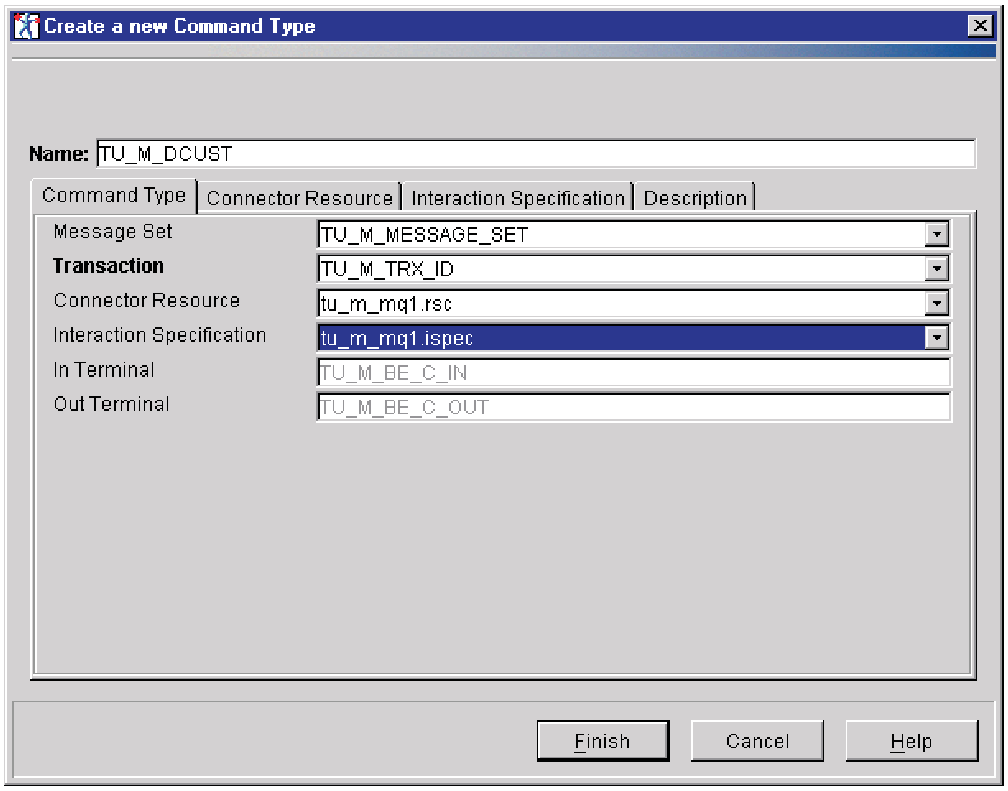
1) Create the TU_M_DCUST Command type for the DPL Command.

a) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_M_DCUST* in the Name field.

b) Using the drop down menus, set the following field property values:

Table 28. DCUST Command property values

Field	Value
Message Set	TU_M_MESSAGE_SET
Transaction	TU_M_TRX_ID
Connector Resource	tu_m_mq1.rsc
Interaction Specification	tu_m_mq1.ispec


Create a new Command Type

Name: TU_M_DCUST

Message Set	TU_M_MESSAGE_SET
Transaction	TU_M_TRX_ID
Connector Resource	tu_m_mq1.rsc
Interaction Specification	tu_m_mq1.ispec
In Terminal	TU_M_BE_C_IN
Out Terminal	TU_M_BE_C_OUT

Figure 59. Creating a TU_M_DCUST Command type

Click **Finish** to apply the property values.

c. **Create a Data Context Type.**

A data context type is a simple adapter component that is used to store data for later access through a data flow.

In this step, you will need to create a Data Context type to store customer information. This data can be accessed later from a connector data flow.

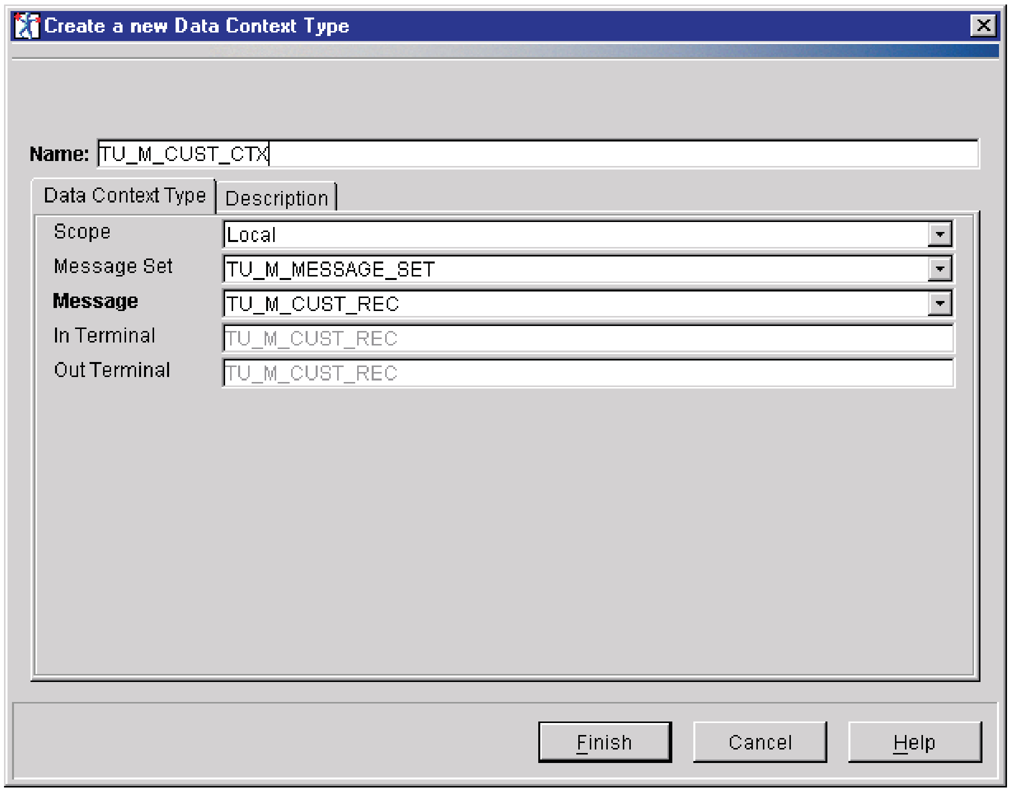
1) Create the TU_M_CUST_CTX Data Context type.

a) Right click on the Data Context Types folder and select **Create > Data Context Type**. Enter *TU_M_CUST_CTX* in the Name field.

b) Using the drop down menus, set the following field property values:

Table 29. TU_M_CUST_CTX Data Context property values

Field	Value
Scope	Local
Message Sets	TU_M_MESSAGE_SET
Messages	TU_M_CUST_REC



Name: TU_M_CUST_CTX

Scope	Local
Message Set	TU_M_MESSAGE_SET
Message	TU_M_CUST_REC
In Terminal	TU_M_CUST_REC
Out Terminal	TU_M_CUST_REC

Figure 60. Creating a TU_M_CUST_CTX Data Context type

Click **Finish** to apply the property values.

d. **Create a microflow type**

A microflow type is a collection of adapter components that models all or part of the message processing. In your adapter, this is the Navigator that calls the transaction and is responsible for controlling adapter request processing and managing states during the microflow processing.

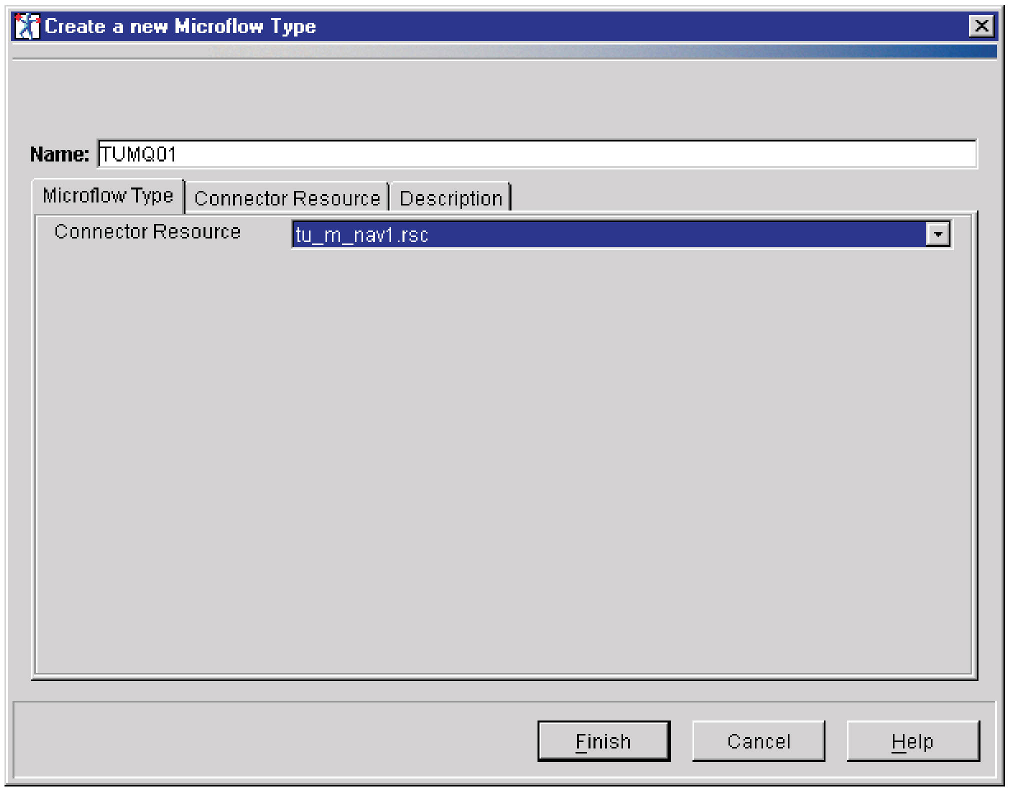
A navigator invokes server adapter programs.

In this step you will create a microflow that will model the processing of the customer data request.

1) Right click on the Microflow Types folder and select **Create > Microflow Type**.

2) Enter TUMQ01 in the Name field.

3) Use the drop down menu in the Connector Resource field to select tu_m_nav1.rsc as the Connector Resource file and then click **Finish**.



Name: TUMQ01

Connector Resource: tu_m_nav1.rsc

Figure 61. Creating a TUMQ01 Microflow Type

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created all of the component types that you will need to model your adapter.

__ Step 6. **Model the adapter.**

In this step you will perform a set of tasks to model the adapter. When you model an adapter you are specifying how the adapter will function at run time. Within the context of the business flow, the adapter model is of the navigation of the server application with the back end systems. The adapter represents the behavior you need to access data from the existing back end applications.

Within the builder, the model of the adapter is represented as a *microflow*, a sequence of nodes and connections. The microflow models the processing of a message as it passes from the input of the adapter to the output of the adapter.

This step is made of the following tasks:

- Adding microflow nodes
- Connecting the microflow nodes
- Defining the mappings

a. **Add the microflow nodes**

   In this task, you will drag the component types that you created in step 5 on page 84, onto the Microflow Definition pane. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

   1) **Add the Input Terminal node**

      An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

      a) Drag the node on to the Microflow Definition pane.

         In the Microflow Types folder, select the TUMQ01 microflow you created.

         Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

         Drag an Input Terminal type from the Adapter Tree View to the Microflow Definition pane

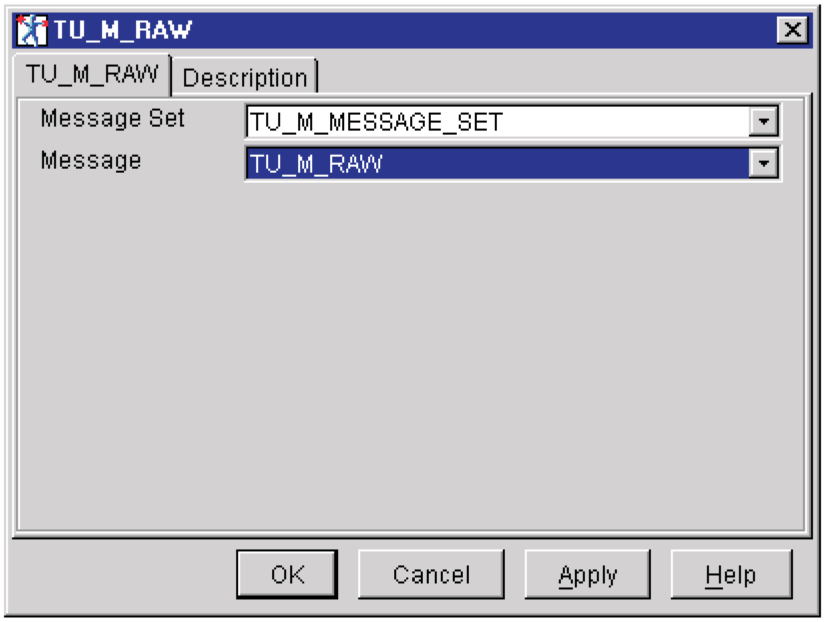
         Left click and hold on the Input Terminal to drag it to the Microflow Definition pane.

      b) Rename the node

         Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to TU_M_RAW and click **Finish**.

      c) Set the properties for the node

         Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_M_MESSAGE_SET in the Message Sets field and select TU_M_RAW in the Messages field. Click **OK**.


TU_M_RAW

TU_M_RAW | Description

Message Set: TU_M_MESSAGE_SET

Message: TU_M_RAW

Figure 62. Configuring the TU_M_RAW Input Terminal node properties

2) **Add the Command node**
   a) Drag the node on to the Microflow Definition pane

      From the Command Types folder in the Adapter Tree View, select a TU_M_DCUST Command type.

      Left click and hold on the TU_M_DCUST Command type to drag it to the Microflow Definition pane. Place the node to the right of the TU_M_RAW Input Terminal node.
   b) Rename the node

      Right click on the TU_M_DCUST1 Command node and select **Rename**. Modify TU_M_DCUST1 in the New name field to the name TU_M_DCUST and click **Finish**.
3) **Add the Decision node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_M_RTN_OK Decision type from the Adapter Tree View to the workspace. Place the node to the right of the TU_M_DCUST Command node.
   b) Rename the node

      Right click on the TU_M_RTN_OK1 Decision node and select **Rename**. Modify TU_M_RTN_OK1 in the New name field to the name TU_M_RTN_OK and click **Finish**.
4) **Add the Data context node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_M_CUST_CTX Data Context type from the Adapter Tree View to the workspace. Place the node above the TU_M_RTN_OK Decision node.
   b) Rename the node

      Right click on the TU_M_CUST_CTX1 Data Context node and select **Rename**. Modify TU_M_CUST_CTX1 in the New name field to the name TU_M_CUST_CTX and click **Finish**.
5) **Add the Output terminal node**

An Output Terminal serves as an exit point for the microflow. The Output Terminal can receive connections only. It can never start a connection. A microflow can have multiple Output Terminals (as in the MQ example). A developer must design the controlling application to recognize the possible reply messages provided by multiple Output Terminals.

a) Drag the node on to the Microflow Definition pane

   Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_M_CUST_CTX node

b) Rename the node

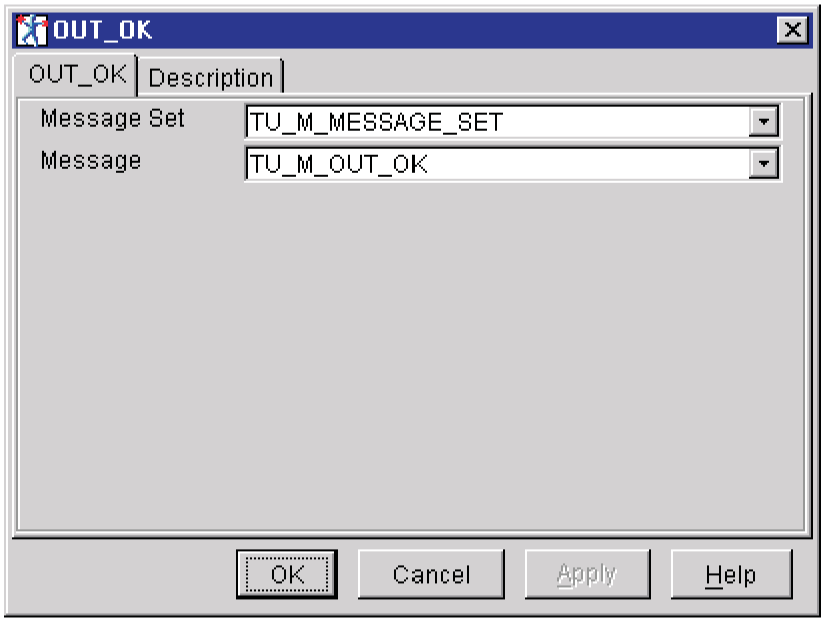
   Rename the Output Terminal node to OUT_OK. Click **Finish**

c) Flip the node

   Right click on the OUT_OK and select **Flip node**

d) Set the properties for the node

   Right click on the OUT_OK node and select **Properties**. From the drop down menus, select TU_M_MESSAGE_SET in the Message Sets field and select TU_M_OUT_OK in the Messages field. Click **OK**



Message Set: TU_M_MESSAGE_SET
Message: TU_M_OUT_OK

Figure 63. Configuring the OUT_OK Output Terminal properties

6) **Add the Error Output terminal node**

a) Drag the node on to the Microflow Definition pane

   Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_M_RTN_OK node.

b) Rename the node

   Rename the Output Terminal node to OUT_ERR.

c) Flip the node

   Right click on the OUT_ERR and select **Flip node**

d) Set the properties for the node

Right click on the OUT_ERR Output Terminal and select **Properties**. From the drop down menus, select TU_M_MESSAGE_SET in the Message Sets field and select TU_M_OUT_ERR in the Messages field. Click **OK**

7) Save your workspace by selecting **File > Save Workspace** from the menubar.

Your Microflow Definition panel should look something like this:

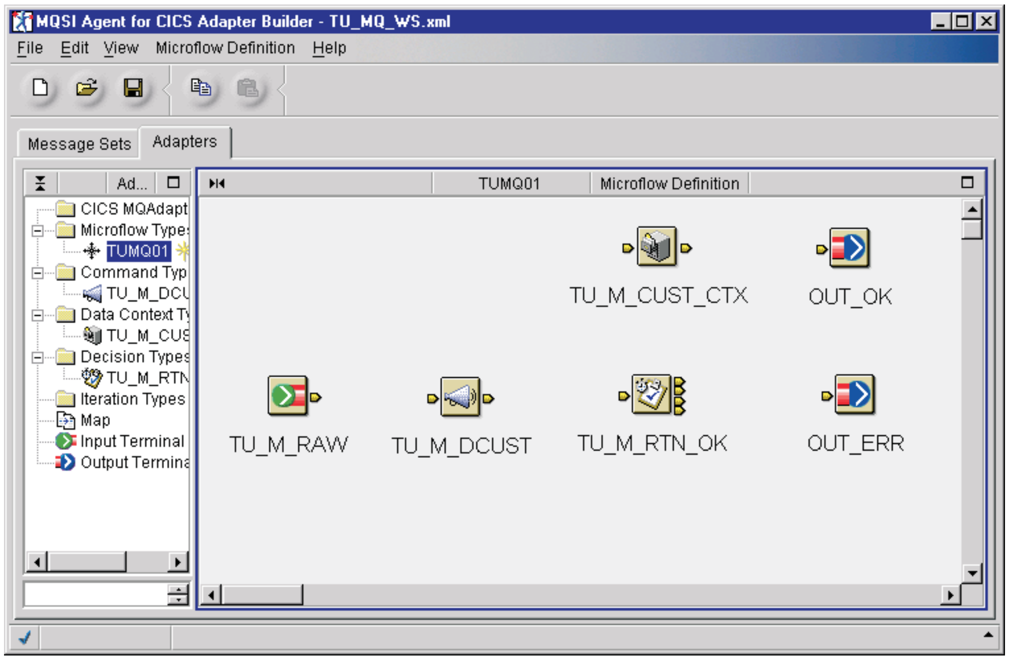


Figure 64. Nodes for the DPL adapter

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the TU_M_RAW Input Terminal node and select **Connect > Out**. Move the connection line to the TU_M_DCUST Command node and left click. This adds a control connection and a map (Map1 node) between the two nodes.

A control connection provides a sequential relationship between 2 nodes in a microflow.

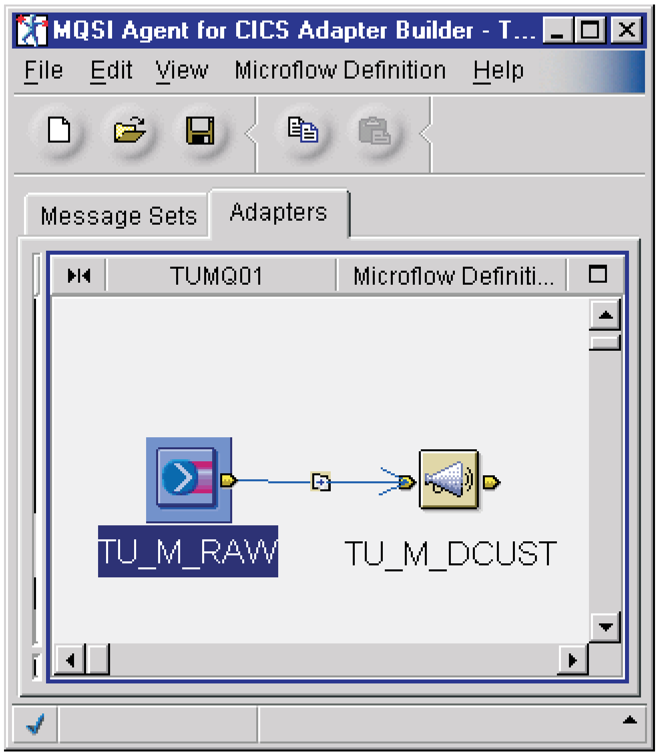

MQSI Agent for CICS Adapter Builder - T...
File Edit View Microflow Definition Help
Message Sets
Adapters
TUMQ01
Microflow Definiti...
TU_M_RAW
TU_M_DCUST

Figure 65. Connecting the TU_M_RAW Input Terminal and TU_M_DCUST Command node

2) Add a control connection from the out terminal on the TU_M_DCUST Command node to the TU_M_RTN_OK Decision node. This auto-adds a Map node (Map2) on the control connection line.

3) Add a Map node (Map3) between the TU_M_RTN_OK Decision node and the OUT_OK Output Terminal node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel).

4) Add a control connection from the first out terminal (labeled Good) on the TU_M_RTN_OK Decision node to the Map3 node and from the Map3 node to the OUT_OK Output Terminal node.

5) Add a series of three Map nodes (Map 4–Map 6) between the TU_M_RTN_OK Decision node and the OUT_ERR Output Terminal node.

Note: See Figure 66 on page 96 to see the placement of the four Map nodes (Map 3–Map 6). The Map node labels are for annotation only and will not appear in your workspace.

6) Add control connections from the second out terminal (labeled Warning) on the TU_M_RTN_OK Decision node to the Map4 node and from the Map4 node to the OUT_ERR Output Terminal node.

7) Add control connections from the third out terminal (labeled Error) on the TU_M_RTN_OK Decision node to the Map5 node and from the Map5 node to the OUT_ERR Output Terminal node.

8) Add control connections from the fourth out terminal (labeled default) on the TU_M_RTN_OK Decision node to the Map6 node and from the Map6 node to the OUT_ERR Output Terminal node.

9) Add a data connection from the Map2 node to the TU_M_CUST_CTX Data Context node and from the out terminal of the TU_M_CUST_CTX Data Context node to the Map3 node. Refer to Figure 66 to see all of the node connections in the microflow.

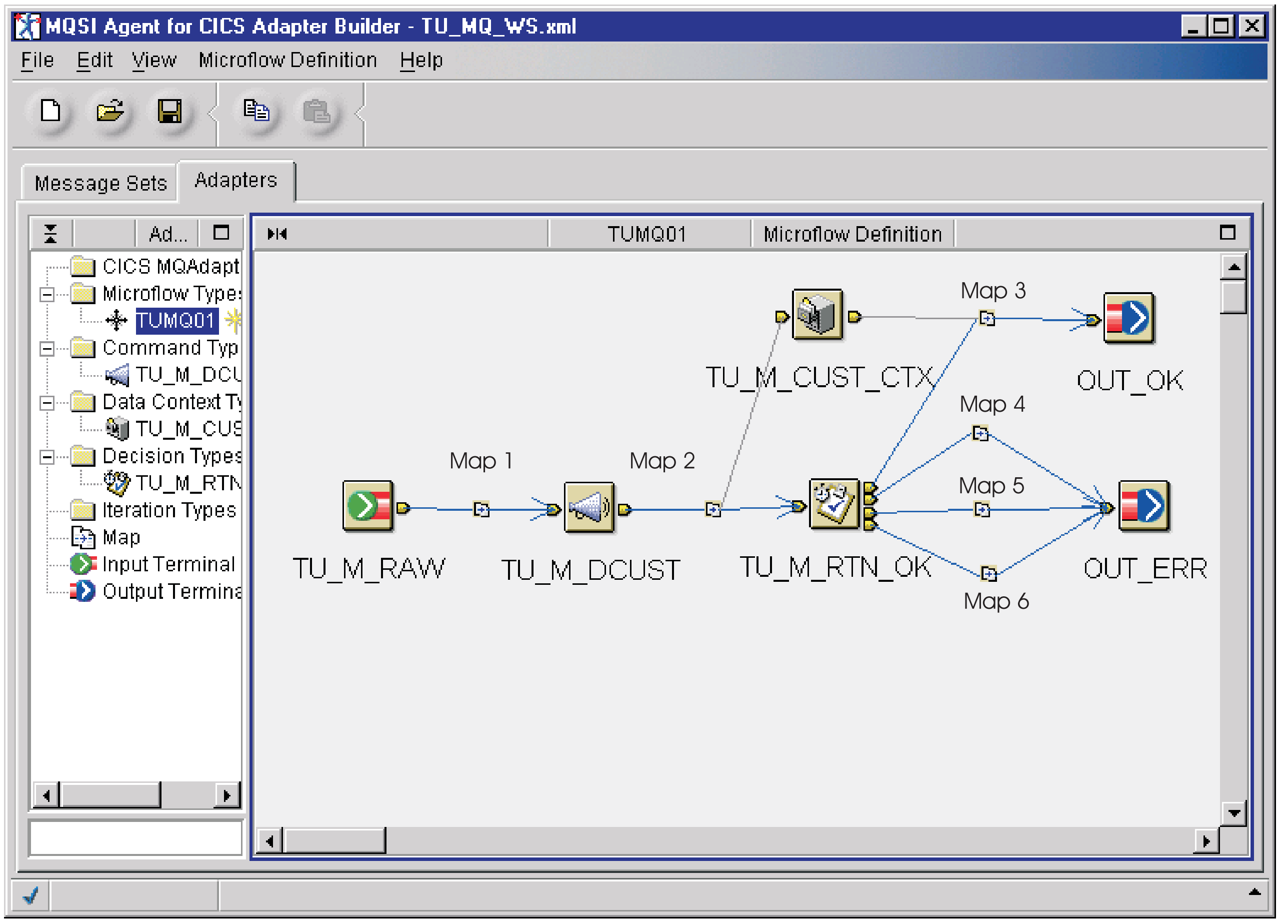


Figure 66. The TUMQ01 microflow

10) Save your workspace by selecting **File > Save Workspace** from the menubar.

c. **Map your adapter**

You are now ready to *map* your adapter. The act of *mapping* refers to the modeling of data transformation via a Map node, between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.

- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 30 and shown in Figure 67 on page 98. This map passes the customer number and inquire action indicator ′I′ to the MQ program DFHMABP6.

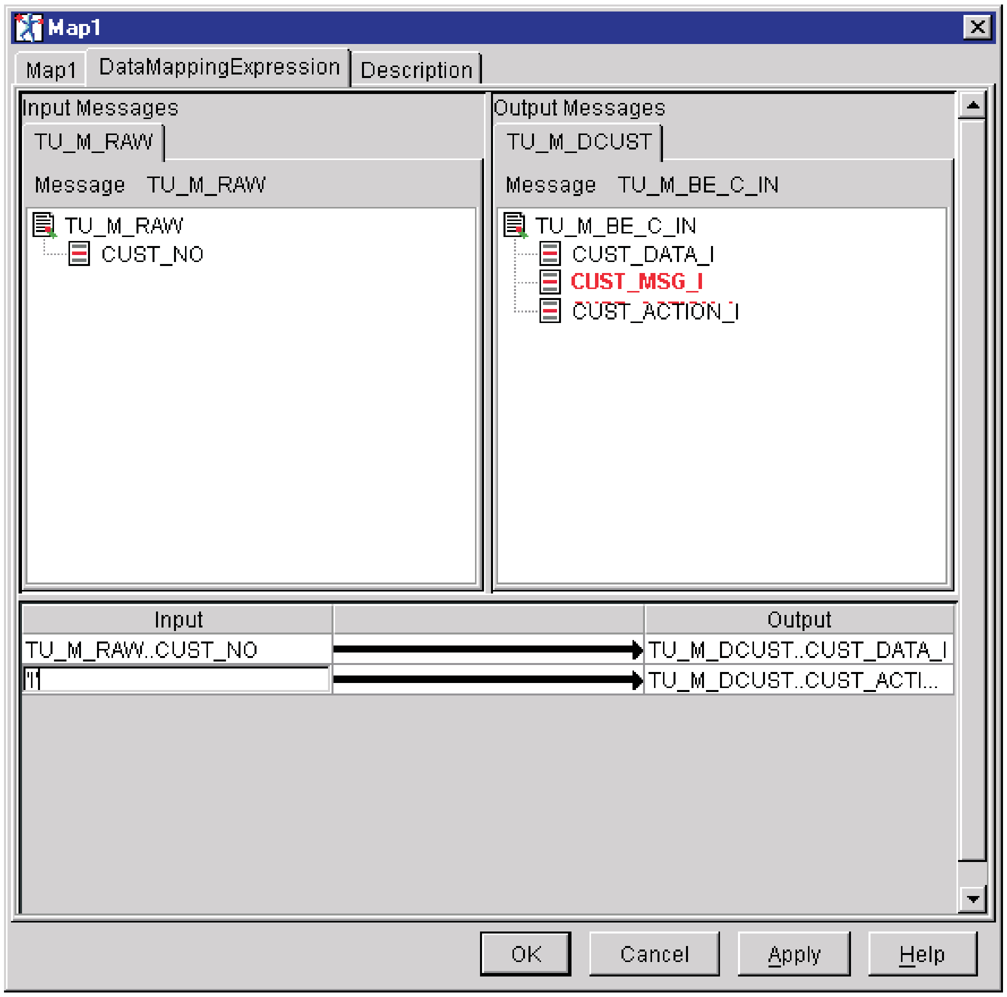
   Right click on the Map1 node (the Map node that appears between the TU_M_RAW and TU_M_DCUST nodes) and select **Properties**. Click the DataMappingExpression tab.

   Left click on the CUST_NO field under the TU_M_RAW message (view input message on right of panel) and drag the mouse cursor to the CUST_DATA_I field under the TU_M_BE_C_IN message (view output message on left of panel). This will create a mapping between the two fields (see Table 30 and Figure 67 on page 98).

   The second mapping (′I′ to CUST_ACTION_I) is a literal mapping. To perform a literal mapping, display the Map node′s properties and make sure the DataMappingExpression tab is selected. Right click on the destination field for the literal (the CUST_ACTION_I field in the TU_M_DCUST Output Message) and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ′I′ (quotes must be used). Click **OK**.

Table 30. Mapping fields for Map1 node (TU_M_RAW message to TU_M_BE_C_IN message)

Input Field	Output Field	Description
CUST_DATA	CUST_DATA_I	Used to pass customer data through the flow
′I′	CUST_ACTION_I	Type of action — Inquiry



Input		Output
TU_M_RAW..CUST_NO	→	TU_M_DCUST..CUST_DATA_I
'1'	→	TU_M_DCUST..CUST_ACTI...

Figure 67. Mapping for Map1 node

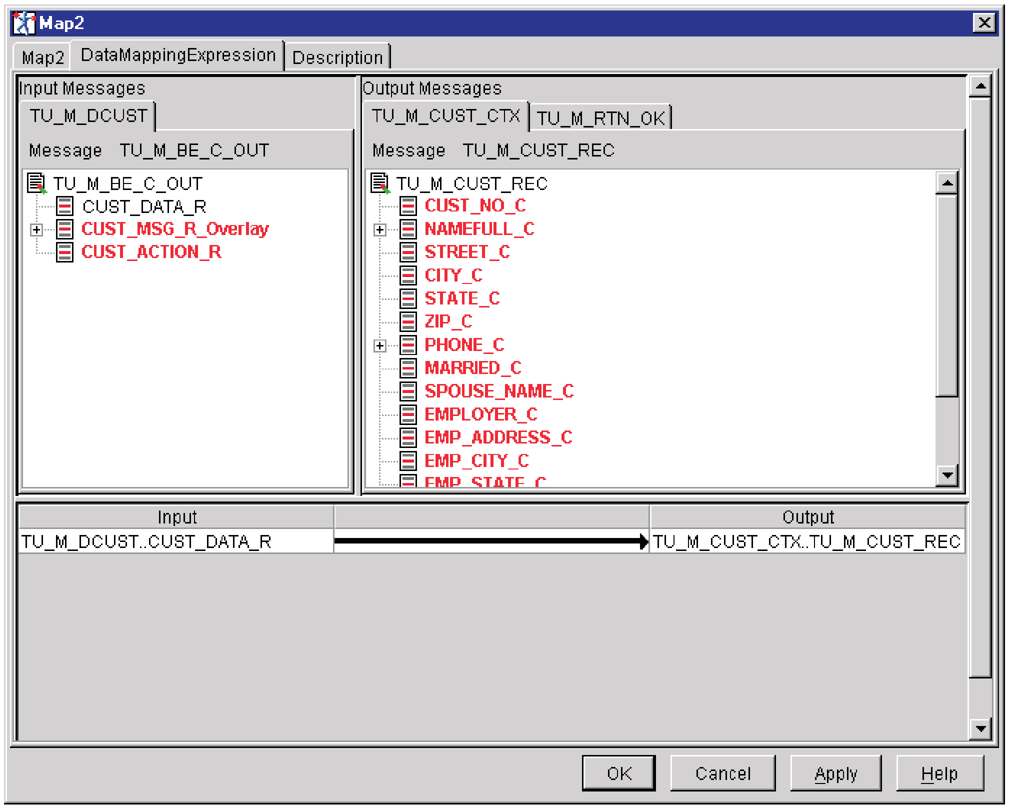
2) Perform the mapping for the Map2 nodes as listed in Table 31 and Table 32 and as shown in Figure 68 on page 99 and Figure 69 on page 100. This map passes the customer data from the MQ program DFHMABP6 to the TU_M_RTN_OK Decision node.

Table 31. Mapping fields for Map2 node (TU_M_BE_C_OUT message to TU_M_CUST_REC message)

Input Field	Output Field	Description
CUST_DATA_R	TU_M_CUST_REC	Store customer data received from the back-end system in a data context

Table 32. Mapping fields for Map2 node (TU_M_BE_C_OUT message to TU_M_DEC message)

Input Field	Output Field	Description
CUST_MSG_TXT_R	MSG_D	Message ID number
CUST_MSG_IND_R	MSG_IND_D	Output message
CUST_ACTION_R	ACTION_D	Requested action



Input Messages
TU_M_DCUST
Message TU_M_BE_C_OUT

TU_M_BE_C_OUT
CUST_DATA_R
CUST_MSG_R_Overlay
CUST_ACTION_R

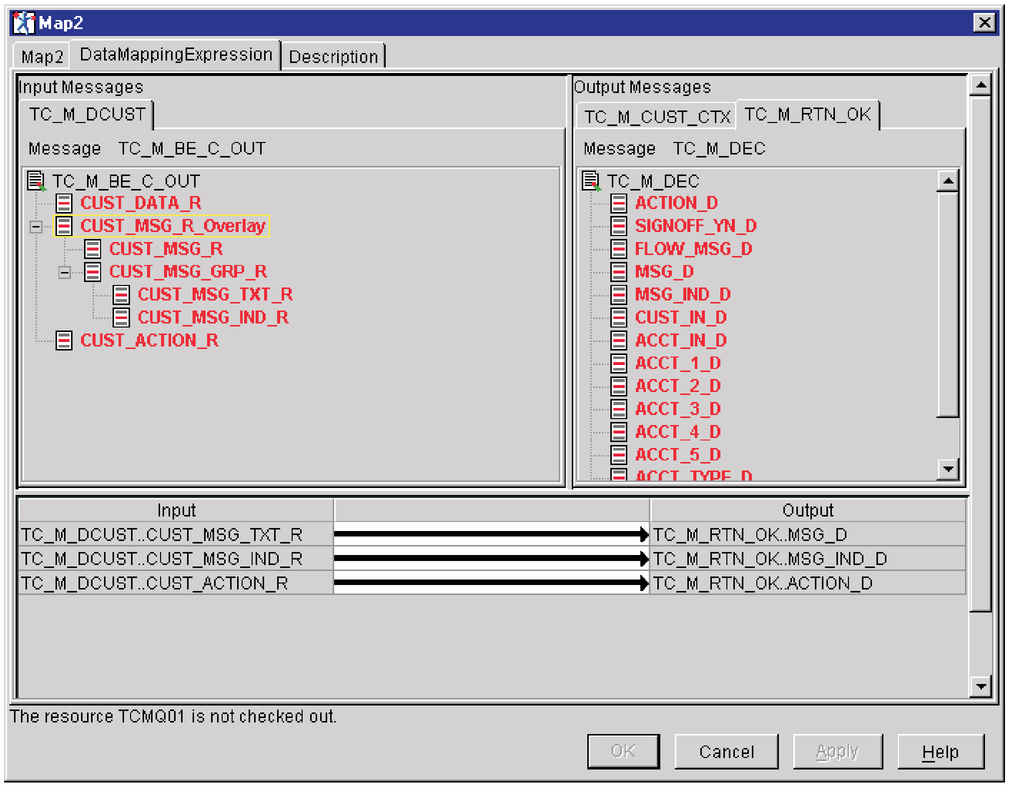
Output Messages
TU_M_CUST_CTX
TU_M_RTN_OK
Message TU_M_CUST_REC

TU_M_CUST_REC
CUST_NO_C
NAMEFULL_C
STREET_C
CITY_C
STATE_C
ZIP_C
PHONE_C
MARRIED_C
SPOUSE_NAME_C
EMPLOYER_C
EMP_ADDRESS_C
EMP_CITY_C
EMP_STATE_C

Input		Output
TU_M_DCUST..CUST_DATA_R	→	TU_M_CUST_CTX..TU_M_CUST_REC

Figure 68. Mapping for Map2 node

Note: CUST_MSG_R can be located by expanding the CUST_MSG_R_OVERLAY and the CUST_MSG_TXT_R and CUST_MSG_IND_R fields can be located by expanding CUST_MSG_GRP_R. Click on the + sign to expand.



Input		Output
TC_M_DCUST..CUST_MSG_TXT_R	→	TC_M_RTN_OK..MSG_D
TC_M_DCUST..CUST_MSG_IND_R	→	TC_M_RTN_OK..MSG_IND_D
TC_M_DCUST..CUST_ACTION_R	→	TC_M_RTN_OK..ACTION_D

The resource TCMQ01 is not checked out.

Figure 69. Mapping for Map2 node

3) Perform the mapping for Map3 node as listed in Table 33 and Table 34 and shown in Figure 70 on page 101. This map passes customer data and good response messages to the MQ Good response Output Terminal (OUT_OK).

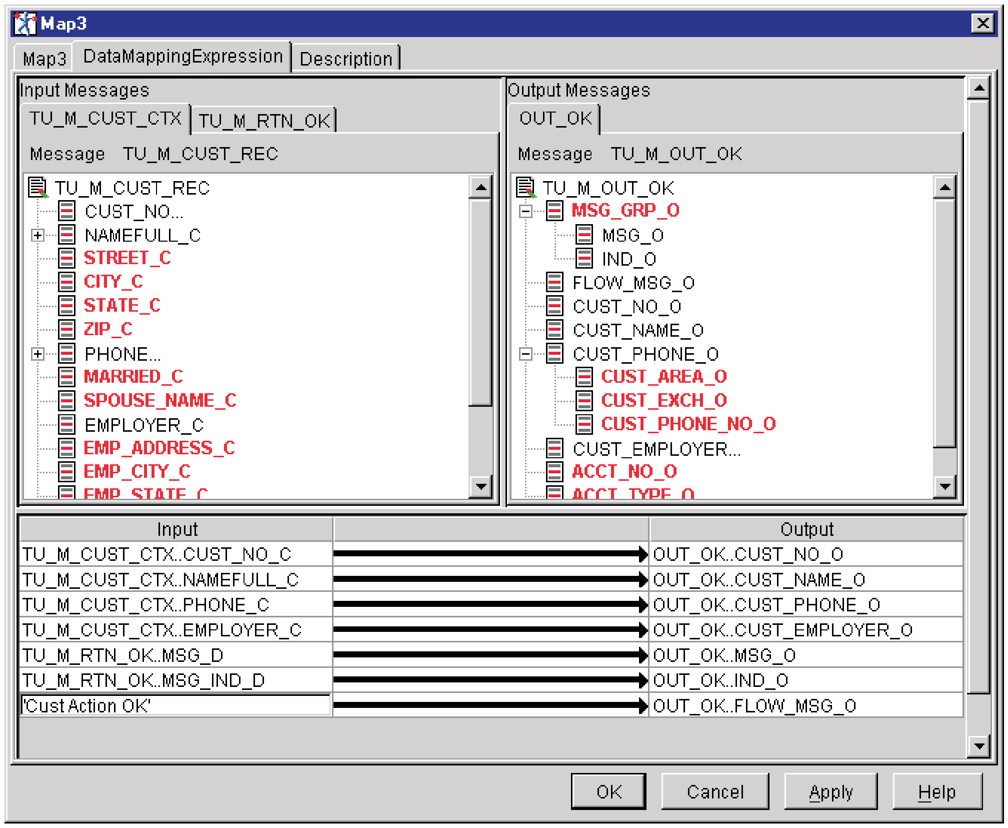
Table 33. Mapping fields for Map3 node (TU_M_CUST_REC message to OUT_OK message)

Input Field	Output Field	Description
CUST_NO_C	CUST_NO_O	Customer ID number
NAMEFULL_C	CUST_NAME_O	Customer name
PHONE_C	CUST_PHONE_O	Customer phone number
EMPLOYER_C	CUST_EMPLOYER_O	Customer employer

Table 34. Mapping fields for Map3 node (TU_M_RTN_OK message to OUT_OK message)

Input Field	Output Field	Description
MSG_D	MSG_O	Message ID number
MSG_IND_D	IND_O	Output OK message
'Cust Action OK'	FLOW_MSG_O	Response message for customer action

Note: The MSG_O and IND_O output fields can be located by expanding the MSG_GRP_O element. Click on the + sign.



Input Messages

TU_M_CUST_CTX | TU_M_RTN_OK

Message TU_M_CUST_REC

- TU_M_CUST_REC
  - CUST_NO...
  - NAMEFULL_C
  - STREET_C
  - CITY_C
  - STATE_C
  - ZIP_C
  - PHONE...
  - MARRIED_C
  - SPOUSE_NAME_C
  - EMPLOYER_C
  - EMP_ADDRESS_C
  - EMP_CITY_C
  - EMP_STATE_C

Output Messages

OUT_OK

Message TU_M_OUT_OK

- TU_M_OUT_OK
  - MSG_GRP_O
    - MSG_O
    - IND_O
  - FLOW_MSG_O
  - CUST_NO_O
  - CUST_NAME_O
  - CUST_PHONE_O
    - CUST_AREA_O
    - CUST_EXCH_O
    - CUST_PHONE_NO_O
  - CUST_EMPLOYER...
  - ACCT_NO_O
  - ACCT_TYPE_O

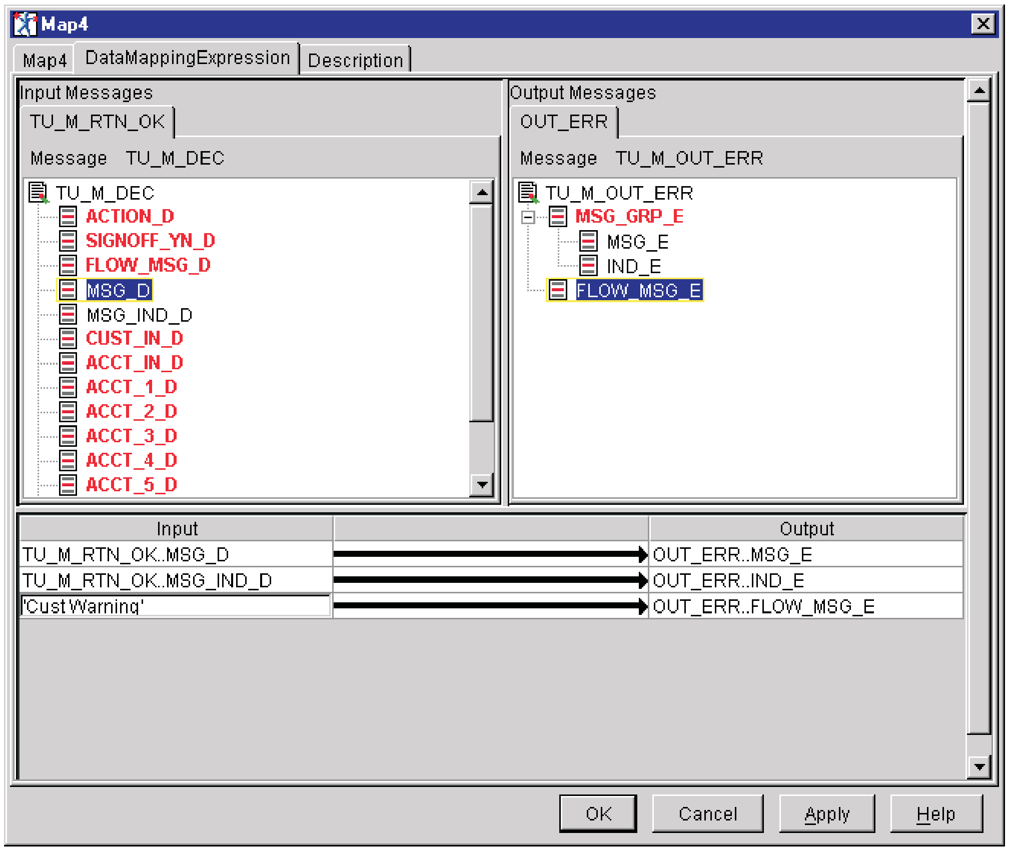
Input		Output
TU_M_CUST_CTX..CUST_NO_C	→	OUT_OK..CUST_NO_O
TU_M_CUST_CTX..NAMEFULL_C	→	OUT_OK..CUST_NAME_O
TU_M_CUST_CTX..PHONE_C	→	OUT_OK..CUST_PHONE_O
TU_M_CUST_CTX..EMPLOYER_C	→	OUT_OK..CUST_EMPLOYER_O
TU_M_RTN_OK..MSG_D	→	OUT_OK..MSG_O
TU_M_RTN_OK..MSG_IND_D	→	OUT_OK..IND_O
'Cust Action OK'	→	OUT_OK..FLOW_MSG_O

Figure 70. Mapping for Map3 node (TU_M_CUST_CTX and TU_M_RTN_OK messages to OUT_OK message)

4) Perform the mapping for Map4 node as listed in Table 35 and shown in Figure 71 on page 102. This map passes warning response messages to the MQ Error response Output Terminal (OUT_ERR).

Table 35. Mapping fields for Map4 node (TU_M_DEC message to TU_M_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
'Cust. Warning'	FLOW_MSG_E	Warning message text



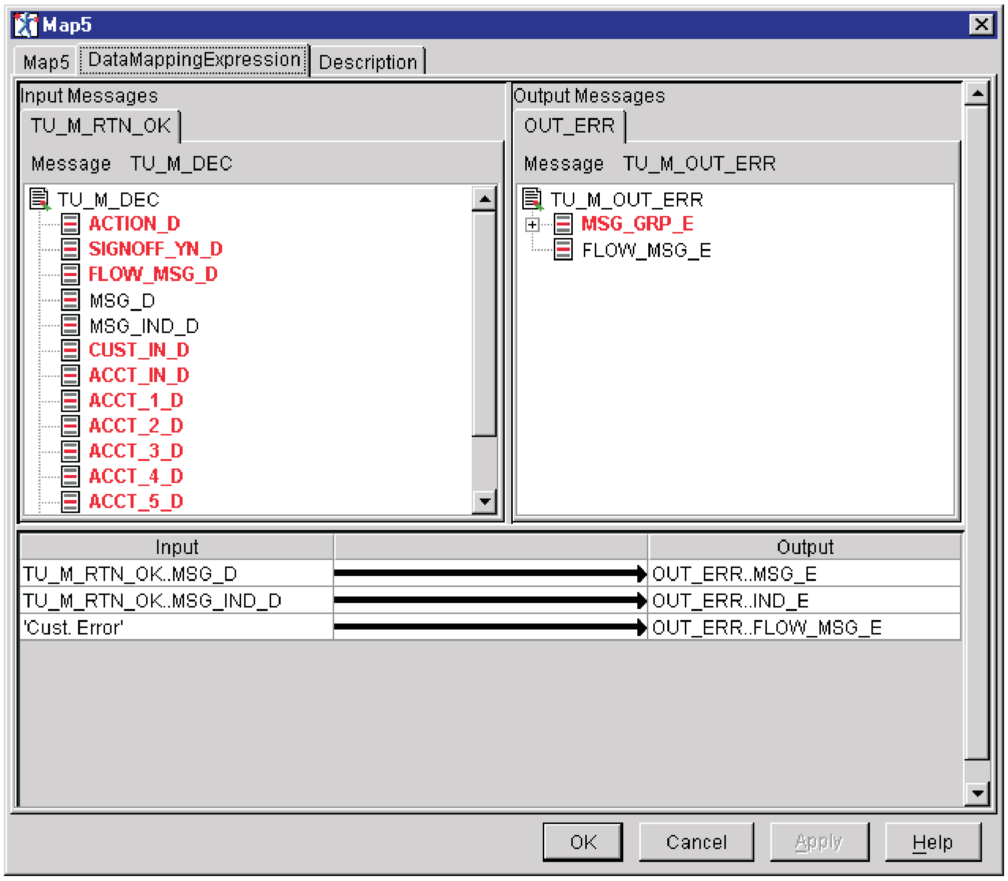
Input		Output
TU_M_RTN_OK..MSG_D	→	OUT_ERR..MSG_E
TU_M_RTN_OK..MSG_IND_D	→	OUT_ERR..IND_E
'Cust Warning'	→	OUT_ERR..FLOW_MSG_E

Figure 71. Mapping for Map4 node

5) Perform the mapping for Map5 node as listed in Table 36 and shown in Figure 72 on page 103. This map passes error response messages to the MQ Error response Output Terminal (OUT_ERR).

Table 36. Mapping fields for Map5 node (TU_M_DEC message to TU_M_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
’Cust. Error’	FLOW_MSG_E	Error message text



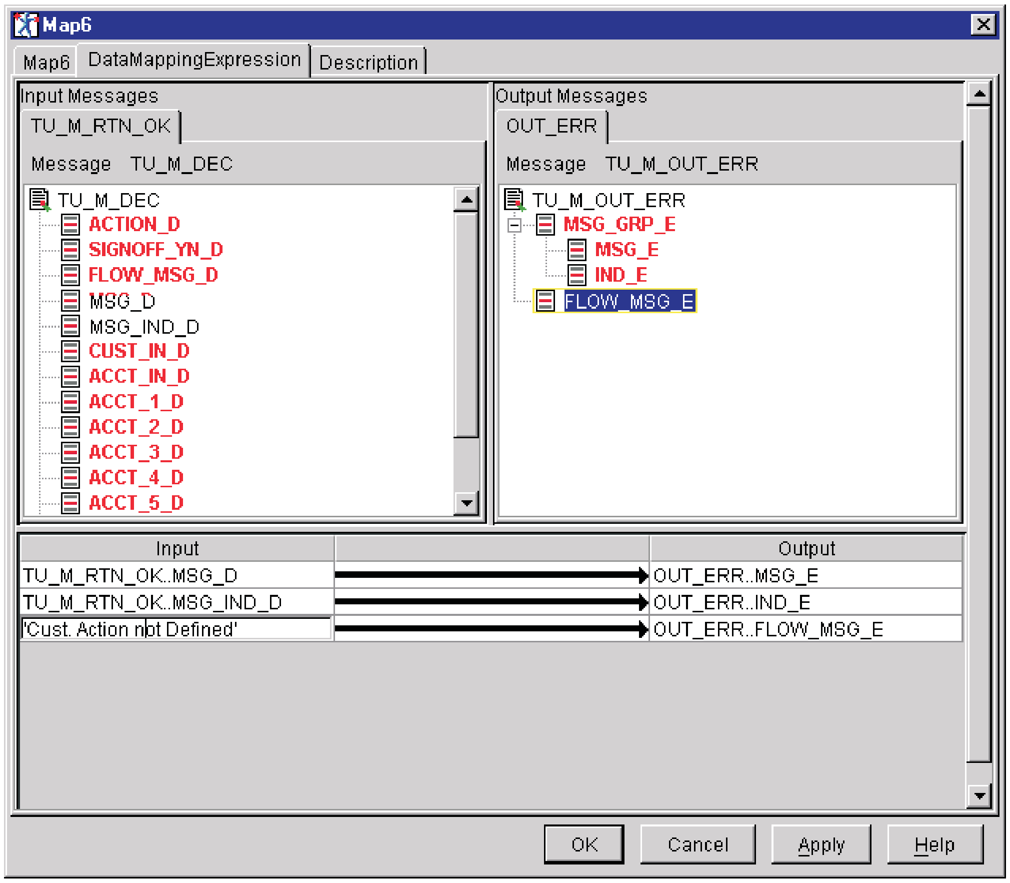
Input	Output
TU_M_RTN_OK..MSG_D	OUT_ERR..MSG_E
TU_M_RTN_OK..MSG_IND_D	OUT_ERR..IND_E
'Cust. Error'	OUT_ERR..FLOW_MSG_E

Figure 72. Mapping for Map5 node

6) Perform the mapping for Map6 node as listed in Table 37 and shown in Figure 73 on page 104. This map passes error response messages to the MQ Error response Output Terminal (OUT_ERR).

Table 37. Mapping fields for Map6 node (TU_M_DEC message to TU_M_OUT_ERR message)

Input Field	Output Field	Description
MSG_D	MSG_E	Error indicator
MSG_IND_D	IND_E	Output Error Message
’Cust. Action not defined’	FLOW_MSG_E	Error message text



Input		Output
TU_M_RTN_OK..MSG_D	→	OUT_ERR..MSG_E
TU_M_RTN_OK..MSG_IND_D	→	OUT_ERR..IND_E
'Cust. Action not Defined'	→	OUT_ERR..FLOW_MSG_E

Figure 73. Mapping for Map6 node

You just completed the modelling stage in the process of building your adapter.

In your model, you have coded the instructions on how the adapter is supposed to behave at run time. You are now ready to create the adapter.

__ Step 7. **Assign the model to a CICS MQAdapter**

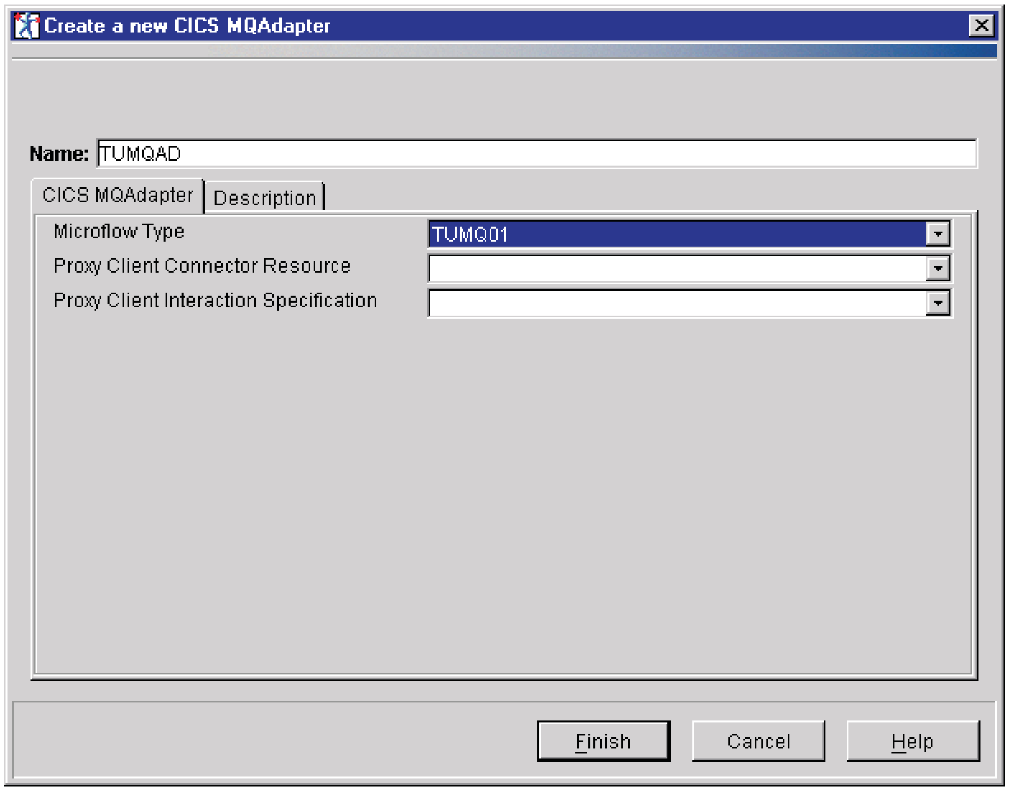
In this step you will associate the microflow (the model that you just completed), with a CICS MQAdapter.

A CICS MQAdapter provides the actual implementation of the adapter request processing.

a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

b. On the Create a new CICS MQAdapter dialog, enter TUMQAD for the Name and use the drop down menu to select TUMQ01 for the

Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TUMQAD

Microflow Type: TUMQ01

Proxy Client Connector Resource:

Proxy Client Interaction Specification:

Figure 74. Creating a CICS MQAdapter

You have completed the microflow and setup your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

Now you are ready to generate your adapter.

__ Step 8. **Generate the adapter**

The adapter code files will be generated in the output directory that you specify.

Adapter code generation is a two step process:

a. Generate copybooks from message definitions (in Message Sets view).

b. Generate the adapter run time code from the modeled microflow (in Adapters view).

- **Generate Copybooks**.

  You will generate copybooks for the following messages:
  - TU_M_RAW
  - TU_M_DEC
  - TU_M_OUT_OK
  - TU_M_OUT_ERR
  - TU_M_CUST_REC
  - TU_M_BE_C_IN

– TU_M_BE_C_OUT

Note: To generate a copybook for a message, the message must be checked out or newly created.

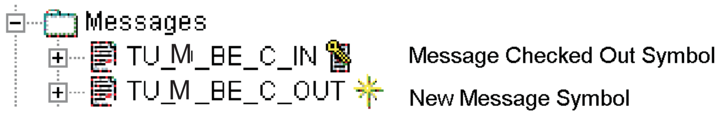


Figure 75. Messages Sets folder showing checked out message and newly created message This screen capture contains a Messages Sets folder showing a checked out message, indicating by an associated key symbol, and newly created message, indicated by a yellow star symbol.

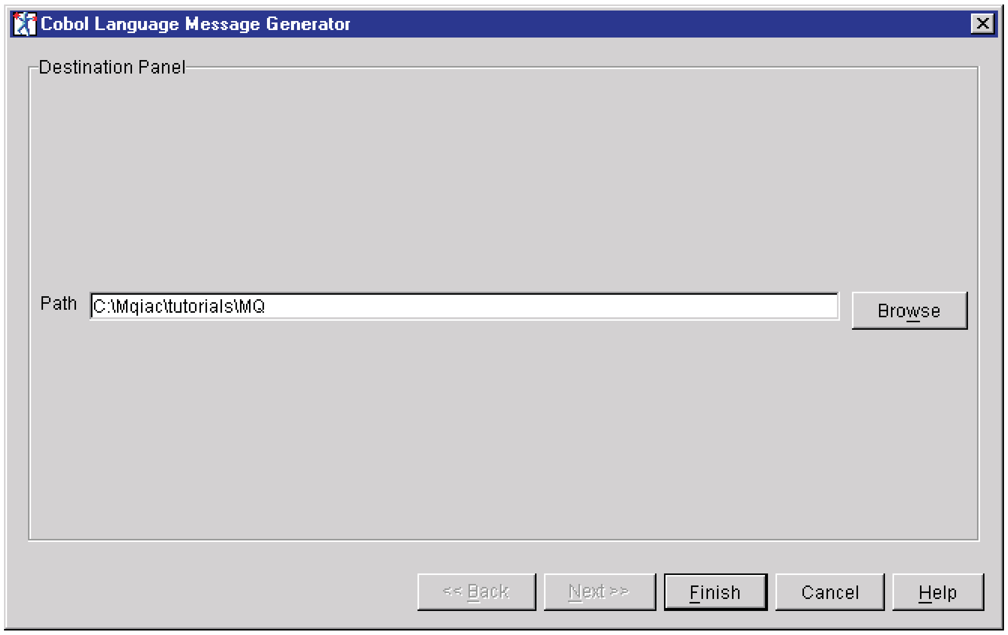
To generate copybooks, make sure that you are in the Message Sets view and then, follow this procedure:

a. Make sure the list of messages are visible under the Messages folder for the TU_M_MESSAGE_SET. To view the messages, click on the + sign in front of the Messages folder to display the list of messages.

b. Right click on the message for which you want to generate a copybook (for example, TU_M_RAW) and select **Generate > COBOL**.

c. Enter the output destination in the Path field and click Finish.

Note: The copybook generate removes underscores from the message names and only uses the first eight characters of the filename to generate the new copybook name.



Path C:\Mqiac\tutorials\MQ

Figure 76. Specifying pathname for copybook generation output

d. Repeat the process to generate copybooks for the remaining messages in the list.

- **Generate adapter Code**

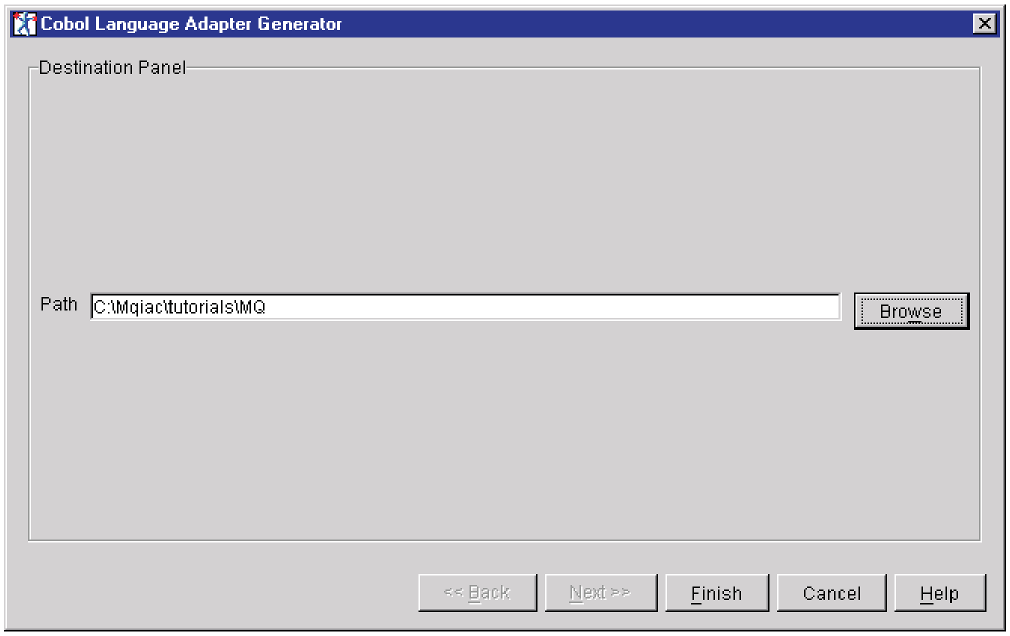
Build an adapter that supports an MQ interface

To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

Note: You must generate the adapter code in the same directory where you generated the copybooks.

a. Right click on TUMQAD adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination in the PATH field (the example uses C:\Mqiac\Tutorials\MQ). Click **Finish**.

   The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\tutorials\MQ

Figure 77. Specifying pathname for adapter code generation output

Deploying an adapter

GOAL

In the following section you will learn how to deploy the adapter that you created. The deploy operation sends the copybooks, source code, JCL and the configuration parameters for each microflow that you generated, to the host system, for source code configuration, object code build and parameter update operations.

You will need an account and password to the OS/390 environment that will host the adapter you are deploying.

Make sure that you have customized the build time JCL templates to your site standards. See "Building adapters" on page 6 for information on the JCL you need to customize.

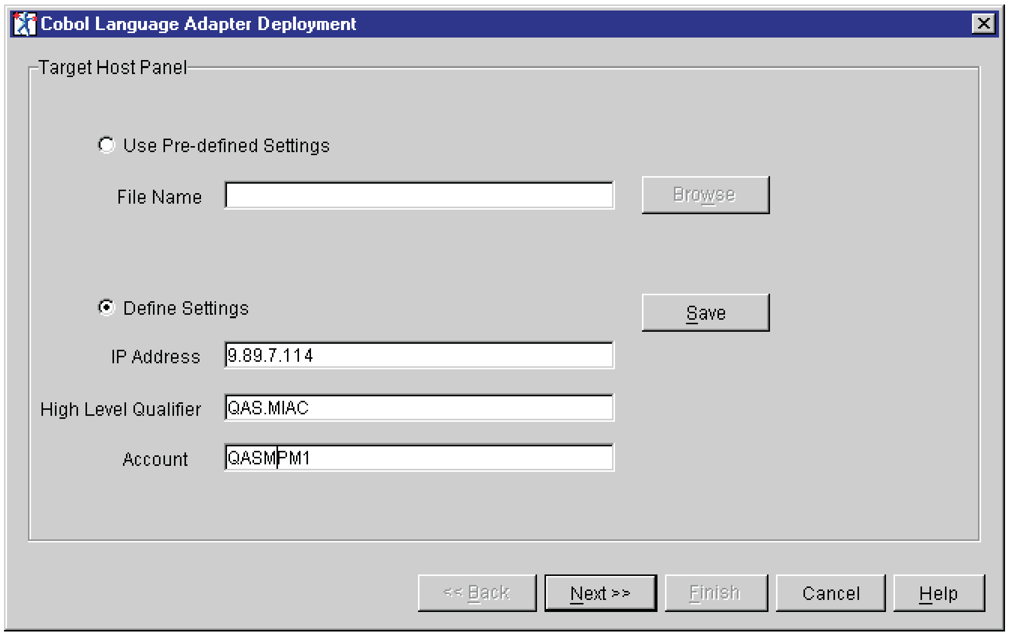
Build an adapter that supports an MQ interface

To deploy an adapter, make sure that you are in the Adapters view and then, follow this procedure:

1. Right click on TUMQAD adapter (listed under the CICS MQAdapters folder) and select **Generate > Deploy COBOL Adapter**. Click the Define Settings radio button and enter the following information:
   - IP Address — IP Address - The host system IP address (for example, 9.89.7.114)
   - High Level Qualifier — The high level qualifier for the partition data set (PDS)
   - Account — The account under which JCL submits a job for compilation.

   Note: If you wish to save these settings for reuse, then click **Save**. You will be prompted to specify an output location and filename to store the setting information. The next time you deploy adapter code you can click the **Use Pre-defined Settings** radio button and enter the saved filename.

   Click **Next**.



Define Settings [✓]

IP Address: 9.89.7.114

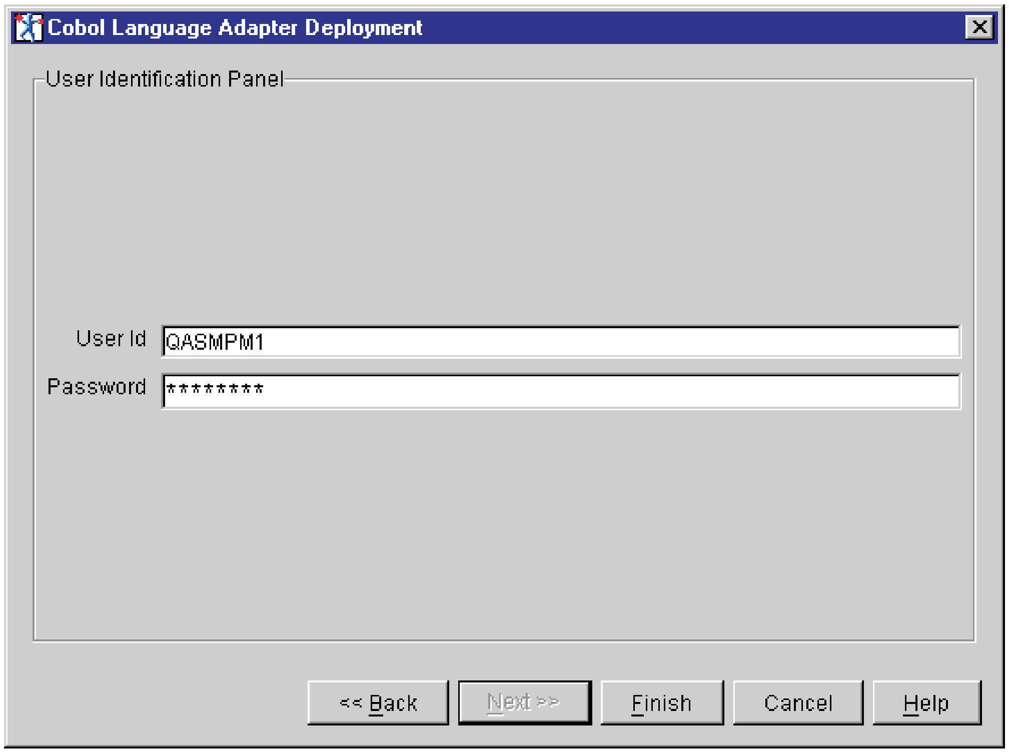
High Level Qualifier: QAS.MIAC

Account: QASMPM1

Figure 78. Specifying the target host

   Note: The values displayed in Figure 78 are for example purposes only. The values that you enter will depend on your site's host system parameters.

2. On the User Identification panel enter your user ID and password.



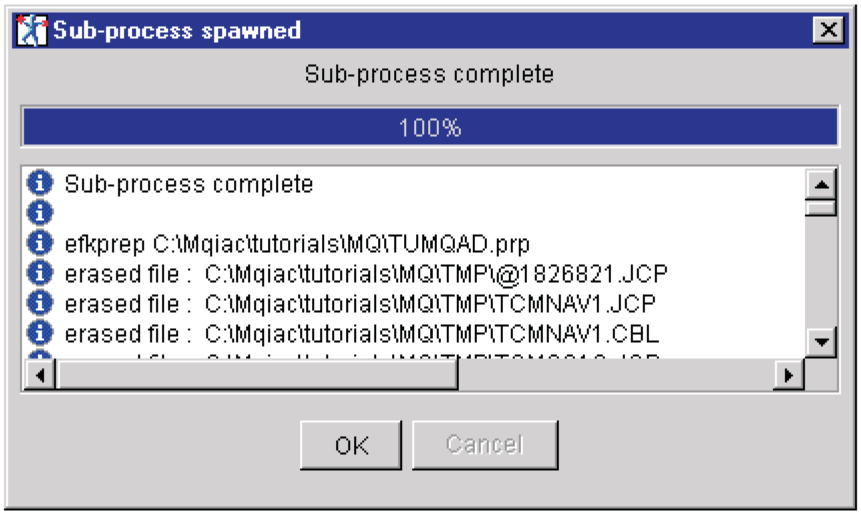
User Id QASMPM1
Password ********

Figure 79. Logon to the host

Click **Finish**.

3. The *Sub-process dialog* appears and provides a status of the deploy process as it happens. When the deploy is complete the generated adapter code, copybooks, and JCL (Compile / Properties File Update) files will be moved to the OS/390 server.

   Note: You should scroll through the output listing in the Sub-process dialog window to see if any errors occurred.



```
Sub-process complete

efkprep C:\Mqiac\tutorials\MQ\TUMQAD.prp
erased file :  C:\Mqiac\tutorials\MQ\TMP\@1826821.JCP
erased file :  C:\Mqiac\tutorials\MQ\TMP\TCMNAV1.JCP
erased file :  C:\Mqiac\tutorials\MQ\TMP\TCMNAV1.CBL
```

Figure 80. Sub-process dialog indicating status of the deploy process

4. Select **OK** to close the dialog.

The adapter now resides on the OS/390 server and is ready to be tested. See Chapter 6, “Validating the adapters” on page 231 for instructions on how to test the adapter.

Check to see that the adapter compiled in CICS

After you have deployed the adapter to the OS/390 server, you need to make sure that it compiled with no errors. Consult with your CICS systems administrator for assistance with this procedure.

Defining the adapter resources to CICS

If you do not have access to CICS at your site, you will need to ask your CICS administrator to perform the necessary CEDA and CEMT functions. You will need to provide the CICS administrator with the following information as it relates to the adapter that you deployed:

- Program names
- Group name
- Transaction Identifiers

For the MQ adapter, the following values apply:

Table 38. Values for the Define Transactions screen

Program	Group	Transid
TUMNAV1	MIACUSER	TUM1
TUMQ01P	MIACUSER	TUMP
TUMQ01G	MIACUSER	TUMG

To define resources to CICS, the CICS administrator must:

- Run the CEDA transaction to define programs and any files to CICS.
- Submit JCL to run the Properties File Update job.

  This is necessary only if you did not automatically submit JCL using the builder’s generator facility.

  If you were not allowed to submit JCL automatically, you can manually submit JCL (DFHMAMPU) to run the Properties File Update job (DFHMAMUP). See the MQSI Agent for CICS Run Time User’s Guide for information on the Properties file update JCL (DFHMAMPU).

The CICS administrator must NEWCOPY any server adapter programs that were modified.

For an example of defining CICS resources to CICS, See “Example procedure for defining adapter resources to CICS” on page 239.

Chapter 5. Build an adapter that supports a FEPI interface

Before you begin this tutorial, read Chapter 2, “Tutorial overview” on page 9. The Tutorial Overview section contains important information on the business transaction to be modeled, as well as information on the tutorial’s file structure. The Tutorial Overview also lists the assumed environment requirements that must be adhered to in order to build and deploy the adapter. The Tutorial overview also provides important information on the tutorial files, the tutorial directory structure and how to avoid naming conflicts when you create message sets and messages.

From this tutorial, you will learn how to use the MQSI Agent for CICS Adapter Builder tool to model and generate code for an adapter that supports a FEPI interface.

You will model an adapter that has the functionality to access an existing application using a CICS / FEPI Interface. See “About the adapter you will design” on page 112 for a description of the adapter that you will model.

If you have not had any exposure to the Adapter Builder component of the MQSeries Integrator Agent for CICS Transaction Server product, you should read the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center*. This book contains information on the concepts of the MQSeries Integrator Agent for CICS Adapter Builder.

This tutorial consists of:
- “Designing an adapter”
- “Creating an adapter that supports a CICS FEPI interface” on page 120
- “Deploying an adapter” on page 225

After completing this tutorial you should be able to:
- Identify required Host based information you need to gather and use.
- Import COBOL copybooks and create message sets.
- Import 3270 screens.
- Create workspaces to define adapter flow logic.
- Create and generate a COBOL adapter.
- Deploy and test the generated COBOL adapter.

Designing an adapter

As was discussed in “Requirements analysis and design considerations” on page 1, before you start to use the MQSI Agent for CICS Adapter Builder, you would spend some time analyzing the business need that the adapter will address and then spend some time considering how you will design the adapter.

When you finish with requirements analysis and design considerations, you should have a sound understanding of how your adapter will *behave* at run time in order to manage and fulfill a business transaction.

Build an adapter that supports a FEPI interface

To help you gain a frame of reference for what you will create in this tutorial, you should understand the following:

- The business need to be addressed
- The messages in and out structure
- The CICS resources required

Addressing a business need

An adapter should address a particular business need. In this tutorial, the business need is to provide a controlling application with an interface to a back-end environment for the purpose of accessing an existing CICS application that performs a customer inquiry.

In this tutorial you will be accessing the same back-end environment that was installed and used by the run time installation verification procedure (IVP). For information on the programs used by the IVP, see the chapter on performing post installation tasks in the *MQSI Agent for CICS Run Time User's Guide*.

About the adapter you will design

This section identifies some preliminary information that you would gather or consider prior to invoking the builder. It is essential that you understand your objectives and the environment you will be working in.

The adapters that you build using the MQSI Agent for CICS Adapter Builder are visual models of business transactions. They are intended to map out the activities that comprise the entire business transaction, from invocation to completion.

The adapter that you build contains the instructions, logic and code that enable it to run on an OS/390 server, this includes an interface technology for accessing information on the back-end system. In this tutorial the business transaction on which you will base your adapter is a *customer inquiry request* and the interface method used is a CICS / FEPI interface.

Your adapter design will include instructions on accessing a back-end application to retrieve customer information and will include instructions on where to put the information so that it can be returned to the controlling application.

The tutorial models a FEPI adapter that consists of a base navigator microflow and five *subflows* A subflow is a microflow that is nested under the base navigator flow:

- The main flow or the *base navigator microflow* (TU_F_NAV) — represents the adapter microflow that models the behavior of the Navigator in the run time environment.
- Subflow (TU_F_PARSER) — identifies the current 3270 screen via FEPI
- Subflow (TU_F_SIGNON) — signs the user onto the Customer application via FEPI
- Subflow (TU_F_INQ) — performs an inquiry on Customer information via FEPI
- Subflow (TU_F_SGNOFF) — signs off from the Customer application and displays a blank CICS screen via FEPI
- Subflow (TU_F_RESET) — resets the Customer screen for inputting the next customer number via FEPI.
- This tutorial describes how to design and construct an adapter that when deployed will perform the following functions:

1. Accept the message structure TU_F_RAW from the Simulator in the form of a request.
2. Pass the information to the parser subflow (TU_F_PARSER):
   - The parser command (TU_F_PARSER) will determine the 3270 screen identity
   - Map the screen identifier to the DOC_SCR indicator
3. Use a decision node (TU_F_SCR_ID) to determine the screen identity passed from the TU_F_PARSER subflow. Based on the identity of the screen, select one of the following flow paths:
   - Signon to the application (TU_F_SIGNON subflow)
   - Reset the customer screen (TU_F_RESET subflow)
   - Unknown, which terminates the flow and outputs a reply message.

   Note: To continue with the signon path go to step 4. To continue with the reset the customer screen path go to step 12 on page 114.
4. The signon to the application subflow (TU_F_SIGNON subflow) performs the following:
   - Use the SYS_LU_LOGON data context to supply the values in LU_OWNER and PASSTICKET so these values can be mapped to the appropriate USERID and PASSWORD fields on a 3270 Signon Screen.
   - The signon command (TU_F_SIGNON) uses the supplied LU_OWNER and PASSTICKET values to display a blank native CICS screen.
   - If signon is successful, the TU_F_CMAV command inputs the CMAV transaction to display the Customer screen. Output a good customer application screen reply.
   - If signon is not successful or the screen returned is unknown, output an error reply.
5. Check the reply message for either a good signon or error using a Decision node (TU_F_GOOD_SIGNON).
6. For a successful signon, perform a Customer information inquiry using the TU_F_INQ subflow.
   - Map the Customer inquiry request information to the Customer Information screen.
   - Process the Customer inquiry command
   - Use a Decision node (TU_F_REC_NOT_FND) to determine whether a valid data record is returned for the Customer inquiry.
   - If the data record exists, map the data in the record to the output reply message, otherwise map an error reply message.
7. Store the data record information in a Data Context node (TU_F_HOLD_REPLY). Map the input data (TU_F_RAW) to the message supplied to a Decision node (TU_F_SIGNOFF).
8. Decide whether to signoff (release the LU connection) using a Decision node (TU_F_SIGNOFF).
9. If signoff is selected. Map information from the Data Context node (TU_F_HOLD_REPLY) and the FEPI Data Context node (SYS_FEPI_OVERRIDES) to the input message supplied to the signoff subflow (TU_F_SGNOFF). The FEPI Override feature is used to dynamically set the FEPI LogoffType property.
10. The signoff subflow (TU_F_SGNOFF) performs the following:

– Maps a signoff key (PF3) to the Customer Information screen.
– If the signoff is valid, then map the required CESF LOGOFF transaction information and process the transaction using the CESF LOGOFF command. Output reply message indicating a valid logoff to be returned to the Simulator and exit.
– If the signoff is not valid, then output a reply message indicating an invalid logoff to be returned to the Simulator and exit.

11. If signoff is not selected, then in the main flow pass along the Reply record to an output reply message to be returned to the Simulator and exit.
12. If you are already on the Customer screen because of a previously processed inquiry, then reset the Customer screen to accept input for the next customer number. The reset subflow (TU_F_RESET) performs the following:
    – Maps a PF12 key to reset the Customer screen for new input.
    – Process the PF12 input using the Customer screen command (TU_F_CUST).
    – If an error occurs during the Customer screen reset, then output an error message.

Identify the components of the run time environment

Before building the adapter we need to:

- Define the CICS programs and transaction IDs for the adapter programs that are generated.

  For the purpose of this tutorial the following will be generated:

Table 39. MQ Adapter programs

Program type	Program name	Transaction ID
Navigator	TUFNAV	TUF1
FEPI Adapters	TUFPRSER	TUF4
	TUFSGON	TUF2
	TUFINQ	TUF6
	TUFSGOFF	TUF3
	TUFRESET	TUF5

- Determine the FEPI screens that are invoked. For this tutorial the following screens will be invoked:
  – Signon screen (TUFSGON)
  – Inquiry screen (TUFINQ)
  – Sign off screen (TUFSGOFF)
  – Reset screen (TUFRESET)
- Determine the CICS region where the adapter programs will execute.
- Determine the CICS region where the FEPI program will execute to access the back-end system.

After some analysis, we determine that the host environment for the deployed adapter will look like Figure 81 on page 115. In this host environment, the generated adapter programs, TUFNAV and its associated subflow programs TUFSGON, TUFSGOFF, TUFPRSER, TUFRESET, and TUFINQ execute in CICS region QAS1. CMAV acts as the back-end system in the CICS region DEV2.

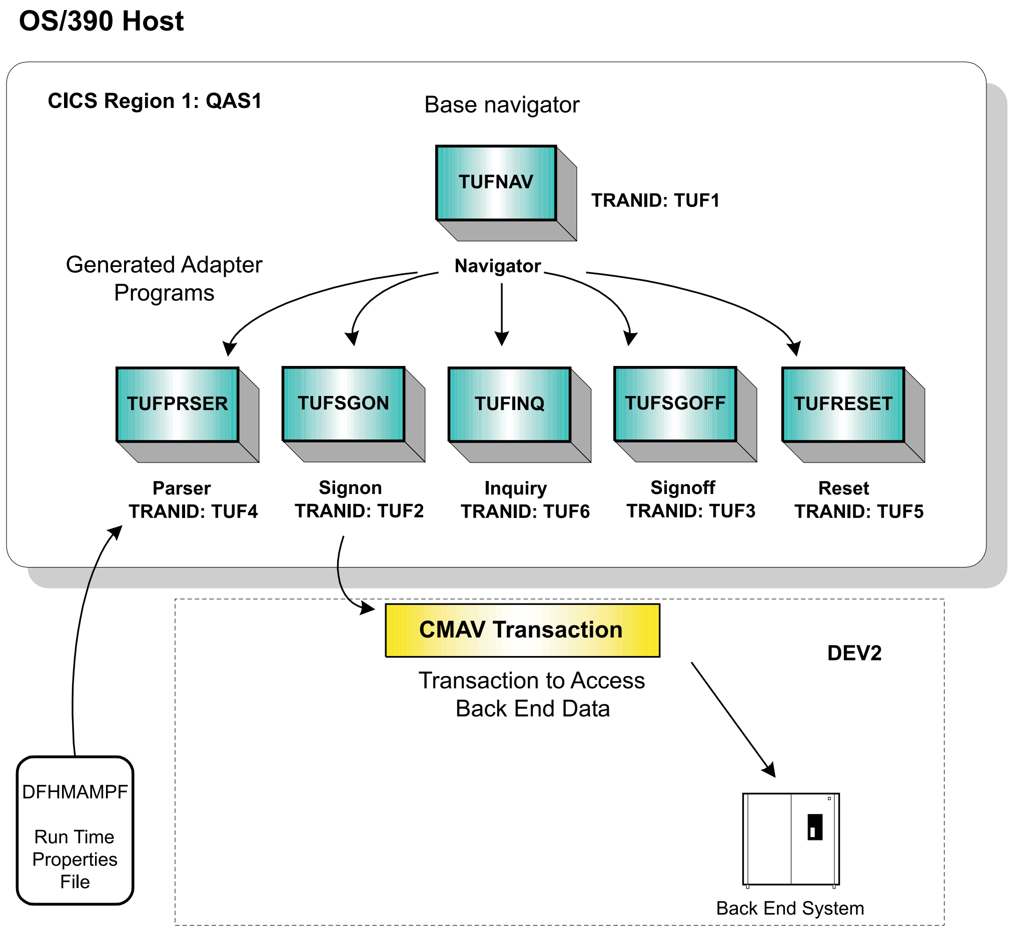


Figure 81. Tutorial run time environment for FEPI adapter

Accessing the FEPI tutorial files

The files you will need in order to build and deploy an adapter that supports a FEPI interface are located in two directories as follows:

- C:\<mqiac_tutorials>\fepi
- C:\<mqiac_base>\cics

In the **C:\<mqiac_tutorials>\fepi** directory you will find the following files:

Table 40. Files in the C:\<mqiac_base>\cics directory

File name	Description	Use
TU_F_RDS.cbl	COBOL record description.	Used as import for messages. Contains the message structure.
TC_FEPI_WS.zip	Completed workspace for FEPI adapter.	A completed workspace that you import and use as the basis for the workspace used to create the FEPI adapter. See "Accessing a completed workspace" on page 12 for information on using the contents of this file

Table 40. Files in the C:\<mqiac_base>\cics directory (continued)

File name	Description	Use
*.cpy	Generated copybooks.	The generated copybooks for the FEPI adapter

In the **C:\<mqiac_base>\cics** directory you will find the following files:

Table 41. Files in the C:\<mqiac_base>\cics directory

File name	Description	Use
tu_f_fepiinteraction.ispec	Interaction specification file	Identifies the use of the FEPI command.
tu_f_nav.rsc	Connector resource file	Specifies synchronous rollback, Navigator type, COBOL program name for the FEPI adapter and the CICS TransID.
tu_f_INQfepi.rsc	Connector resource file	Specifies synchronous rollback, FEPI Navigator type, program name for the FEPI adapter, CICS TransID, logoff type, FEPI pool, FEPI target, FEPI conversation status, timeout value, maximum Commarea length and FEPI PassTicket status.
tu_f_PRSERfepi.rsc	Connector resource file	FEPI Navigator. See description for tu_f_INQfepi.rsc
tu_f_RESETfepi.rsc	Connector resource file	FEPI Navigator. See description for tu_f_INQfepi.rsc
tu_f_SGONfepi.rsc	Connector resource file	FEPI Navigator. See description for tu_f_INQfepi.rsc
tu_f_SGOFFfepi.rsc	Connector resource file	FEPI Navigator. See description for tu_f_INQfepi.rsc
Note: There is also a version of the Specification files prefixed with *tc_f_* that are used for the completed workspace supplied in the TC_FEPI_WS.zip file.		

Configuring the Specification Files

GOAL

In this section you will learn how to configure the physical properties of each FEPI adapter component (the base microflow and the subflows). These properties represent the XML definitions that are sent to the Properties file on the host at deployment time.

For information on the Properties file, see the MQSI Agent for CICS run time documentation.

Specification files are XML-format files that provide specific values to certain components created in MQSI Agent for CICS. An *Interaction Specification* file provides unique values for the component to which it is assigned. A *Connector Resource* file provides more general values for the component.

Some of the information in the Interaction Specification file and Connector Resource file maps to a run time properties file, DFHMAMPF. Other information in the Interaction Specification file is incorporated in generated Command and Navigator programs. The DFHMAMPF stores data that is needed to run the generated adapter code programs on the host. See the MQSI Agent for CICS run time documentation for information on DFHMAMPF.

The FEPI adapter requires the following specification files:

- Interaction Specification file (TU_F_fepiinteraction.ispec) for the FEPI Command type
- Connector Resource file (TU_F_NAV.rsc) for the Microflow type (adapter Navigator)
- Connector Resource files for the FEPI Microflow Types
  - Inquiry subflow (TU_F_INQfepi.rsc)
  - Parser subflow (TU_F_PRSERfepi.rsc)
  - Reset subflow (TU_F_RESETfepi.rsc)
  - Signon subflow (TU_F_SGONfepi.rsc)
  - Signoff subflow (TU_F_SGOFFfepi.rsc)

The specification files are located in the `<mqiac_base>/cics` directory. You **must** configure the settings in the specification files used for the tutorial.

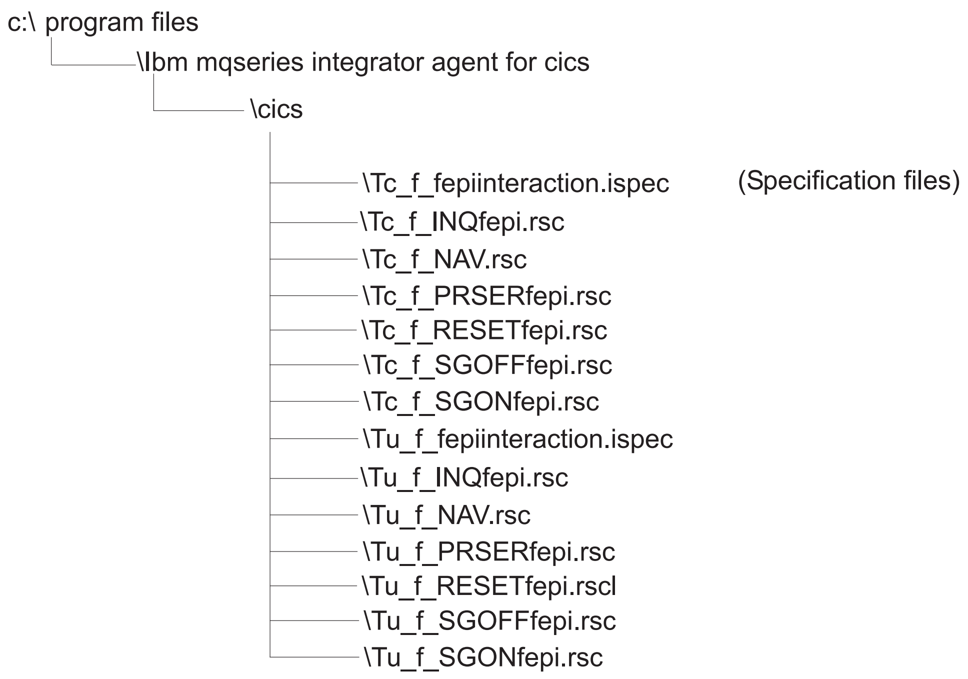


Figure 82. Directory structure for locating specification files for the FEPI interface

The FEPI Command type uses an Interaction Specification file. In the Interaction Specification file (`TU_F_fepiinteraction.ispec`), the MAT_CMDTYPE identifies the type of command. In the example, the MAT_CMDTYPE variable has a value of MAT_FEPI.

Build an adapter that supports a FEPI interface

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Interaction Specification">
   <Attribute xmi.label="MAT_CMDTYPE"  type="MAT_DPL MAT_MQ MAT_FEPI"
xmi.uuid="" valueMandatory="true" value="MAT_FEPI" encoded="false"/>
</AttributeGroup>
```

The Connector Resource file for the base Navigator Microflow type used in the tutorial is `tu_f_nav.rsc`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Connector Resource">
   <Attribute xmi.label="MAT_REQTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="0" encoded="false"/>
   <Attribute xmi.label="MAT_NAVTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="R" encoded="false"/>
   <Attribute xmi.label="MAT_PROGID" type="" xmi.uuid="" valueMandatory="false"
         value="TUFNAV" encoded="false"/>
   <Attribute xmi.label="MAT_TRANID" type="" xmi.uuid="" valueMandatory="false"
         value="TUF1" encoded="false"/>
</AttributeGroup>
```

Table 42. Keyword values used for base Navigator Microflow Connector Resource file

Keyword Symbolic	Description / Use	Example Value
MAT_REQTYPE	Specifies whether the request is run on the server in asynchronous, synchronous or synchronous rollback mode 0 (asynchronous) 1 (synchronous) 2 (synchronous rollback)	0
MAT_NAVTYPE	Specifies whether the Microflow Type is a base Navigator (R) or a FEPI Navigator (F)	R
MAT_PROGID	The name of the COBOL program generated for the FEPI Navigator microflow.	TUFNAV
MAT_TRANID	The CICS TransID for the server command program generated on the server.	TUF1

The Connector Resource file for the base FEPI Microflow type for the Inquiry subflow used in the tutorial is `TU_F_INQfepi.rsc`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE AttributeGroup SYSTEM "mqsi.dtd">
<AttributeGroup xmi.label="Connector Resource">
   <Attribute xmi.label="MAT_REQTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="0" encoded="false"/>
   <Attribute xmi.label="MAT_NAVTYPE" type="" xmi.uuid="" valueMandatory="false"
         value="F" encoded="false"/>
   <Attribute xmi.label="MAT_PROGID" type="" xmi.uuid="" valueMandatory="false"
         value="TUFINQ" encoded="false"/>
   <Attribute xmi.label="MAT_TRANID" type="" xmi.uuid="" valueMandatory="false"
         value="TUF6" encoded="false"/>
   <Attribute xmi.label="MAT_LOGOFFTYPE" type="" xmi.uuid="" valueMandatory="false"
value="A" encoded="false" />
   <Attribute xmi.label="MAT_POOL" type="" xmi.uuid="" valueMandatory="false"
value="CICSDEV2" encoded="false" />
   <Attribute xmi.label="MAT_TARGET" type="" xmi.uuid="" valueMandatory="false"
value="" encoded="false" />
   <Attribute xmi.label="MAT_CONVERSE" type="" xmi.uuid="" valueMandatory="false"
```

```
value="Y" encoded="false" />
   <Attribute xmi.label="MAT_TIMEOUT" type="" xmi.uuid="" valueMandatory="false"
value="025" encoded="false" />
   <Attribute xmi.label="MAT_MAXCALEN" type="" xmi.uuid="" valueMandatory="false"
value="400" encoded="false" />
   <Attribute xmi.label="MAT_USELUPASS" type="" xmi.uuid="" valueMandatory="false"
value="Y" encoded="false" />
 </AttributeGroup>
```

Table 43. Keyword values used for a FEPI Connector Resource file

Keyword Symbolic	Description / Use	Example Value
MAT_REQTYPE	Specifies whether the request is run on the server in asynchronous, synchronous or synchronous rollback mode 0 (asynchronous) 1 (synchronous) 2 (synchronous rollback)	0
MAT_NAVTYPE	Specifies whether the Microflow Type is a base Navigator (R) or a FEPI Navigator (F)	F
MAT_PROGID	The name of the COBOL program generated for the FEPI command.	TUFINQ
MAT_TRANID	The CICS TransID for the server command program generated on the server.	TUF6

MAT_LOGOFFTYPE	Used to set the state of the conversation upon exit of a FEPI microflow. Valid values are:
	R = release
	A = assign
	P = pass
	OVERRIDE
		A

MAT_POOL	Specifies the name of the FEPI Pool from which connections will be attached. Use OVERRIDE to dynamically set the Pool property during run time.	CICSDEV2
MAT_TARGET	Specifies the FEPI Target, that is the back-end region with which FEPI will communicate. Use OVERRIDE to dynamically set the Target property during run time.	no value specified (blank)
MAT_CONVERSE	Indicates that the FEPI conversation to the back-end region is conversational.	Y
MAT_TIMEOUT	Amount of time, in seconds, before a FEPI Receive from the back-end will terminate with a timeout condition. Use OVERRIDE to dynamically set the Timeout property during run time.	025
MAT_MAXCALEN	Specifies the maximum Commarea length for the MAX_LINKNAME program.	400

Table 43. Keyword values used for a FEPI Connector Resource file (continued)

Keyword Symbolic	Description / Use	Example Value
MAT_USELUPASS	Switch to indicate whether a FEPI microflow should generate a UserID and PassTicket for signing on to the back-end region.	Y

The other FEPI subflows use the same Connector Resource values with the exception of the MAT_PROGID and the MAT_TRANID values.

Table 44. Program and transaction IDs values used for the FEPI microflows

FEPI Microflow Type	Resource file	MAT_PROGID	MAT_TRANID	MAT_LOGOFFTYPE
Signon	TU_F_SGONfepi.rsc	TUFSGON	TUF2	A
Signoff	TU_F_SGOFFfepi.rsc	TUFSGOFF	TUF3	OVERRIDE
Parser	TU_F_PRSERfepi.rsc	TUFPRSER	TUF4	A
Reset	TU_F_RESETfepi.rsc	TUFRESET	TUF5	A
Inquiry	TU_F_INQfepi.rsc	TUFINQ	TUF6	A

You have just completed the steps necessary to configure the Properties file. You are now ready to import the required screens and message sets into the adapter builder.

Creating an adapter that supports a CICS FEPI interface

In this section you will learn how to import the necessary COBOL record descriptions and system interfaces for the FEPI adapter. These are stored in the logical message model in the Adapter Builder for use in the FEPI adapter flow.

Import Message Sets

Follow these instructions to begin the process of building an adapter that supports a FEPI interface:

__ Step 1. **Start the builder and create a new workspace**.

To start the builder, go to the **Start** > **Programs** > **IBM MQSI Agent for CICS** >**IBM MQSI Agent for CICS**. This will launch the tool as shown below, in Figure 83 on page 121.

Build an adapter that supports a FEPI interface

You should begin the tutorial with a new workspace. A workspace is a view of what you can work with at one time. A workspace is displayed as the graphical space in the builder where you will build the adapter to support the FEPI interface.

From the **File** pull-down menu, select **New Workspace**.

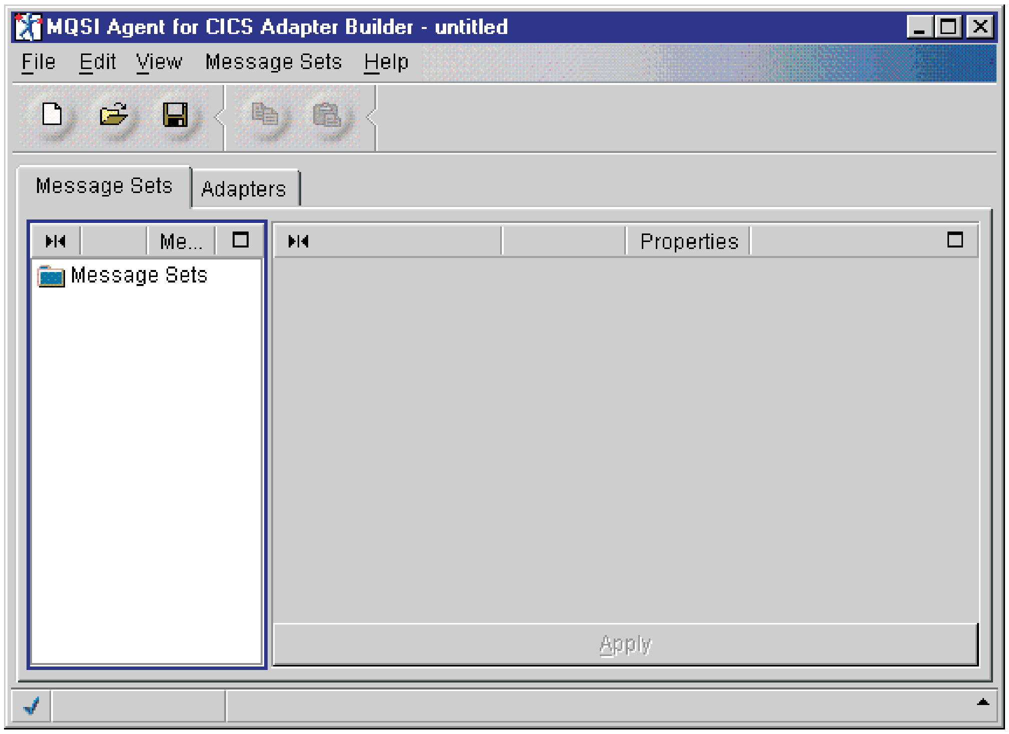


Figure 83. Initial panel of the MQSI Agent for CICS Adapter Builder

__ Step 2. **Name your tutorial workspace and save it to the repository**.

From the **File** pull-down menu, select **Save Workspace**. Enter a name for the workspace, such as TU_FEPI_WS, and click **Save**

Note: Be sure to use under_scores and not dashes "-" when naming the workspace.

.

__ Step 3. **Import message sets**.

A message set is a collection of structured XML-based data types that are stored in the message repository.

When you import a message set, what you are really doing is bringing in the COBOL structured data type definitions from existing CICS transactions and the 3270 screen interactions with the host system, into the Adapter Builder's control center. The control center utilizes the message set as an interface between the adapter builder tool and the business transaction to be modelled.

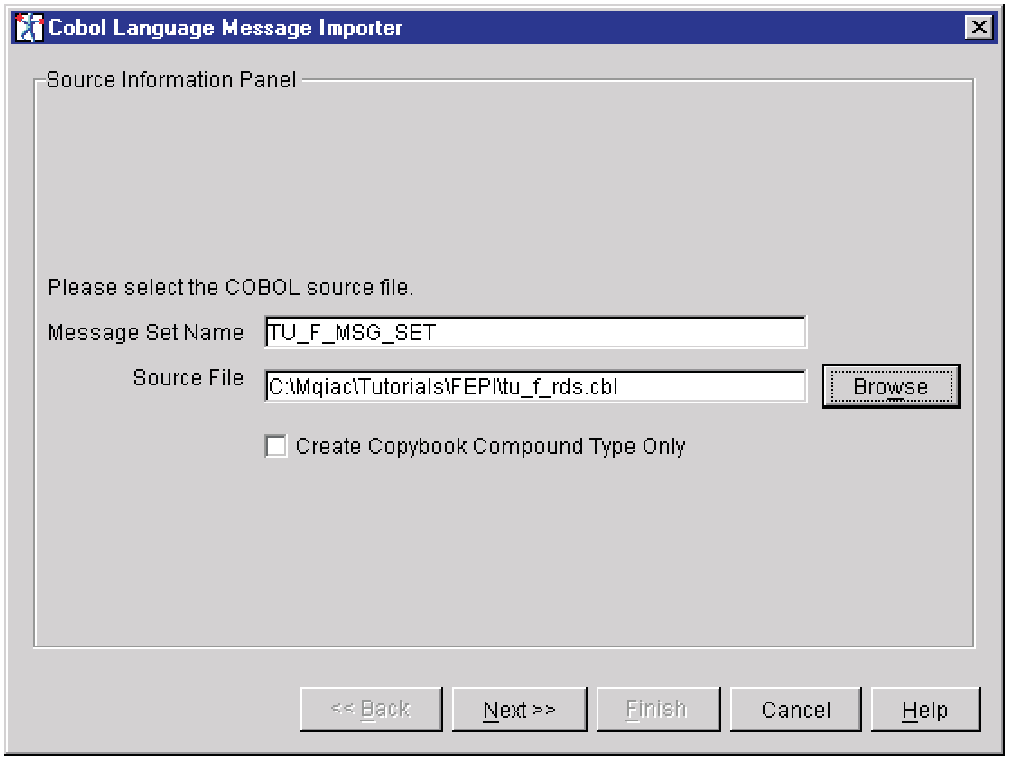
After importing a message, you can modify and store it.

Note: It is much easier to import a COBOL structured data type definition than it is to build the message set. If there is no record description, create one with a text editor and import it.

This tutorial uses the following message sets:

- TU_F_MSG_SET — Basic adapter messages. These include the following:
  - TU_F_RAW — The input record description from the controlling application.
  - TU_F_DEC — The Decision node message.
  - TU_F_REPLY — The Response message.
- CICS_SAMPLES — Standard CICS sample messages. These include:
  - CICS Request and Response
  - System FEPI overrides
  - System LU LOGON
- TU_F_3270_MSG_SET — CICS captured screen messages and screen layouts.

a. Right click on the Message Sets folder, select **Import to New Message Set > COBOL**.



Please select the COBOL source file.

Message Set Name TU_F_MSG_SET

Source File C:\Mqiac\Tutorials\FEPI\tu_f_rds.cbl

☐ Create Copybook Compound Type Only

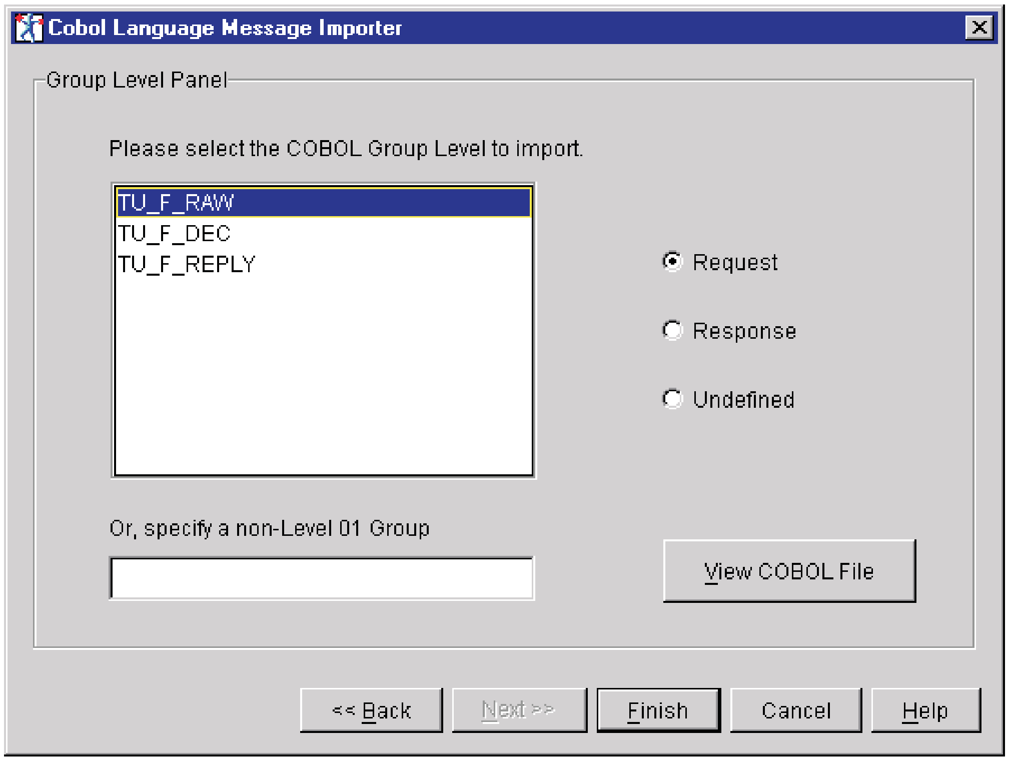
Figure 84. Import a message set (source information)

On the COBOL Language Message Importer dialog (Source Information Panel), enter the Message Set Name (in the tutorial, TU_F_MSG_SET) and the directory path where the Source Files for the copybooks are located (**<mqiac_tutorials>\fepi\tu_f_rds.cbl**).

For the purposes of this tutorial, leave the**Create Copybook Compound Type Only** box unchecked. This box is an option that controls *how* copybooks can be imported.

b. Click **Next**.

c. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import (for the tutorial, select TU_F_RAW) and select the Request message type radio button.



Cobol Language Message Importer

Group Level Panel

Please select the COBOL Group Level to import.

TU_F_RAW
TU_F_DEC
TU_F_REPLY

(●) Request
( ) Response
( ) Undefined

Or, specify a non-Level 01 Group

Figure 85. Import a message set (group level)

The radio button selections are as follows:

Request
Use if the message is going to be used as an input message in a transaction.

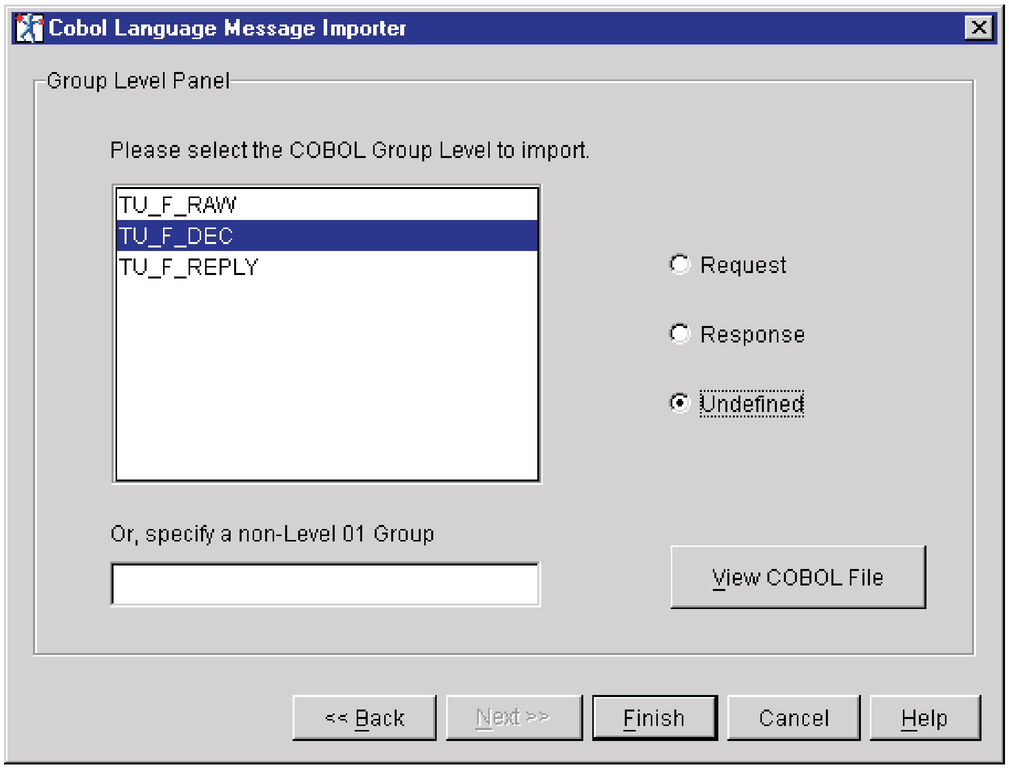
Response
Use if the message is going to be used as an output message in a transaction.

Undefined
Can be used for messages that are not used in a transaction.

Click **Finish** to complete the import.

d. Right click on the newly created TU_F_MSG_SET folder and select **Import to Message Set > COBOL**. On the COBOL Language Message Importer dialog (Source Information Panel), enter the directory path where the Source Files for the copybooks are located (**<mqiac_tutorials>\fepi\tu_f_rds.cbl**). See Figure 84 on page 122). Click **Next**.

e. On the COBOL Language Message Importer dialog (Group Level Panel), select the message to import (for the tutorial, select TU_F_DEC) and select the Undefined message type radio button.



Please select the COBOL Group Level to import.

TU_F_RAW
TU_F_DEC
TU_F_REPLY

○ Request
○ Response
● Undefined

Or, specify a non-Level 01 Group

Figure 86. Import a message set (group level)

Click **Finish** to complete the import.

f. Repeat the procedure in steps c and d until all of the following messages (with the specified message types) are imported:

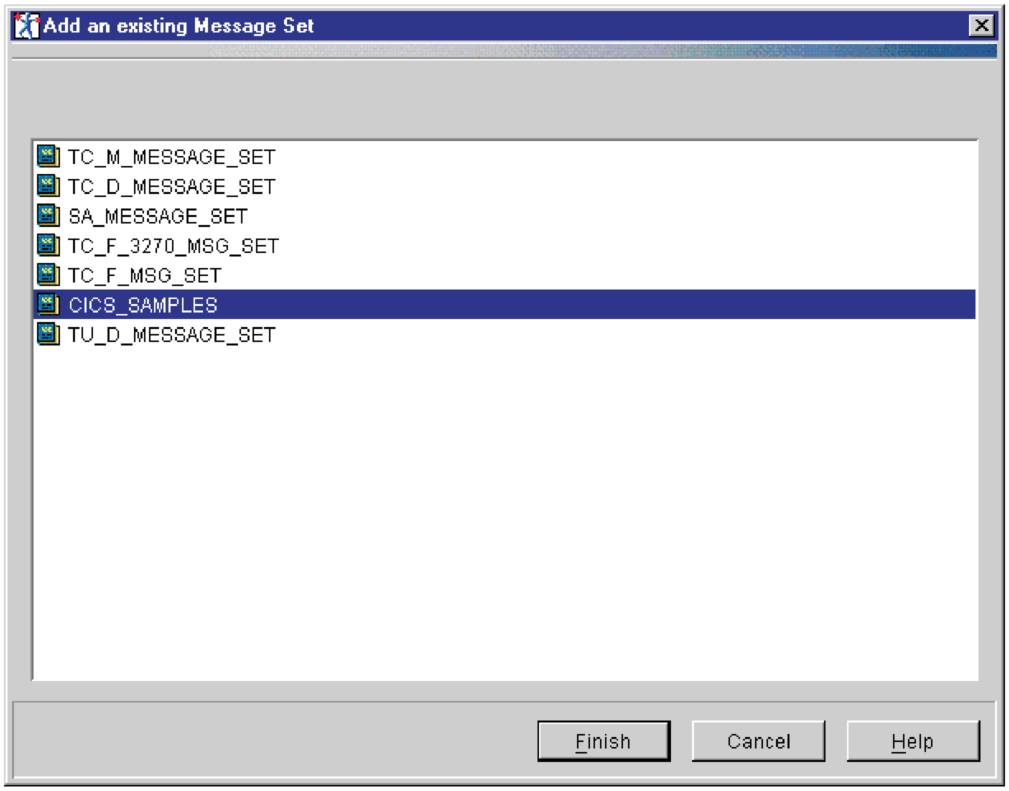
Table 45. Messages to add to the workspace

Message	Message Type	Purpose
TU_F_RAW	Request	Input message used by the navigator to receive information from the controlling application.
TU_F_DEC	Undefined	Decision node message used by the navigator to determine how to flow logically within the flow. This message provides a series of fields, the context of which are evaluated by the Navigator to control logical flow.
TU_F_REPLY	Response	Message used to receive customer demographic information as provided by the back end application.

g. Next, you need to add the CICS_SAMPLES message set to your workspace. Right click on the Message Sets folder and select **Add to Workspace > Message Set**.

The CICS_SAMPLES message set will be utilized later on in this tutorial to assist with screen recognition and transaction implementation issues in overrides and passticket processing.

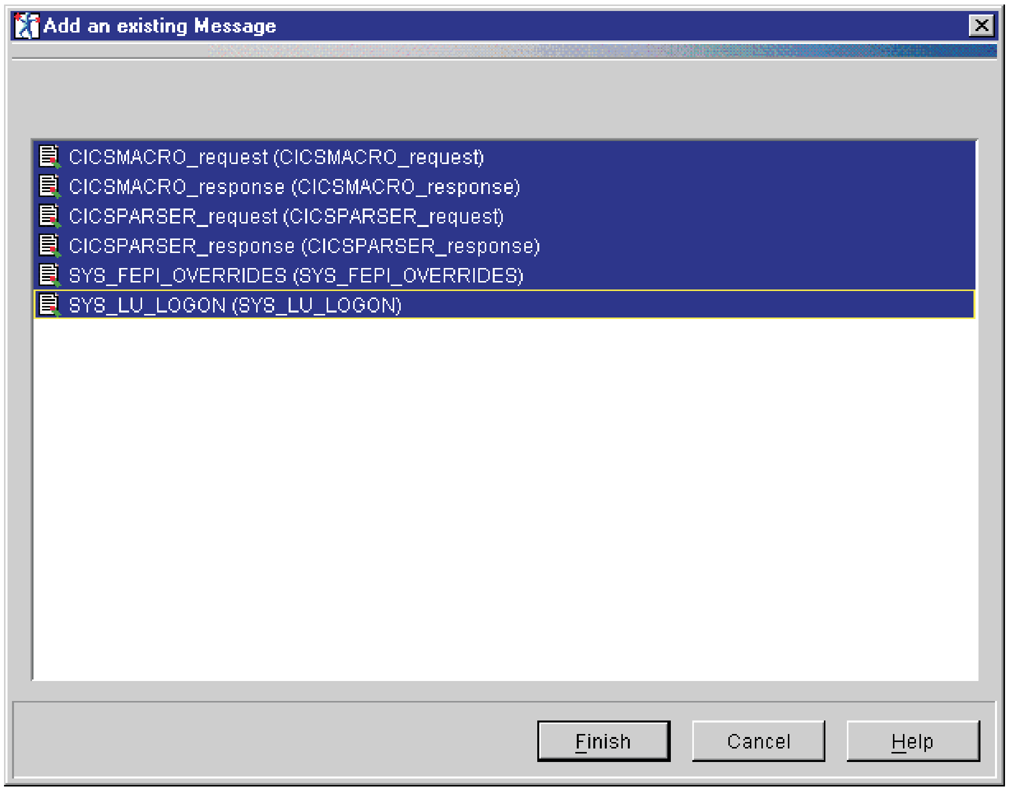
h. Select the CICS_SAMPLES message set in the Add an Existing Message Set window and click **Finish**.



TC_M_MESSAGE_SET
TC_D_MESSAGE_SET
SA_MESSAGE_SET
TC_F_3270_MSG_SET
TC_F_MSG_SET
CICS_SAMPLES
TU_D_MESSAGE_SET

Figure 87. Add the CICS_SAMPLES message set

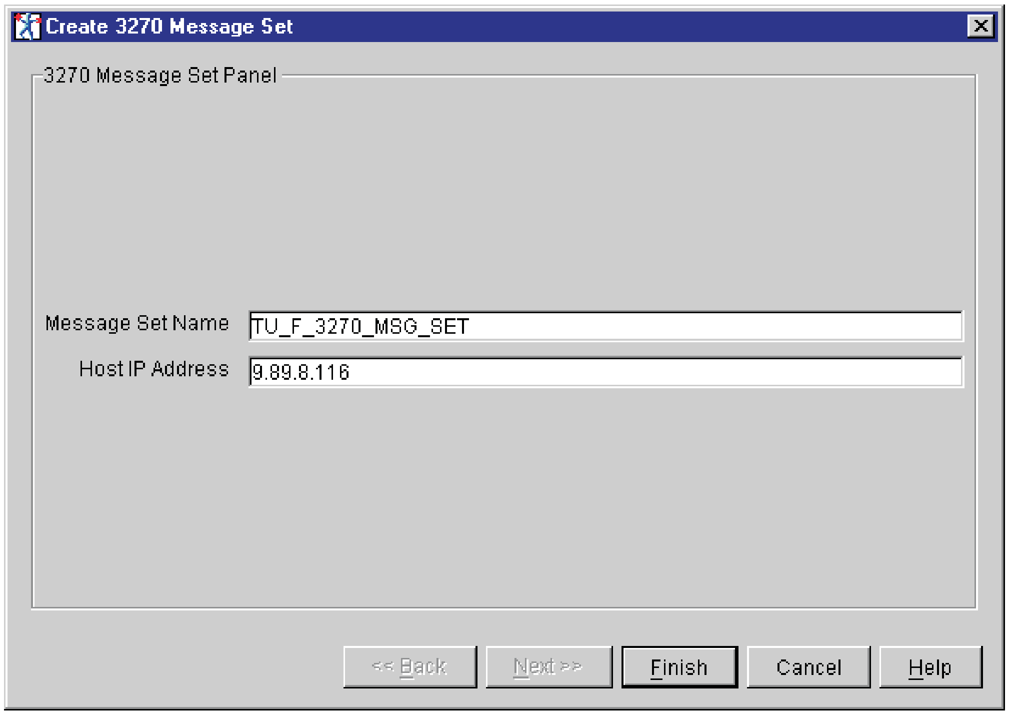
i. Right click on the CICS_SAMPLES message set folder and select **Add to Workspace > Message**. Select all the messages in the Add an Existing Message window (click on each message while you press the CTRL key) and click **Finish**.



CICSMACRO_request (CICSMACRO_request)
CICSMACRO_response (CICSMACRO_response)
CICSPARSER_request (CICSPARSER_request)
CICSPARSER_response (CICSPARSER_response)
SYS_FEPI_OVERRIDES (SYS_FEPI_OVERRIDES)
SYS_LU_LOGON (SYS_LU_LOGON)

Figure 88. Add messages to the CICS_SAMPLES message set

j. Finally, you need to create a message set to hold imported 3270 messages. Right click on the Message Sets folder, select **Import to New Message Set > 3270**. On the Create 3270 Message Set dialog, enter the Message Set Name (in the tutorial, TU_F_3270_MSG_SET) and the Host IP Address. The Host IP Address you enter should correspond to the host system you are accessing. Click **Finish**.



Message Set Name: TU_F_3270_MSG_SET
Host IP Address: 9.89.8.116

Figure 89. Create the 3270 message set

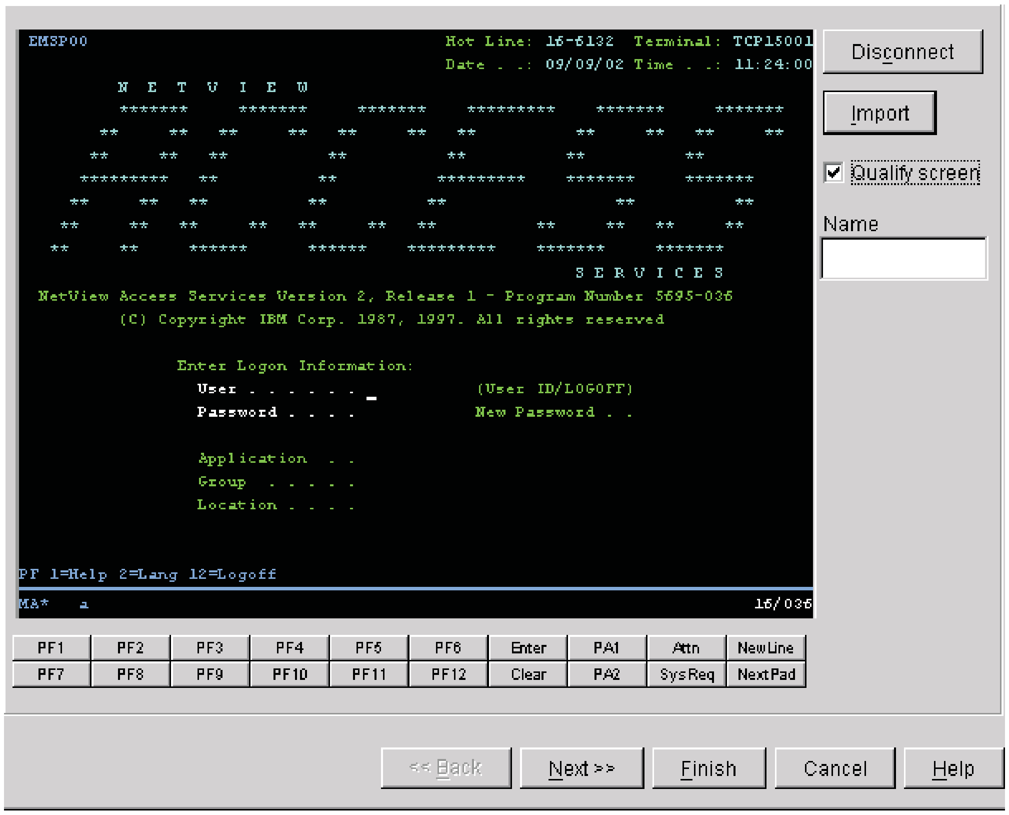
You have just completed importing the required message sets using the COBOL importer.

You are now ready to import the 3270 screens to the message sets.

Importing Screens

a. Right click on the TU_F_3270_MSG_SET message set folder and select **Import to Message Set > 3270**.

b. On the 3270 Screen Importer dialog, click **Connect** to connect to the Host. This will put you in 3270 emulation mode.

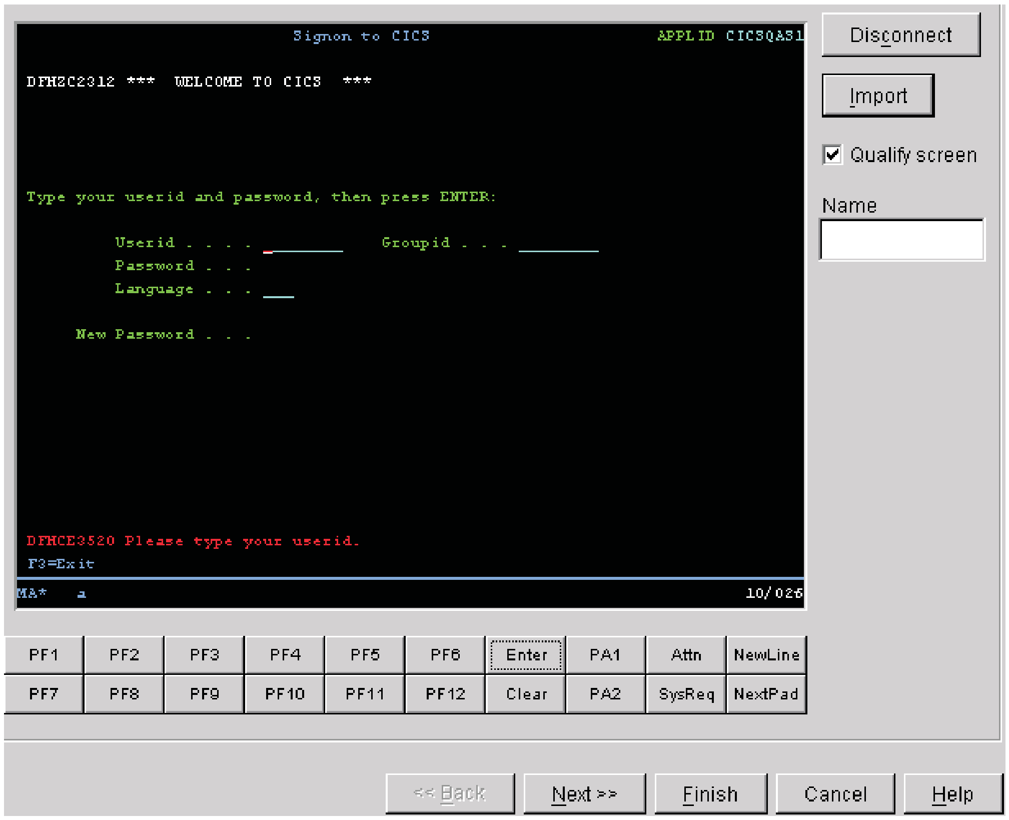


```
 EMSP00                                    Hot Line: 16-6132  Terminal: TCP15001
                                           Date . .: 09/09/02 Time . .: 11:24:00
          N  E  T  U  I  E  W
          *******    *******    *******    *********    *******    *******
        **     **   **     **   **     **   **          **     **   **     **
       **     **   **          **          **          **          **
      *********   **          **          *********    *******    *******
     **     **   **          **          **                  **          **
    **     **   **     **   **     **   **          **     **   **     **
   **     **    ******     ******     *********    *******    *******
                                                       S E R U I C E S
  NetView Access Services Version 2, Release 1 - Program Number 5695-036
          (C) Copyright IBM Corp. 1987, 1997. All rights reserved

                Enter Logon Information:
                  User . . . . . . _          (User ID/LOGOFF)
                  Password . . . .            New Password . .

                  Application  . .
                  Group  . . . . .
                  Location . . . .


PF 1=Help 2=Lang 12=Logoff
MA*   a                                                                  16/036
```

Figure 90. 3270 Screen Importer in emulation mode

c. Logon to the CICS region where the installation verification back-end programs were installed.

Note: You should use the Userid and Password that you have been assigned to access your host system.

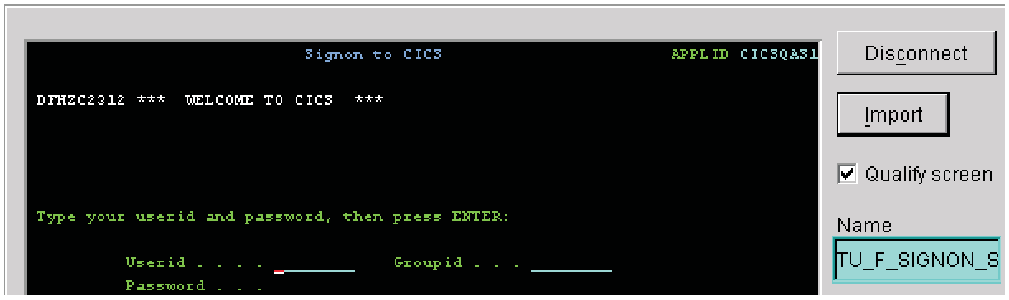


```
                          Signon to CICS                        APPLID CICSQAS1

DFHZC2312 ***  WELCOME TO CICS  ***



Type your userid and password, then press ENTER:

        Userid . . . . ________    Groupid . . . ________
        Password . . .
        Language . . . ___

    New Password . . .








DFHCE3520 Please type your userid.
F3=Exit
MA*   a                                                                10/026
```

Figure 91. CICS logon screen

d. Perform a screen capture for the CICS logon screen. Enter TU_F_SIGNON_SCR in the Name field and click the **Import** button.

Note: No Userid or Password values are entered in the screen before the screen capture because these values will be supplied using the FEPI PassTicket feature in the tutorial.



```
                          Signon to CICS                        APPLID CICSQAS1

DFHZC2312 ***  WELCOME TO CICS  ***



Type your userid and password, then press ENTER:

        Userid . . . . ________    Groupid . . . ________
        Password . . .
```

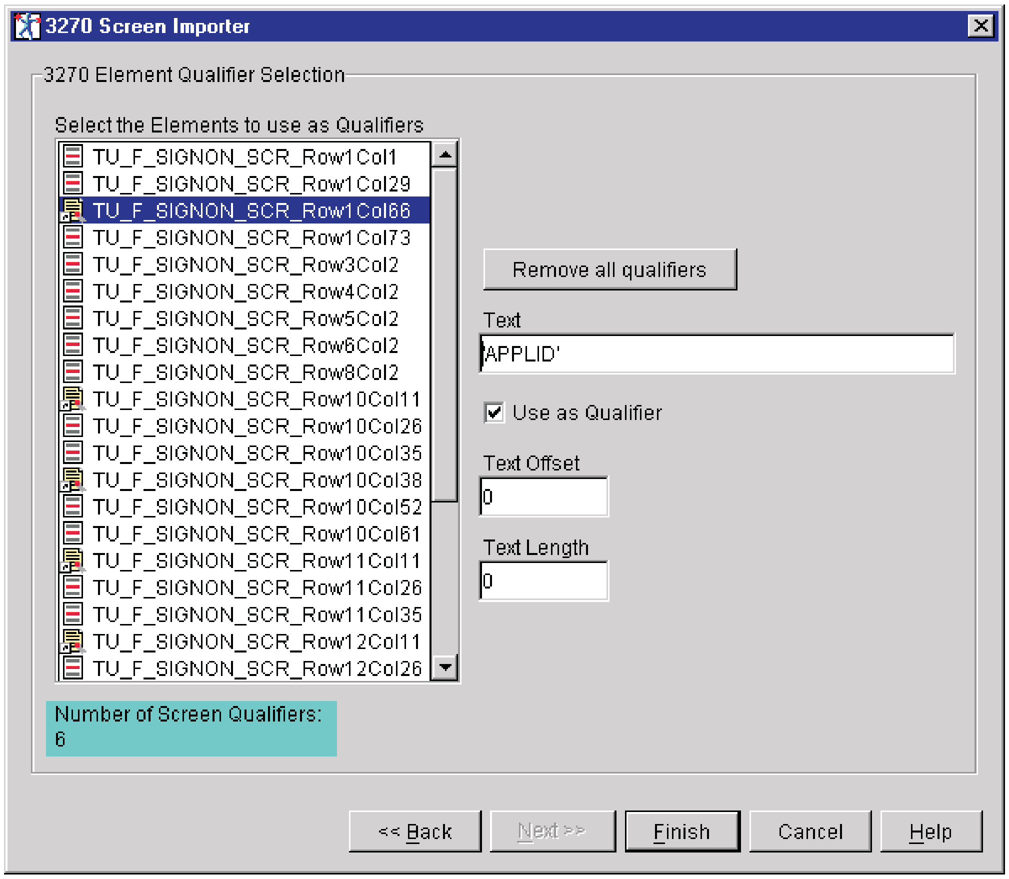
Figure 92. Capturing the CICS logon screen

When you import a screen, the builder uses an algorithm to select the best elements to represent the screen layout. Each field on a screen is imported as an element. In addition, an element qualifier is created for each field that was chosen by the algorithm to use for screen identification.

You can view the element qualifiers that were created for the CICS Login screen (TU_F_SIGNON_SCR) by clicking the **Next** button (see

Figure 93). Notice that there are 6 Screen Qualifiers specified. Select the TU_F_SIGNON_SCR_Row1Col66 qualifier which indicates the screen position (row 1 and column 66). The text for the element qualifier is ’APPLID’ and notice that the Use as Qualifier checkbox is now selected.

You can customize element qualifiers, but for now just click the **Back** button to continue with the 3270 screen import process.


3270 Screen Importer

3270 Element Qualifier Selection

Select the Elements to use as Qualifiers

- TU_F_SIGNON_SCR_Row1Col1
- TU_F_SIGNON_SCR_Row1Col29
- TU_F_SIGNON_SCR_Row1Col66
- TU_F_SIGNON_SCR_Row1Col73
- TU_F_SIGNON_SCR_Row3Col2
- TU_F_SIGNON_SCR_Row4Col2
- TU_F_SIGNON_SCR_Row5Col2
- TU_F_SIGNON_SCR_Row6Col2
- TU_F_SIGNON_SCR_Row8Col2
- TU_F_SIGNON_SCR_Row10Col11
- TU_F_SIGNON_SCR_Row10Col26
- TU_F_SIGNON_SCR_Row10Col35
- TU_F_SIGNON_SCR_Row10Col38
- TU_F_SIGNON_SCR_Row10Col52
- TU_F_SIGNON_SCR_Row10Col61
- TU_F_SIGNON_SCR_Row11Col11
- TU_F_SIGNON_SCR_Row11Col26
- TU_F_SIGNON_SCR_Row11Col35
- TU_F_SIGNON_SCR_Row12Col11
- TU_F_SIGNON_SCR_Row12Col26

Number of Screen Qualifiers: 6

Text: 'APPLID'

[✓] Use as Qualifier

Text Offset: 0

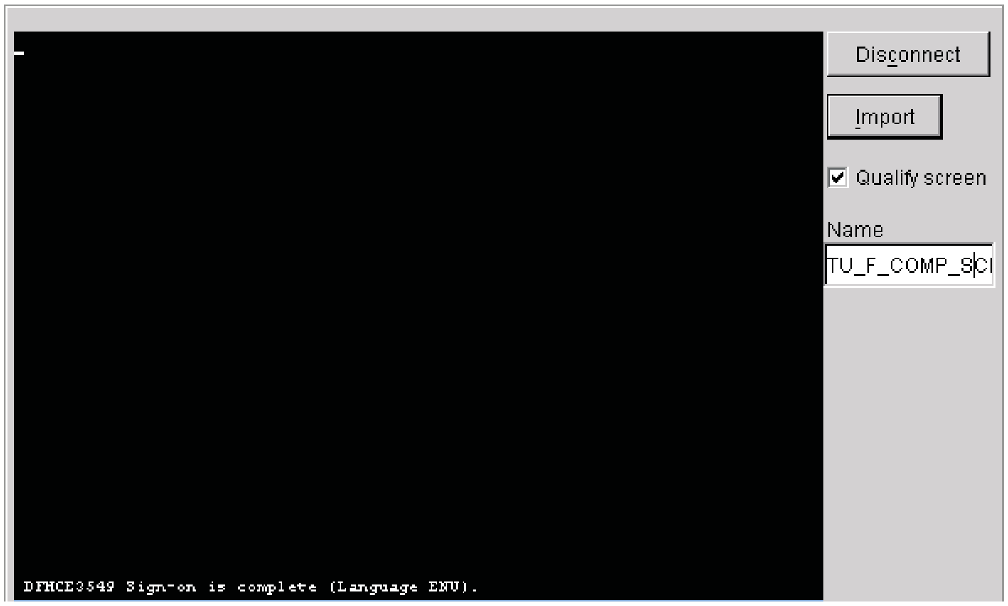
Text Length: 0

Figure 93. Element Qualifiers for the CICS logon screen

e. On the CICS Login screen, click in the Userid field and enter your Userid. Enter your Password in the Password field. Click the **Enter** button to logon. This will display the Signon Complete screen.

f. Make sure that the **Qualify Screen** check box is checked.

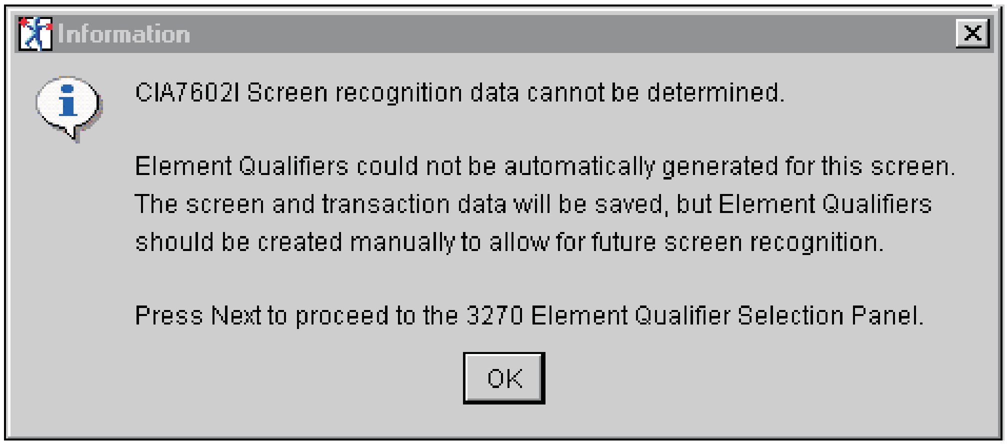
g. Capture the Signon Complete screen. Enter TU_F_COMP_SCR in the Name field and click the **Import** button.



```
DFHCE3549 Sign-on is complete (Language ENU).
```

Figure 94. CICS Signon Complete screen

The information message shown in Figure 95 displays. This message indicates that you need to customize element qualifiers to identify the CICS Signon Complete screen. Click **OK** to close the information message and click the **Next** button.



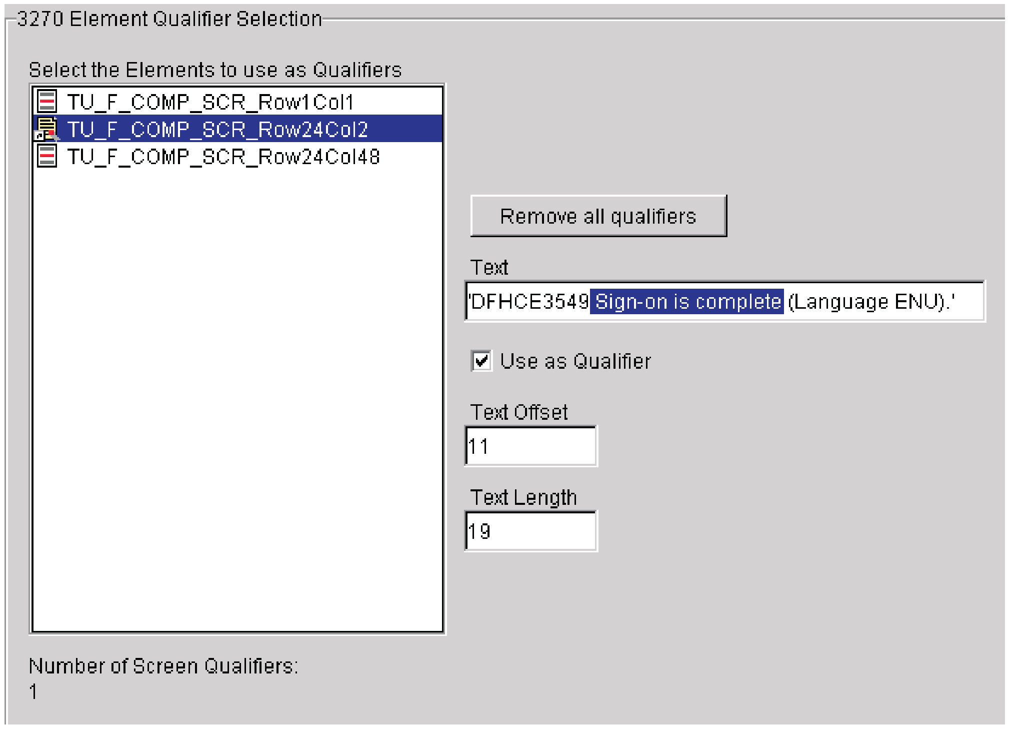
CIA7602I Screen recognition data cannot be determined.

Element Qualifiers could not be automatically generated for this screen. The screen and transaction data will be saved, but Element Qualifiers should be created manually to allow for future screen recognition.

Press Next to proceed to the 3270 Element Qualifier Selection Panel.

Figure 95. 'Screen recognition data cannot be determined' information message

h. Create an element qualifier for the CICS Signon Complete screen.
   1) Select the TU_F_COMP_SCR_Row24Col2 element qualifier.
   2) Highlight the substring "Sign-on is complete" within the Text field. This will be the substring that will be used to identify the CICS Signon Complete screen.
   3) Check the **Use as Qualifier** check box. Notice when the substring is selected the Text Offset field is 11 and Text Length 19.


3270 Element Qualifier Selection

Select the Elements to use as Qualifiers

TU_F_COMP_SCR_Row1Col1
TU_F_COMP_SCR_Row24Col2
TU_F_COMP_SCR_Row24Col48

Text: 'DFHCE3549 Sign-on is complete (Language ENU).'

☑ Use as Qualifier

Text Offset: 11

Text Length: 19

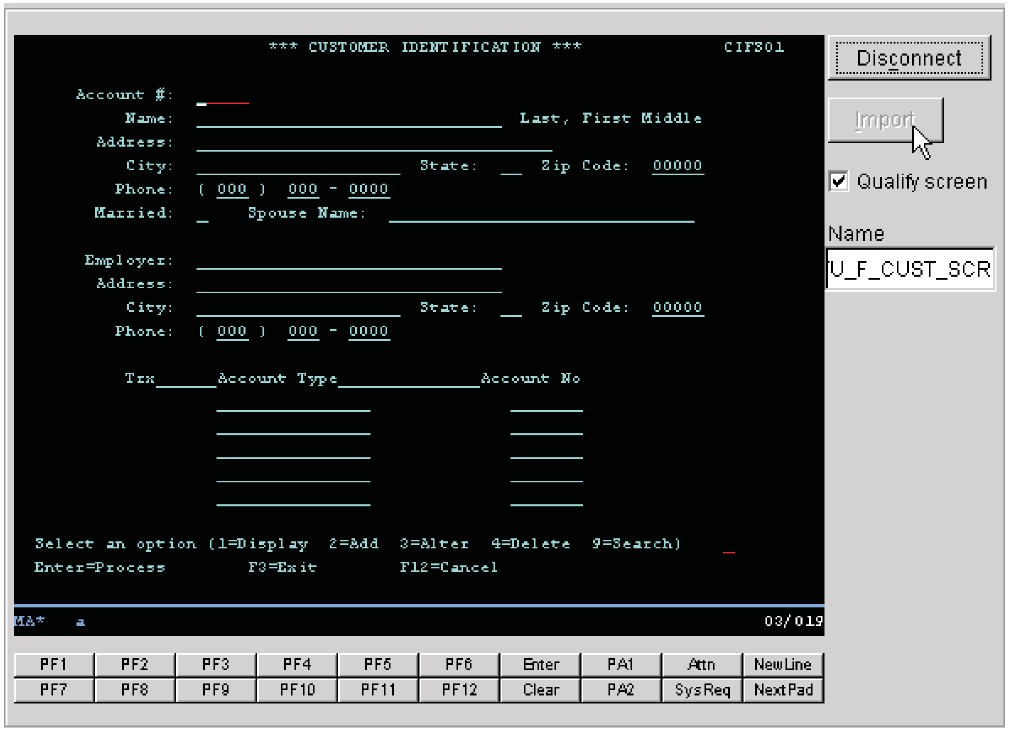
Number of Screen Qualifiers: 1

Figure 96. Customizing the element qualifier

4) Click **Finish**. The Continue Importing 3270 Screen message box will display.
5) Click **Yes** to continue importing screens.

i. Input the following CICS transaction to continue to the Customer Information screen:
   1) Put the cursor focus on the CICS Signon Complete screen.
   2) Click the **Clear** button (located below the screen).
      a) Put the cursor focus on the CICS Signon Complete screen.
   3) Type *CMAV* in the CICS Signon Complete screen.
   4) Click the **Enter** button (located below the screen).
   5) Check the **Qualify Screen** check box.

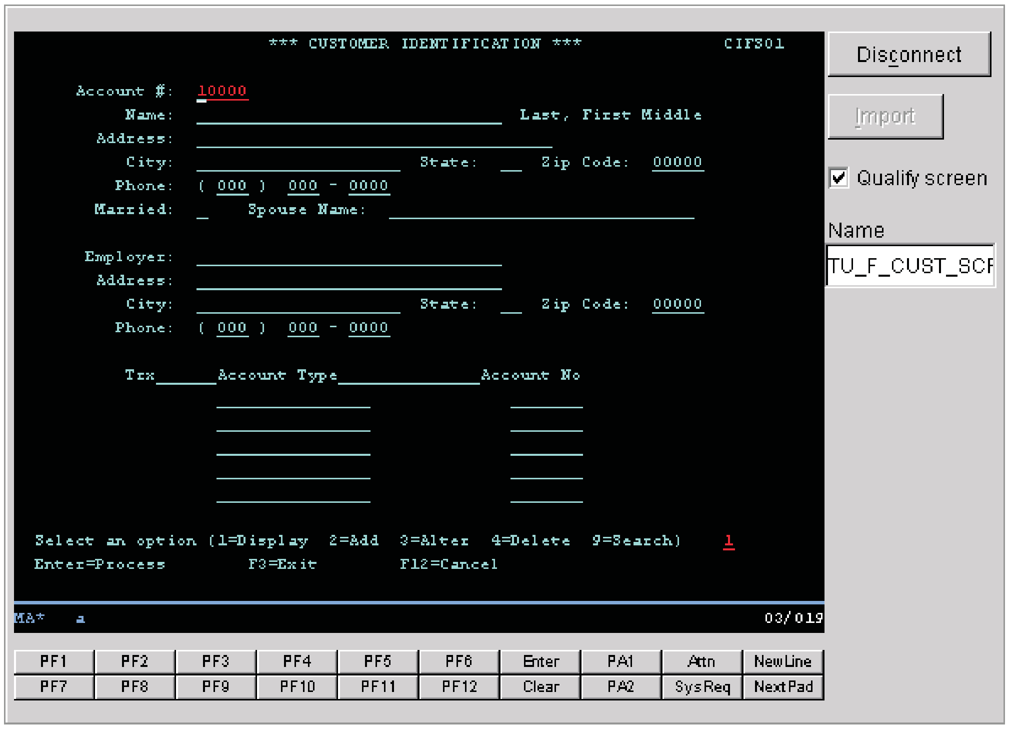
j. Enter TU_F_CUST_SCR in the Name field and click the **Import** button.



```
                    *** CUSTOMER IDENTIFICATION ***             CIFS01

     Account #:  _
        Name:                                      Last, First Middle
     Address:
        City:                              State:     Zip Code:  00000
       Phone:  ( 000 )  000 - 0000
     Married:  _    Spouse Name:

    Employer:
     Address:
        City:                              State:     Zip Code:  00000
       Phone:  ( 000 )  000 - 0000

          Trx______Account Type______________Account No

  Select an option (1=Display  2=Add  3=Alter  4=Delete  9=Search)   _
  Enter=Process        F3=Exit        F12=Cancel

MA*   a                                                          03/019
```

Figure 97. Customer Information screen

k. View a customer record. Enter *10000* in the Account # field, enter *1* in the Select an option field:

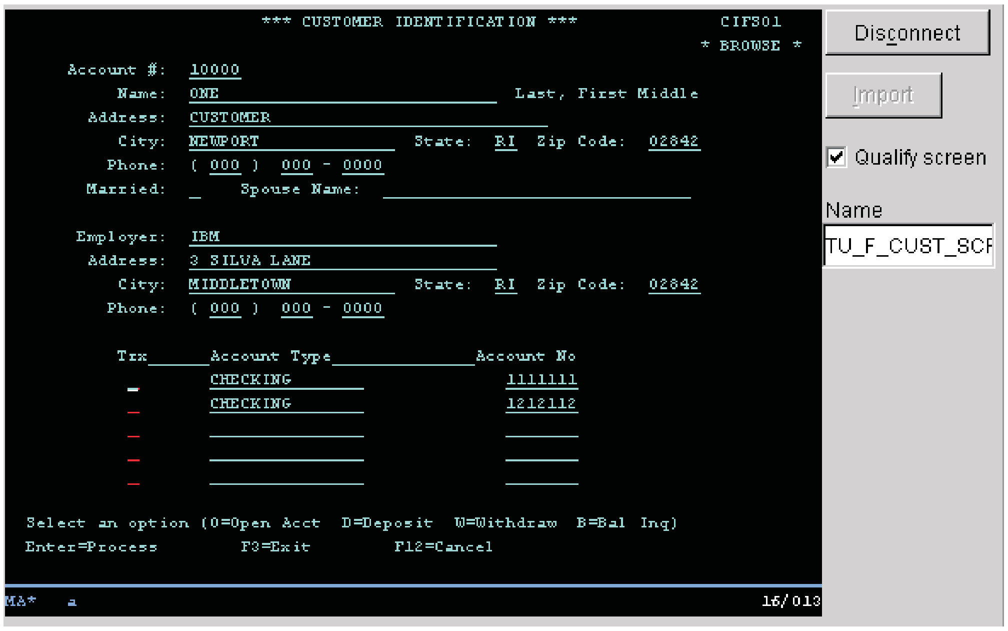


```
                    *** CUSTOMER IDENTIFICATION ***             CIFS01

     Account #:  10000
        Name:                                      Last, First Middle
     Address:
        City:                              State:     Zip Code:  00000
       Phone:  ( 000 )  000 - 0000
     Married:  _    Spouse Name:

    Employer:
     Address:
        City:                              State:     Zip Code:  00000
       Phone:  ( 000 )  000 - 0000

          Trx______Account Type______________Account No

  Select an option (1=Display  2=Add  3=Alter  4=Delete  9=Search)   1
  Enter=Process        F3=Exit        F12=Cancel

MA*   a                                                          03/019
```

Figure 98. Requesting a customer record display on the Customer Information screen

Click **Enter**.

The following record should display.



```
                         *** CUSTOMER IDENTIFICATION ***              CIFS01
                                                                   * BROWSE *
      Account #:  10000
           Name:  ONE____________________________  Last, First Middle
        Address:  CUSTOMER_______________________________
           City:  NEWPORT_______________  State:  RI  Zip Code:  02842
          Phone:  ( 000 )  000 - 0000
        Married:  _     Spouse Name:  ______________________________

       Employer:  IBM____________________________
        Address:  3 SILVA LANE___________________
           City:  MIDDLETOWN______________  State:  RI  Zip Code:  02842
          Phone:  ( 000 )  000 - 0000

           Trx______Account Type______________Account No
            _        CHECKING_______               1111111
            _        CHECKING_______               1212112
            _        _______________               _______
            _        _______________               _______
            _        _______________               _______

  Select an option (O=Open Acct  D=Deposit  W=Withdraw  B=Bal Inq)
  Enter=Process          F3=Exit          F12=Cancel

MA*   a                                                                16/013
```

Figure 99. Customer record display

Before moving on to the next step, you should sign off the host session and disconnect from the 3270 Importer.

From the Customer Identification screen:

- Click PF3 to get to a CICS blank screen
- Type the transaction that signs you off of your host system
- Click **Disconnect** and then click **Finish**.

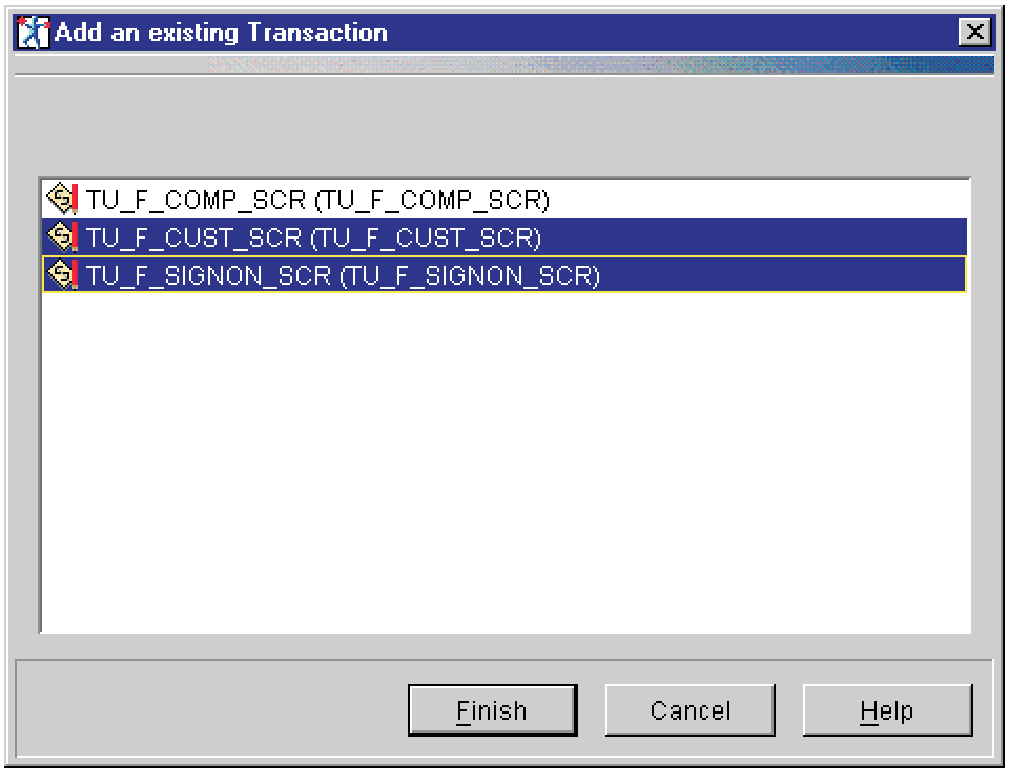
You just completed importing both the COBOL structured data type definitions required to model the FEPI microflow and the 3270 screens that will be used to scrape data from the target back-end system.

__ Step 4. **Add transactions to the workspace.**

a.

A transaction represents the screen recognition, message and data flowing to and from the back-end FEPI screens to be accessed by the adapter. In order to create a FEPI command node, you need to associate the command node with a transaction. The messages associated with the transaction are defined as Input and Output representing the expected format of the input message (and identified as input terminal in the node) and the expected format of the output screen (and identified as the output terminal in the node). Select the Transactions folder under the TU_F_3270_MSG_SET message set. Right click on the Transaction folder. Select **Add to Workspace > Transaction**. Select the TU_F_CUST_SCR and TU_F_SIGNON_SCR transactions and click **Finish**.

A FEPI microflow transaction is a relationship between the content of all the captured back-end application screens. These transactions are then incorporated into the appropriate command nodes.



TU_F_COMP_SCR (TU_F_COMP_SCR)
TU_F_CUST_SCR (TU_F_CUST_SCR)
TU_F_SIGNON_SCR (TU_F_SIGNON_SCR)

Figure 100. Add an existing transaction

b. Customize the system-supplied transactions for use in the microflows. Expand the CICS_SAMPLES message set folder. Right click on the Transaction folder. Select **Add to Workspace > Transaction**. Select the SAMPLE_PARSER and SAMPLE_TRANSACTION transactions and click **Finish**.

   1) Expand the Transactions folder under the CICS_SAMPLES message set. Right click on the SAMPLE_PARSER transaction and click **Copy**.

   2) Right click on the Transactions folder in your TU_F_3270_MSG_SET message set and click **Paste**.

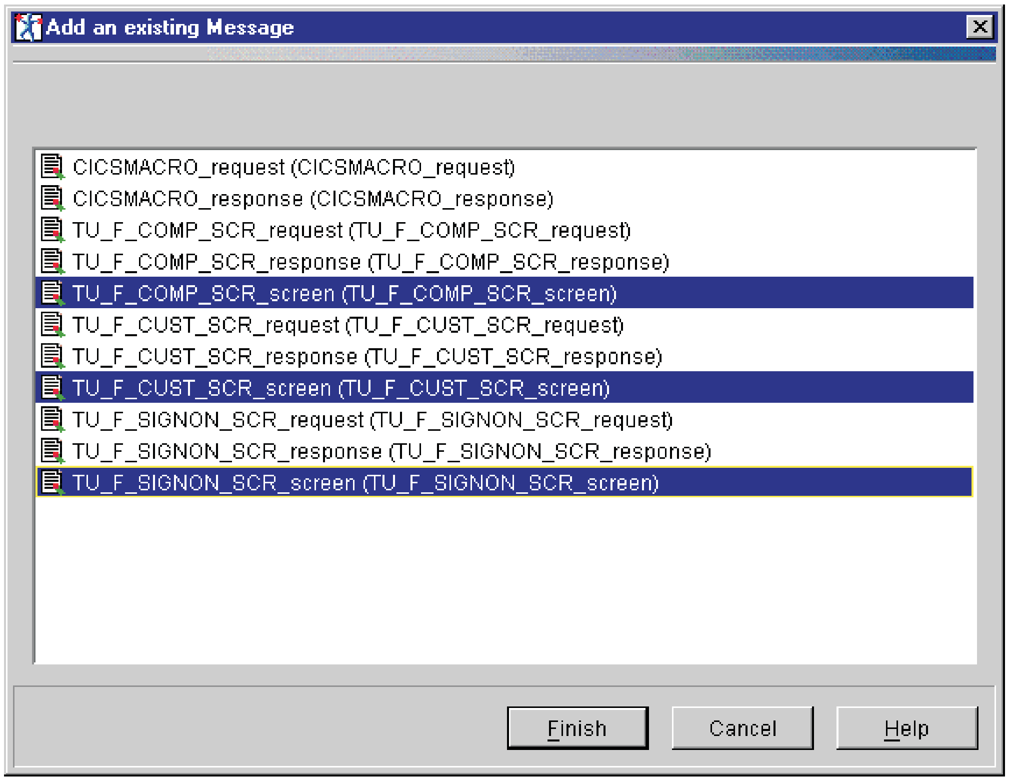
      Note: A warning message stating that ″Any existing elements in this message set will not be overwritten. Are you sure you want to paste this message?″ will appear. Click the **Yes** button.

   You should rename the transaction that you copied to TU_F_SAMPLE_PARSER. Also rename the Identifier field to

TU_F_SAMPLE_PARSER in the Properties pane. Click the **Apply** bar at the bottom of the Properties pane to apply the change.

3) Add the messages representing the imported 3270 screens to the TU_F_SAMPLE_PARSER. These are the screens the parser will attempt to recognize during the adapter processing in order to identify the current user screen.

   Right click on the TU_F_SAMPLE_PARSER transaction and select **Add > Message**. Select the TU_F_SIGNON_SCR_screen, TU_F_CUST_SCR_screen, and TU_F_COMP_SCR_screen messages as depicted in the following figure and click **Finish**.



CICSMACRO_request (CICSMACRO_request)
CICSMACRO_response (CICSMACRO_response)
TU_F_COMP_SCR_request (TU_F_COMP_SCR_request)
TU_F_COMP_SCR_response (TU_F_COMP_SCR_response)
TU_F_COMP_SCR_screen (TU_F_COMP_SCR_screen)
TU_F_CUST_SCR_request (TU_F_CUST_SCR_request)
TU_F_CUST_SCR_response (TU_F_CUST_SCR_response)
TU_F_CUST_SCR_screen (TU_F_CUST_SCR_screen)
TU_F_SIGNON_SCR_request (TU_F_SIGNON_SCR_request)
TU_F_SIGNON_SCR_response (TU_F_SIGNON_SCR_response)
TU_F_SIGNON_SCR_screen (TU_F_SIGNON_SCR_screen)

Figure 101. Add messages to the TU_F_SAMPLE_PARSER folder

When you import a screen, the following items are created:

- screen
- request
- response

The screens identify where you are in the application. The request and response are the commands that are used as input and output for traversing the application and capturing the screens.

4) Under the CICS_SAMPLES message set, right click on the SAMPLE_TRANSACTION transaction and click **Copy**.

5) Right click on the Transactions folder in your TU_F_3270_MSG_SET message set and click **Paste**. You should rename the transaction to TU_F_SMPLE_TRANSACTION. Also rename the Identifier field to TU_F_SMPLE_TRANSACTION in

the Properties pane. Click the **Apply** bar at the bottom of the Properties pane to apply the change.

6) Add the messages representing Customer screen to the TU_F_SMPLE_TRANSACTION. This is needed to add data to a blank CICS screen. This is the screen that will display after the CMAV transaction is processed.

   Right click on the TU_F_SMPLE_TRANSACTION folder and select **Add > Message**. Select the TU_F_CUST_SCR_screen and click **Finish**.

c. The Message Sets view is shown in Figure 102. Save your workspace by selecting **File > Save Workspace**.

   Tip: The SAMPLE_PARSER is renamed TU_F_SAMPLE_PARSER, but the SAMPLE_TRANSACTION is renamed TU_F_SMPLE_TRANSACTION because when the COBOL source code for each of these transactions is generated, the first 8 characters (excluding the underscores "_") are used to create the name. This same 8–character name would be created for the TU_F_SAMPLE_TRANSACTION. By renaming the SAMPLE_TRANSACTION to TU_F_**SMPLE**_TRANSACTION, you avoided the name collision.

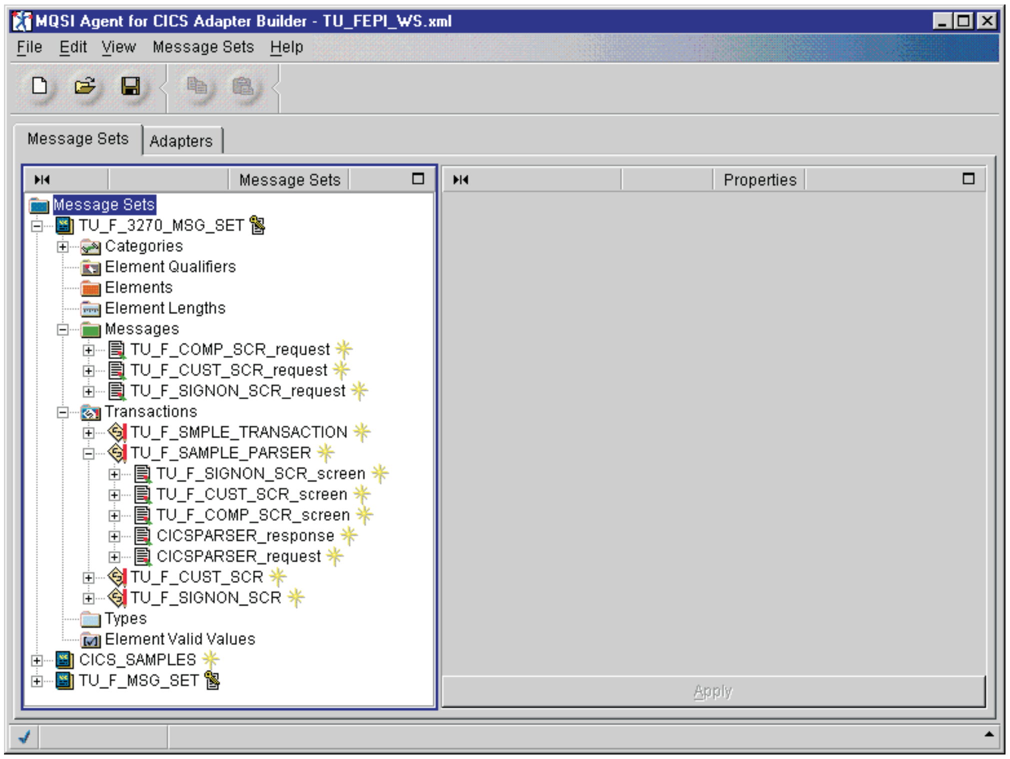


Figure 102. Message Sets view

Create the subflows for the FEPI adapter

In the following sections you will create the subflows for the FEPI adapter.

The following sections contain instructions that will allow you to create the FEPI adapter subflows identified inFigure 81 on page 115.

Each subflow that you create contains the instructions regarding its functional role at run time, as described in “About the adapter you will design” on page 112.

For each subflow, you will perform the following tasks:

1. **Create component types**

   A component type represents a template that can be used as a building block in modeling the subflow. The component types used by the subflow will depend on what the subflow is intended to do. For example, if at run time the subflow is required to test a condition for true or false to resolve a control flow path, you would create a *Decision* component type. For a list of the component types that are supplied with the adapter builder and for a description of their purpose, see the section on *Microflow components* in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

   When you are finished with creating component types for a subflow, you will have all the necessary *building blocks* required to model the subflow’s functionality. The component types will display in the Adapter Tree View. From the Adapter Tree view you will be able to drag a component type onto the Microflow Definition pane and begin the process of constructing the subflow.
2. **Model the subflow**

   In this step you will perform a set of tasks to model the subflow. When you model a subflow you are specifying how it will function at run time. Within the context of the business flow, the model is of the navigation of the server application with the back end systems.

   Within the builder, the model of each subflow is represented as a separate and distinct *microflow*, a sequence of nodes and connections. The microflow models the processing of a message as it passes from the input of the adapter to the output of the adapter.

   The modelling step is made up of the following tasks:

   - Adding microflow nodes
   - Connecting the microflow nodes
   - Defining the mappings
3. **Assign the model of the subflow to a CICS MQAdapter**

   By associating the model of the subflow with a CICS MQAdapter, you create the adapter that provides the actual implementation of the adapter request processing at run time.
4. **Generate the adapter**

   In this step you generate the copybooks and the run time code for the subflow.

Create the Parser subflow

Many production implementations of FEPI allow the FEPI application to leave the connection active or ***hot*** when the application has completed. The next time the connection is used, any screen could be received from the back-end system. Because of this, we recommend that you use a parser subflow to perform screen recognition.

The Parser subflow is used to identify which screen the user is on and then set a switch to determine the processing path.

Follow these steps to create the Parser subflow:

1. **Create the component types for use in the Parser subflow**

   This step is made of the following tasks:

   - Create a Command Type
   - Create a Microflow Type

   a. **Create a command type** .

      The TU_F_PARSER Command type will allow the microflow to identify the current screen (Complete screen, Signon screen or Customer screen). Make sure you are in the Adapters view.

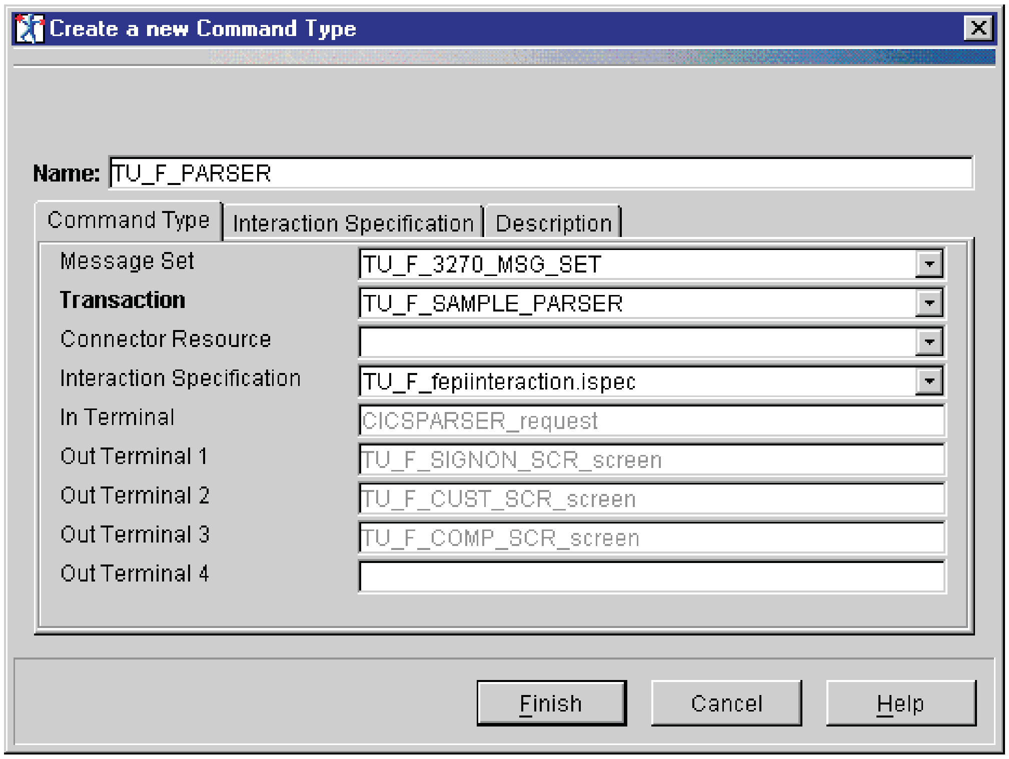
      1) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_F_PARSER* in the Name field.

      2) Using the drop down menus, set the following field property values:

Table 46. TU_F_PARSER Command property values

Field	Value
Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SAMPLE_PARSER
Interaction Specification	TU_F_fepiinteraction.ispec

Note: The Connector Resource for FEPI command types is not used and is left blank.



Name: TU_F_PARSER

Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SAMPLE_PARSER
Connector Resource	
Interaction Specification	TU_F_fepiinteraction.ispec
In Terminal	CICSPARSER_request
Out Terminal 1	TU_F_SIGNON_SCR_screen
Out Terminal 2	TU_F_CUST_SCR_screen
Out Terminal 3	TU_F_COMP_SCR_screen
Out Terminal 4	

Figure 103. Creating a TU_F_PARSER Command type

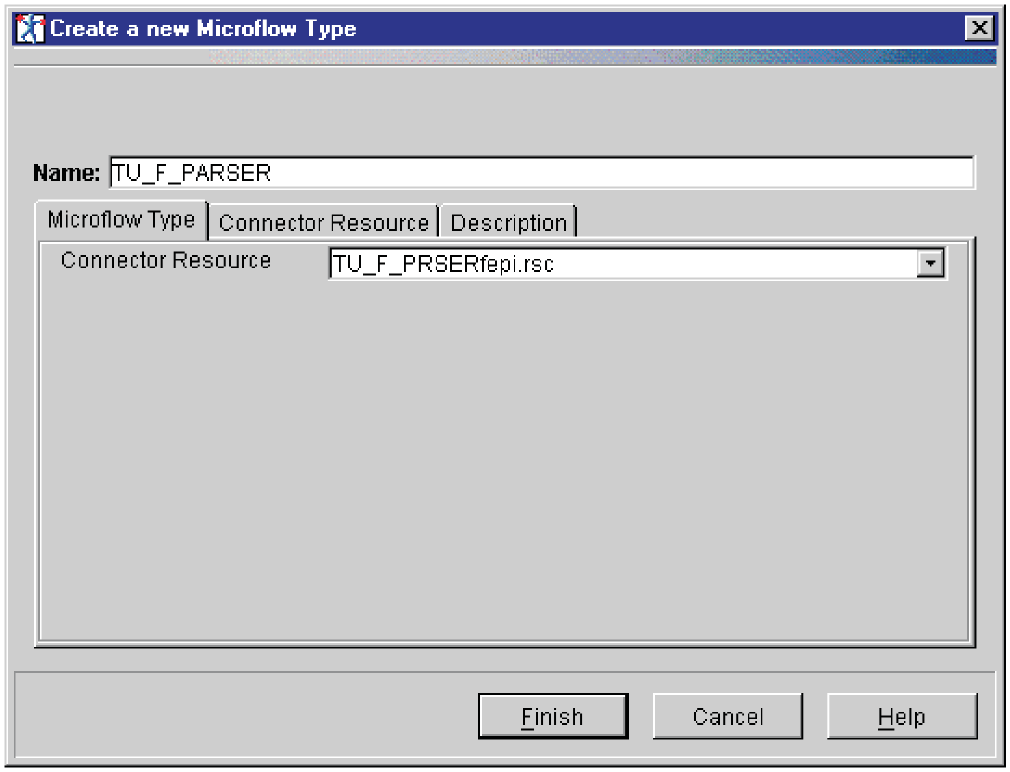
      Click **Finish** to apply the property values.

   b. **Create a microflow type**.

      Create a microflow that will model the Parser subflow processing.

Build an adapter that supports a FEPI interface

1) Right click on the Microflow Types folder and select **Create > Microflow Type**.
2) Enter TU_F_PARSER in the Name field.
3) Use the drop down menu in the Connector Resource field to select TU_F_PRSERfepi.rsc as the Connector Resource file and then click **Finish**.



Name: TU_F_PARSER

Connector Resource: TU_F_PRSERfepi.rsc

Figure 104. Creating a TU_F_PARSER Microflow Type

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created all of the component types that you will need to model the Parser subflow.

2. **Model the Parser subflow**

   In this step you will perform a set of tasks to define and model the Parser subflow's functionality. The model represents the behavior of this subflow at run time.

   This step is made of the following tasks:

   - Adding subflow nodes

- Connecting the subflow nodes
- Defining the mappings

a. **Add nodes to the Parser subflow**.

The subflow processing identifies which screen the user is on and sets a switch. Construct the Parser subflow by adding the nodes as shown in Figure 107 on page 144.

In this task, you will drag the component types that you created in step 1 on page 139, onto the Microflow Definition pane. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

1) **Add the Input Terminal node**

An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

a) Drag the node on to the Microflow Definition pane.

In the Microflow Types folder, select the TU_F_PARSER microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

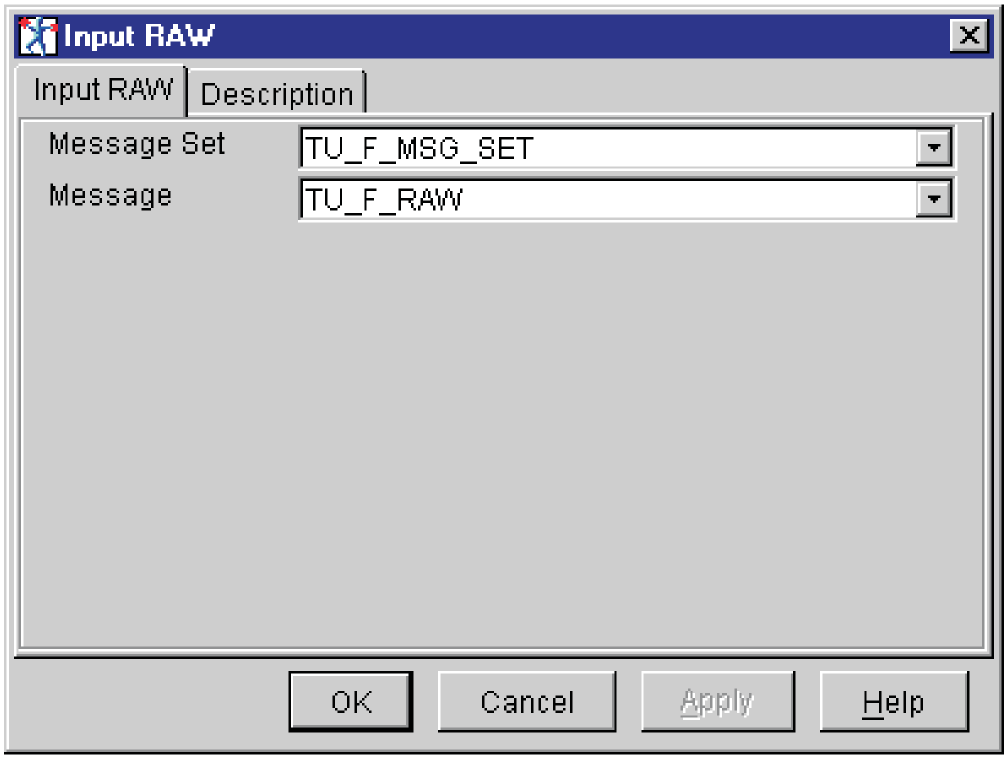
Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

b) Rename the node

Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input RAW and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_RAW in the Messages field. Click **OK**.



Message Set: TU_F_MSG_SET
Message: TU_F_RAW

Figure 105. Configuring the Input RAW Input Terminal node properties

2) **Add the Command node**

   In this step you will create a Parser command type that will be used to determine which screen you are on (Complete screen, Signon screen, or Customer screen).

   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_PARSER Command type from the Adapter Tree View to the workspace. Place the node to the right of the Input RAW node.

   b) Rename the node

      Right click on the TU_F_PARSER1 Command node and select **Rename**. Modify TU_F_PARSER1 in the New name field to the name TU_F_PARSER and click **Finish**.

3) **Add the Output Terminal node**

   An Output Terminal serves as an exit point for the microflow. The Output Terminal can receive connections only. It can never start a connection. A microflow can have multiple Output Terminals (as in the DPL and MQ tutorial exercises). A developer must design the controlling application to recognize the possible reply messages provided by multiple Output Terminals.

   a) Drag the node on to the Microflow Definition pane

      Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_PARSER node.

   b) Rename the node

      Rename the Output Terminal node to Output DEC and click **Finish**.

   c) Flip the node

Right click on the Output DEC node and select **Flip node**.

d) Set the properties for the node

Right click on the Output Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_DEC in the Messages field. Click **OK**.

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the Input RAW node and select **Connect > Out**. Move the connection line to the TU_F_PARSER node and left click. This adds a control connection and a map (Map1 node) between the two nodes.

A control connection provides a sequential relationship between 2 nodes in a microflow.

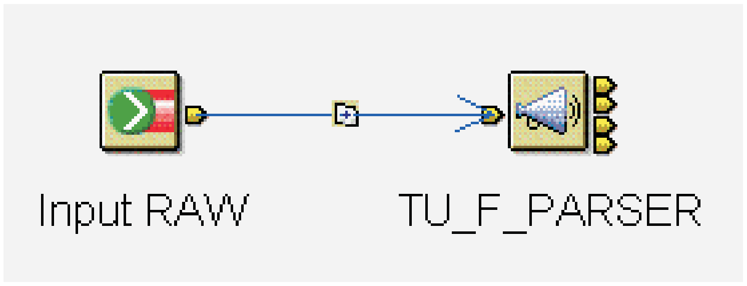


Figure 106. Connecting the Input RAW Input Terminal node and the TU_F_PARSER Command node

2) Add a control connection from the first out terminal (TU_F_SIGNON_SCR_screen) on the TU_F_PARSER node to the Output DEC node. This auto-adds a Map2 node on the control connection line. Refer to Figure 107 on page 144 (the Map node labeling has been added to the figure).

3) Add a control connection from the second out terminal (TU_F_CUST_SCR_screen) on the TU_F_PARSER node to the Output DEC node. This auto-adds a Map3 node on the control connection line.

4) Add a control connection from the third out terminal (TU_F_COMP_SCR_screen) on the TU_F_PARSER node to the Output DEC node. This auto-adds a Map4 node on the control connection line.

5) Add a Map node (Map5) between the TU_F_PARSER node and the Output DEC node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel).

6) Add a control connection from the fourth out terminal (labeled Unknown) on the TU_F_PARSER node to the Map5 node and from the Map5 node to the Output DEC node.

7) Save your workspace by selecting **File > Save Workspace** from the menubar.

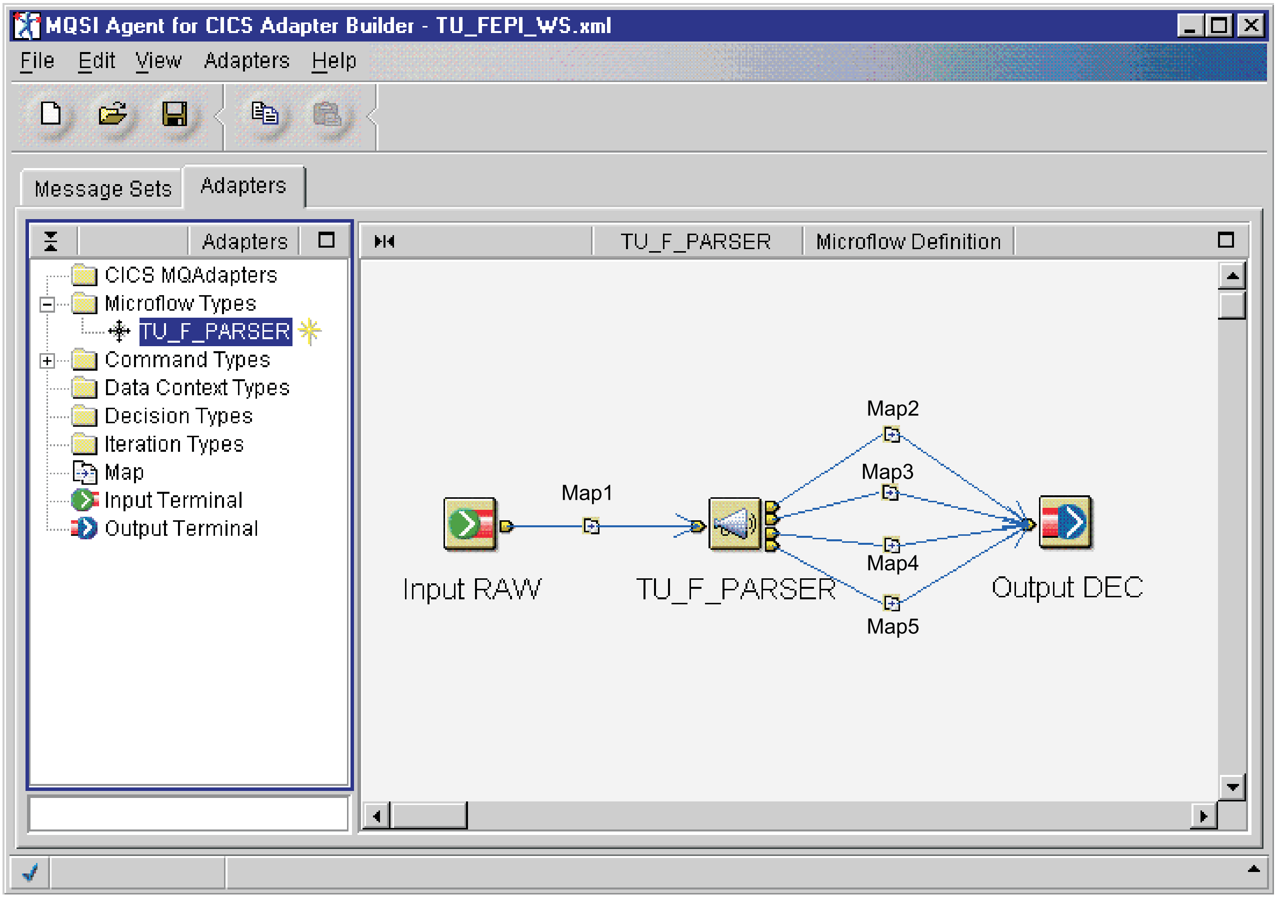


Figure 107. TU_F_PARSER

c. **Map your subflow**

You are now ready to perform the data mappings for the TU_F_PARSER subflow. The act of *mapping* refers to the modeling of data transformation via a Map node, between an output terminal on one node and an input terminal on another node.

Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 47 on page 145 and shown in Figure 108 on page 145. This map sets the CICSPARSER request to a blank character. The blank character is needed because there must be some character mapped in a mapper node.

Right click on the Map1 node (the Map node that appears between the Input RAW and TU_F_PARSER nodes) and select **Properties**. Click the DataMappingExpression tab.

Perform a literal mapping. Right click on the CICSPARSER_request field in the CICSPARSER_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ' ' (quotes must be used around the space).

Table 47. Mapping fields for Map1 node (TU_F_RAW message to CICSPARSER_request message)

Input Field	Output Field	Description
' '	CICSPARSER_request	Blank character to provide mapping content

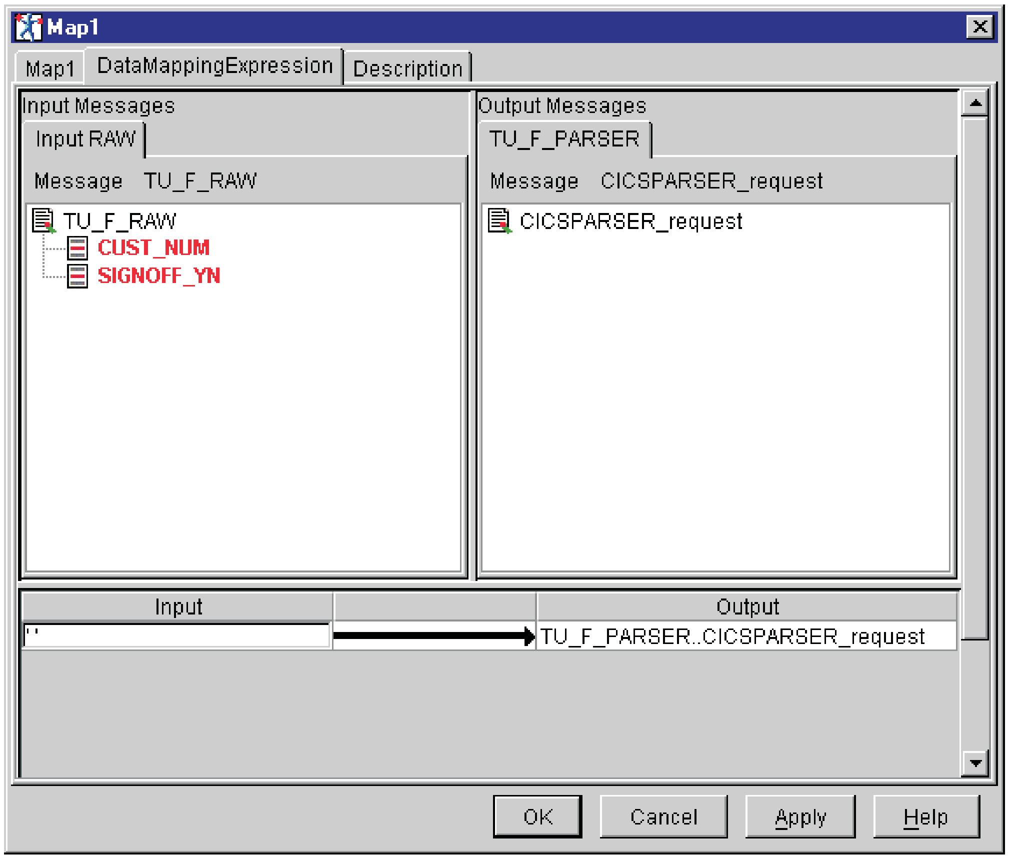
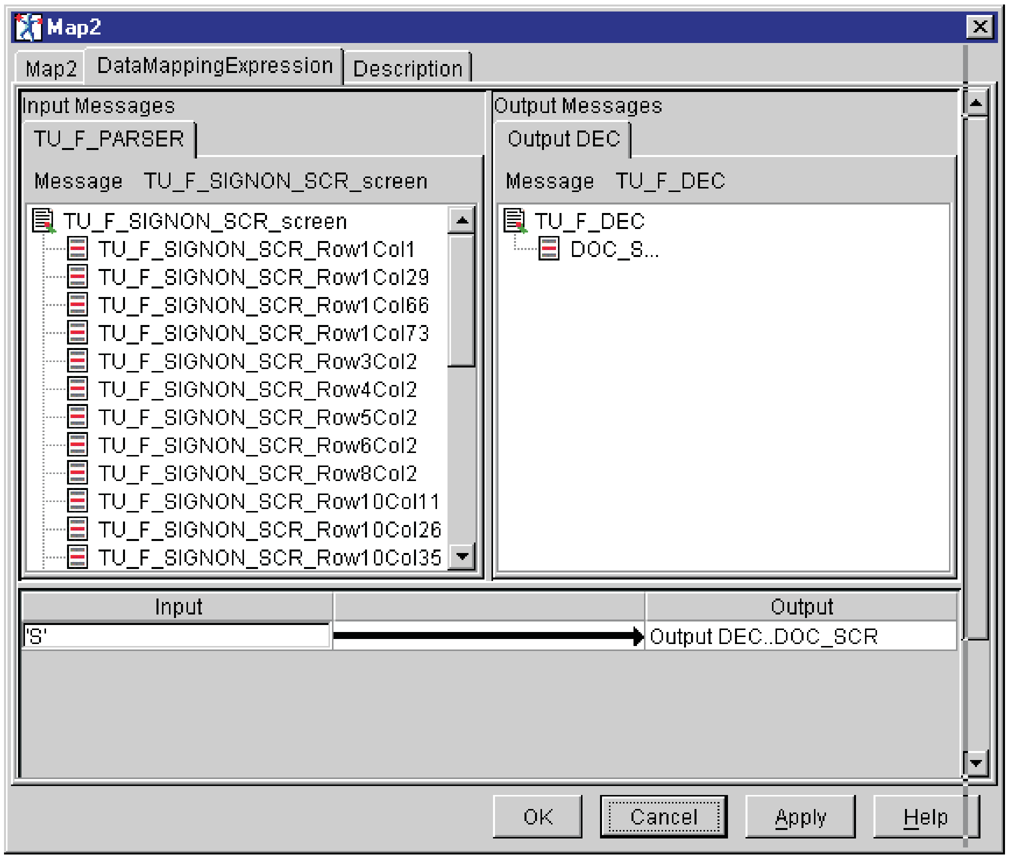


Figure 108. Mapping for Map1 node

2) Perform the mapping for the Map2 node as listed in Table 48 and shown in Figure 109 on page 146. This map passes information about the screen identity when a Signon screen is identified.

Table 48. Mapping fields for Map2 node (TU_F_SIGNON_SCR_screen message to TU_F_DEC message)

Input Field	Output Field	Description
'S'	DOC_SCR	Sets DOC_SCR to 'S' when on the Signon screen



Message TU_F_SIGNON_SCR_screen

- TU_F_SIGNON_SCR_screen
  - TU_F_SIGNON_SCR_Row1Col1
  - TU_F_SIGNON_SCR_Row1Col29
  - TU_F_SIGNON_SCR_Row1Col66
  - TU_F_SIGNON_SCR_Row1Col73
  - TU_F_SIGNON_SCR_Row3Col2
  - TU_F_SIGNON_SCR_Row4Col2
  - TU_F_SIGNON_SCR_Row5Col2
  - TU_F_SIGNON_SCR_Row6Col2
  - TU_F_SIGNON_SCR_Row8Col2
  - TU_F_SIGNON_SCR_Row10Col11
  - TU_F_SIGNON_SCR_Row10Col26
  - TU_F_SIGNON_SCR_Row10Col35

Message TU_F_DEC

- TU_F_DEC
  - DOC_S...

Input		Output
'S'	→	Output DEC..DOC_SCR

Figure 109. Mapping for Map2 node

3) Perform the mapping for the Map3 node as listed in Table 49. This map passes information about the screen identity when a Customer screen is identified.

Table 49. Mapping fields for Map3 node (TU_F_CUST_SCR_screen message to TU_F_DEC message)

Input Field	Output Field	Description
'C'	DOC_SCR	Sets DOC_SCR to 'C' when on the Customer screen

4) Perform the mapping for the Map4 node as listed in Table 50. This map passes information about the screen identity when an Unknown screen is identified.

Table 50. Mapping fields for Map4 node (TU_F_COMP_SCR_screen message to TU_F_DEC message)

Input Field	Output Field	Description
'U'	DOC_SCR	Sets DOC_SCR to 'U' when on the Complete screen

Note: For the purposes of this tutorial, the TU_F_COMP_SCR_screen is mapped as 'U' (Unknown).

5) Perform the mapping for the Map5 node as listed in Table 51 on page 147. This map passes information about the screen identity when a screen is unknown.

Table 51. Mapping fields for Map5 node (UNKNOWN message to TU_F_DEC message)

Input Field	Output Field	Description
'U'	DOC_SCR	Sets DOC_SCR to 'U' when on the Unknown screen

You just modelled the Parser subflow.

In your model, you have coded the instructions on how the subflow is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

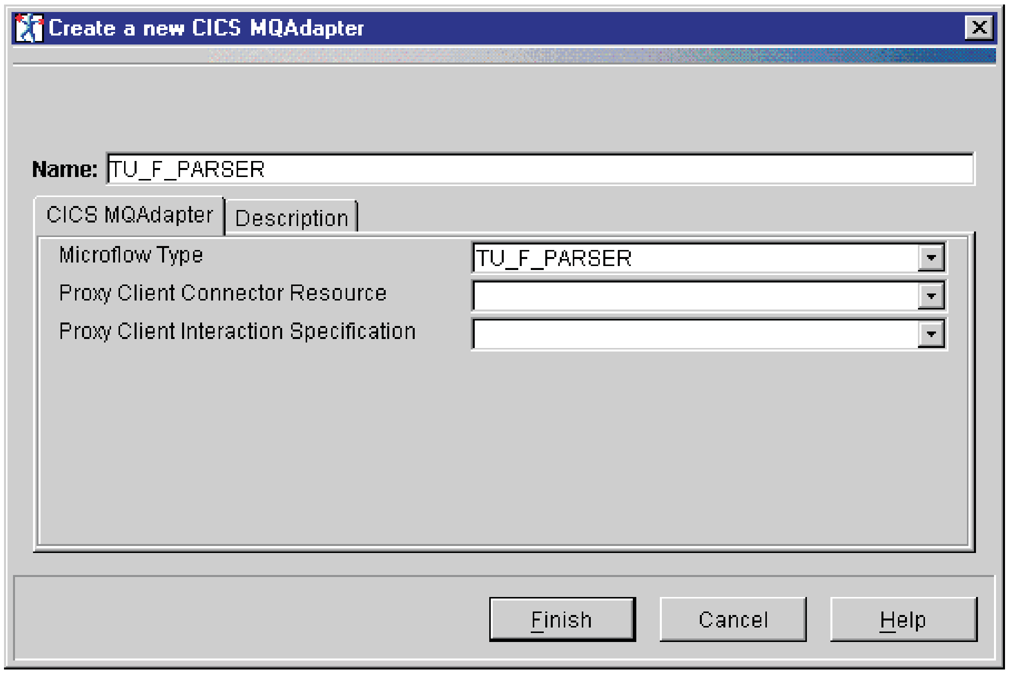
3. **Assign the model of the subflow to a CICS MQAdapter**

   In this step you will associate the subflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual implementation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_PARSER for the Name and use the drop down menu to select TU_F_PARSER for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TU_F_PARSER

Microflow Type: TU_F_PARSER
Proxy Client Connector Resource:
Proxy Client Interaction Specification:

Figure 110. Creating an CICS MQAdapter

You have completed the parser subflow and setup of this segment of your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

Now you are ready to generate your adapter.

4. **Generate the adapter**

   The adapter code files will be generated in the output directory that you specify.

   Adapter code generation is a two step process:

   a. Generate copybooks from message definitions (in Message Sets view).

   b. Generate the adapter run time code from the modeled microflow (in Adapters view).

   a. **Generate Copybooks**

      You will generate copybooks for the following messages:

      - TU_F_DEC
      - TU_F_RAW
      - TU_F_REPLY
      - TU_F_COMP_SCR_request
      - TU_F_CUST_SCR_request
      - TU_F_SIGNON_SCR_request

      Note: To generate a copybook for a message, the message must be checked out or newly created.

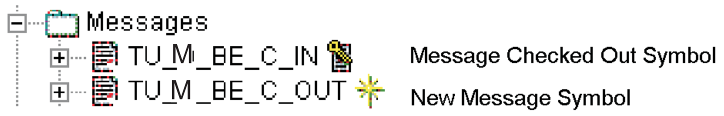
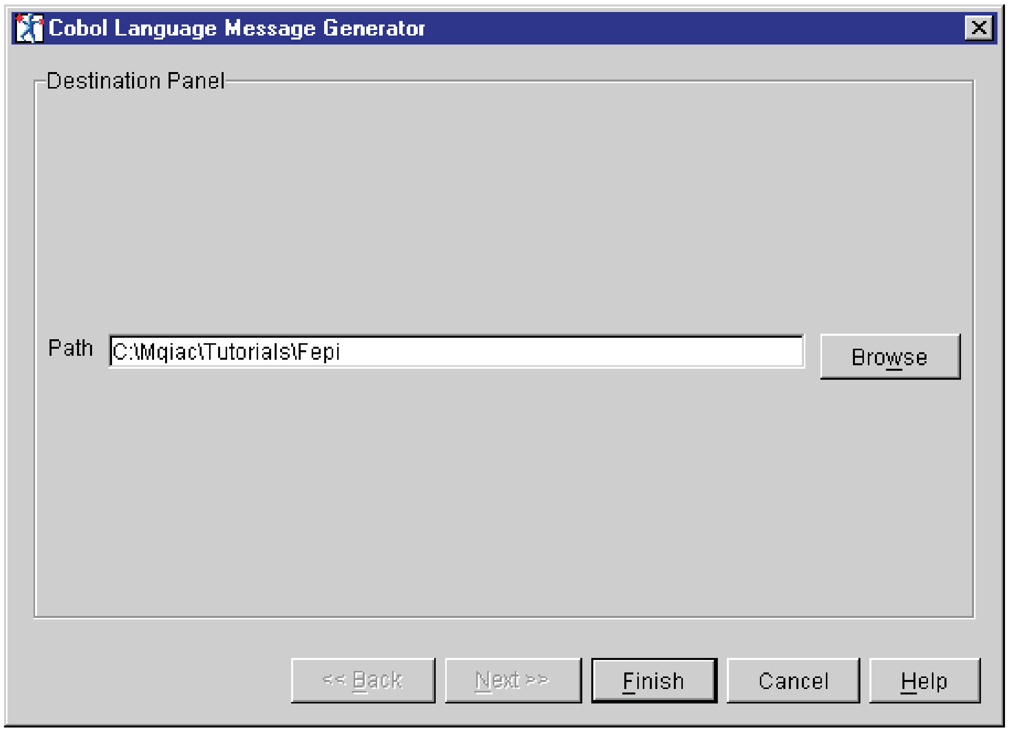


Figure 111. Messages Sets folder showing checked out message and newly created message

To generate copybooks, make sure that you are in the Message Sets view and then, follow this procedure:

1) Make sure the list of messages is visible under the Messages folder for the TU_F_3270_MSG_SET. To view the messages, click on the + sign in front of the Messages folder to display the list of messages.
2) Right click on the message for which you want to generate a copybook (for example, TU_F_COMP_SCR_request) and select **Generate > COBOL**.
3) Enter the output destination **<mqiac_tutorials>\fepi** in the Path field and click Finish.

   Note: The copybook generate removes underscores from the message names and only uses the first eight characters of the filename to generate the new copybook name.



Path C:\Mqiac\Tutorials\Fepi

Figure 112. Specifying pathname for copybook generation output

4) To complete generating copybooks, repeat steps 2 and 3 for each of the messages listed in step 2a on page 148.

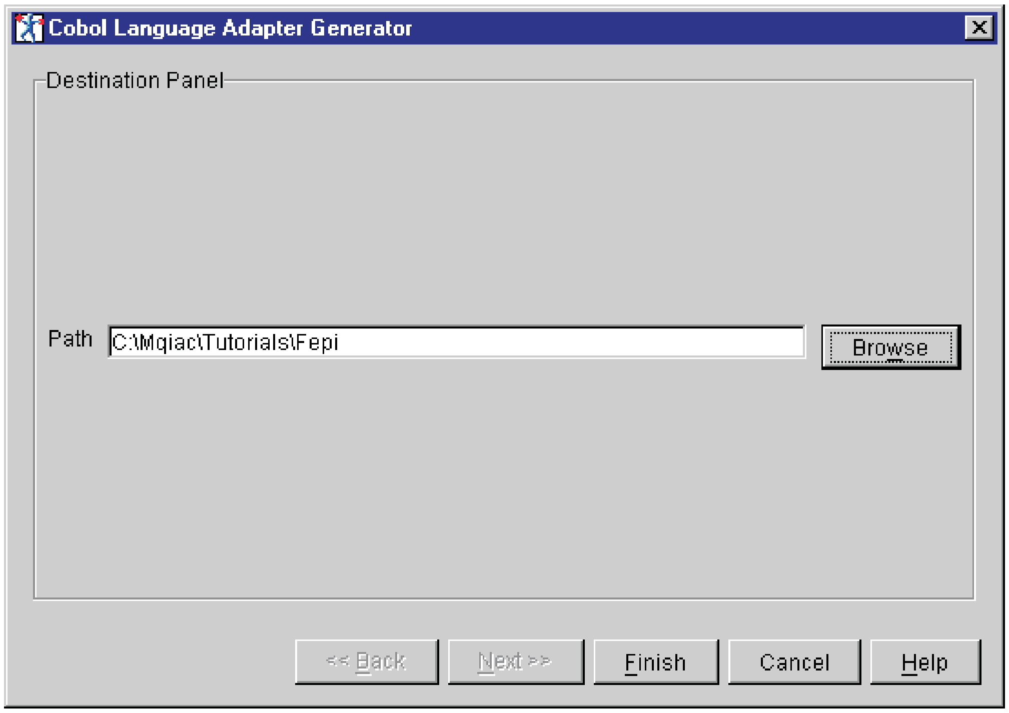
b. **Generate adapter code**

To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

Note: You must generate the adapter code in the same directory where you generated the copybooks.

1) Right click on the TU_F_PARSER adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination **<mqiac_tutorials>\fepi** in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

   The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\Tutorials\Fepi

Figure 113. Specifying pathname for adapter code generation output

Create the Signon subflow

The Signon subflow is used to sign a user onto a Customer application. Follow these steps to create the Signon subflow:

1. **Create the component types for use in the Signon subflow**

   This step is made of the following tasks:
   - Create 2 Command Types
   - Create a Data Context Type
   - Create a Microflow Type

   a. **Create Command Types.**

      You will need to create two Command types:
      - TU_F_SIGNON — Signs the user onto the system using FEPI PassTicket information.
      - TU_F_CMAV — Processes the CMAV transaction to bring up the Customer screen.

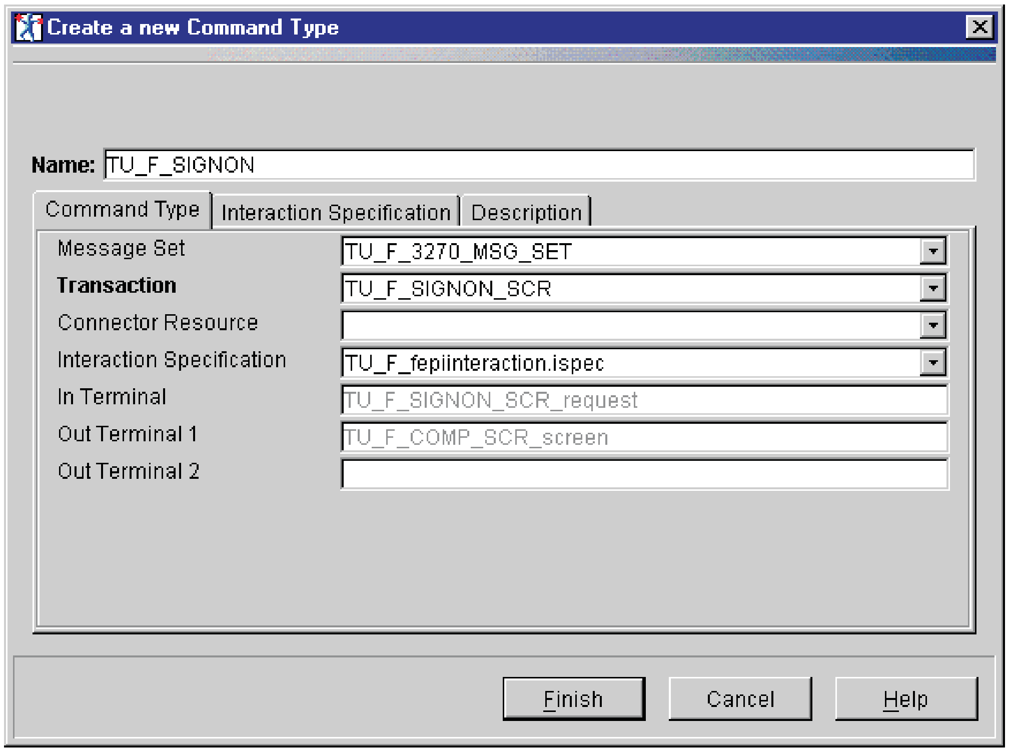
      1) *Create the TU_F_SIGNON Command type.*

         a) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_F_SIGNON* in the Name field.

         b) Using the drop down menus, set the following field property values:

Table 52. TU_F_SIGNON Command property values

Field	Value
Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SIGNON_SCR
Interaction Specification	TU_F_fepiinteraction.ispec



Name: TU_F_SIGNON

Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SIGNON_SCR
Connector Resource	
Interaction Specification	TU_F_fepiinteraction.ispec
In Terminal	TU_F_SIGNON_SCR_request
Out Terminal 1	TU_F_COMP_SCR_screen
Out Terminal 2	

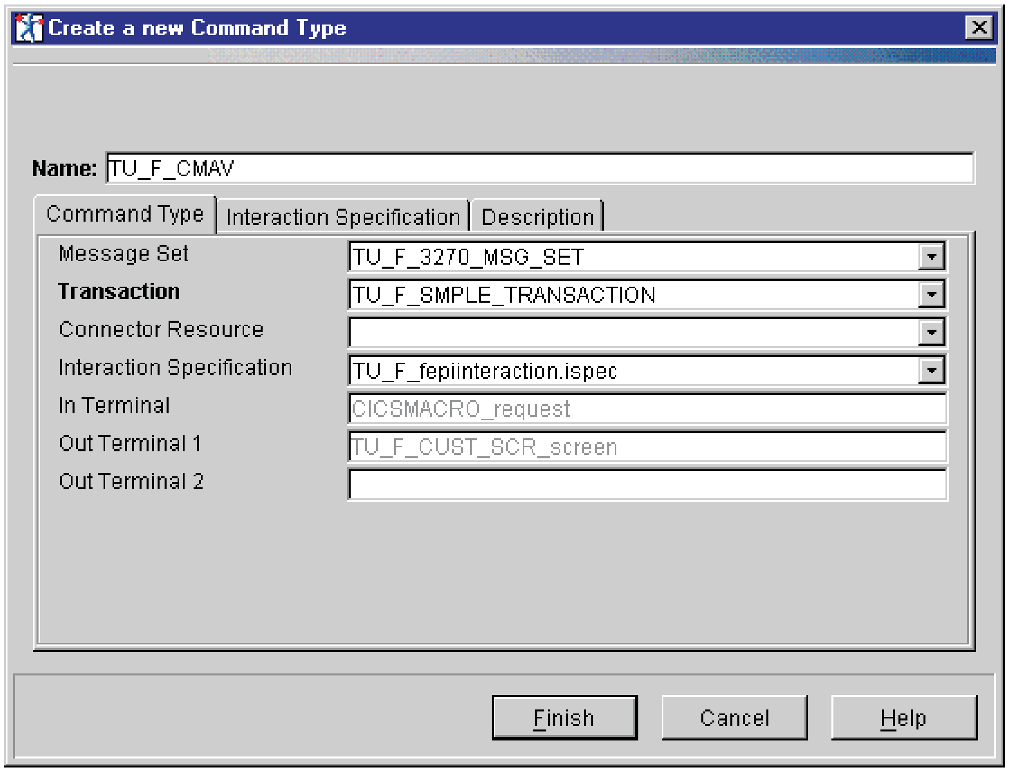
Figure 114. Creating a TU_F_SIGNON Command type

Click **Finish** to apply the property values.

2) *Create the TU_F_CMAV Command type.*
   a) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_F_CMAV* in the Name field.
   b) Using the drop down menus, set the following field property values:

Table 53. TU_F_CMAV Command property values

Field	Value
Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SMPLE_TRANSACTION
Interaction Specification	TU_F_fepiinteraction.ispec



Name: TU_F_CMAV

Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_SMPLE_TRANSACTION
Connector Resource	
Interaction Specification	TU_F_fepiinteraction.ispec
In Terminal	CICSMACRO_request
Out Terminal 1	TU_F_CUST_SCR_screen
Out Terminal 2	

Figure 115. Creating a TU_F_ CMAV Command type

Click **Finish** to apply the property values.

b. **Create the Data Context type**

A data context type is a simple adapter component that is used to store data for later access through a data flow.

Add a SYS_LU_LOGON Data Context. The SYS_LU_LOGON is the name of a system-supplied Message as well as a system-supplied Data Context type. It can be used in conjunction with the MAT_USELUPASS field defined in the Connector Resource associated with a FEPI microflow.

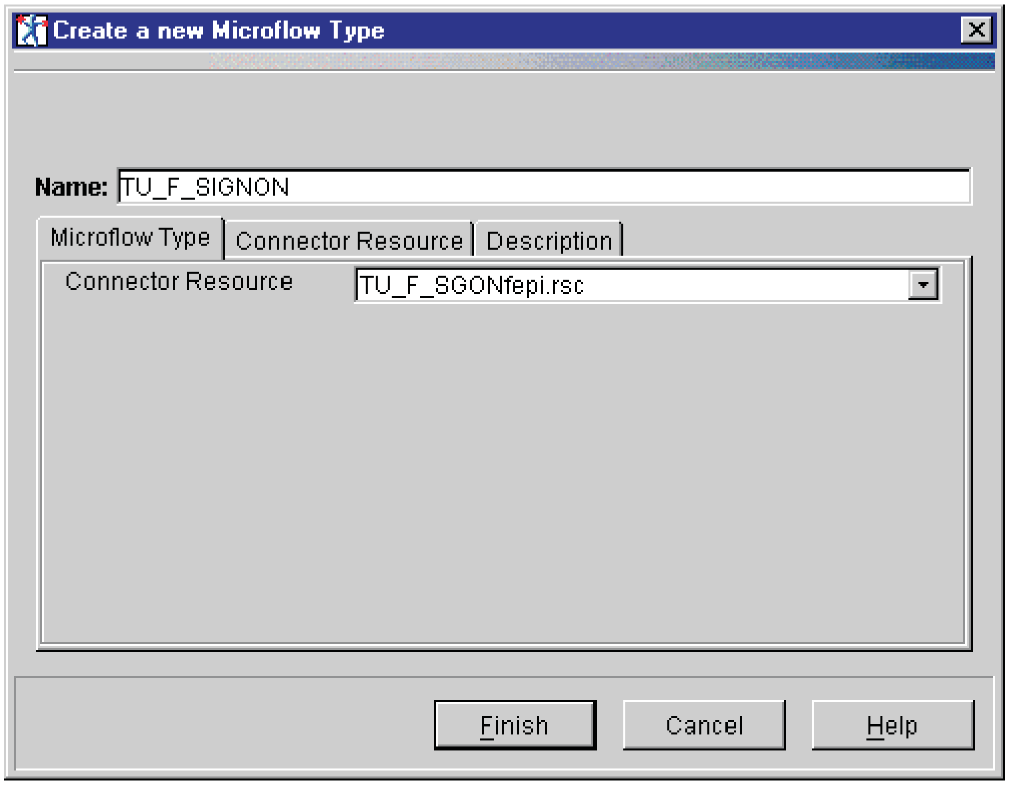
When the MAT_USELUPASS field is set to Y, the FEPI microflow will use the Userid from the MQ Message as the LU_OWNER and retrieve a PASSTICKET value from CICS. These values will be stored in the fields in the SYS_LU_LOGON Message. To use this FEPI PassTicket feature, you need to add a SYS_LU_LOGON Data Context to your workspace.

1) Right click on the Data Context Types folder and select **Add to Workspace > Data Context Type**.
2) In the Add an existing Data Context Type dialog list, select SYS_LU_LOGON and click **Finish**.

c. **Create a microflow type**

Create a microflow that will model the processing of the Signon.

1) Right click on the Microflow Types folder and select **Create > Microflow Type**.
2) Enter TU_F_SIGNON in the Name field.
3) Use the drop down menu in the Connector Resource field to select TU_F_SGONfepi.rsc as the Connector Resource file and then click **Finish**.



Name: TU_F_SIGNON

Connector Resource: TU_F_SGONfepi.rsc

Figure 116. Creating a TU_F_SIGNON Microflow Type

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created all of the component types that you will need to model the Signon subflow.

2. **Model the Signon subflow**

   In this step you will perform a set of tasks to define and model the Signon subflow's functionality. The model represents the behavior of this subflow at run time.

   This step is made of the following tasks:

   - Adding subflow nodes
   - Connecting the subflow nodes
   - Defining the mappings

   a. **Add nodes to the Signon subflow**

      In this task, you will drag the component types that you created in 1 on page 150. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

The Signon subflow processing uses the FEPI Override feature to supply a Userid and PassTicket that allows the user to signon to the host system. You will add the nodes shown in Figure 118 on page 157.

1) **Add the Input Terminal node**

   An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

   a) Drag the node on to the Microflow Definition pane

      In the Microflow Types folder, select the TU_F_SIGNON microflow you created.

      Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

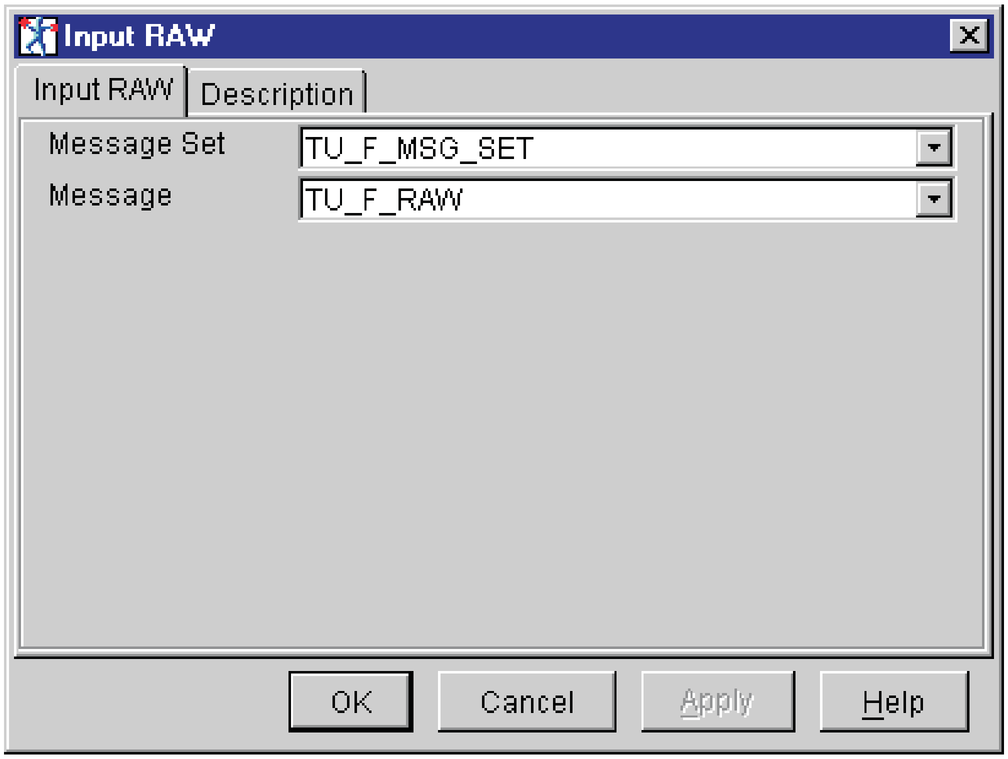
      Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

   b) Rename the node

      Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input RAW and click **Finish**.

   c) Set the properties for the node

      Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_RAW in the Messages field. Click **OK**.



Message Set: TU_F_MSG_SET

Message: TU_F_RAW

Figure 117. Configuring the Input RAW Input Terminal node properties

2) **Add the Signon Command node**

In this step you will add a TU_F_SIGNON Command node which will be used to sign the user onto the host system.

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_SIGNON Command type from the Adapter Tree View to the workspace. Place the node to the right of the Input RAW Input Terminal node

b) Rename the node

Right click on the TU_F_SIGNON1 Command node and select **Rename**.

Modify TU_F_SIGNON1 in the New name field to the name TU_F_SIGNON and click **Finish**.

3) **Add the Data Context node**

In this step you will add the SYS_LU_LOGON Data Context which will provide the data context for the USERID and PASSTICKET fields.

a) Drag the node on to the Microflow Definition pane

Drag a SYS_LU_LOGON Data Context type from the Adapter Tree View to the workspace. Place the node between the Input RAW node and TU_F_SIGNON node but above the nodes.

b) Rename the node

Right click on the SYS_LU_LOGON1 Command node and select **Rename**.

Modify SYS_LU_LOGON1 in the New name field to the name SYS_LU_LOGON and click **Finish**

4) **Add the CMAV Command node**

In this step you will add a TU_F_CMAV Command which will be used to process the CMAV transaction and display the Customer screen.

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_CMAV Command type from the Adapter Tree View to the workspace. Place the node to the right of the TU_F_SIGNON node and slightly above.

b) Rename the node

Right click on the TU_F_CMAV1 Command node and select **Rename**.

Modify TU_F_CMAV1 in the New name field to the name TU_F_CMAV and click **Finish**

5) **Add the Output terminal node**

a) Drag the node on to the Microflow Definition pane

Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_CMAV node.

b) Rename the node

Right mouse click and rename the Output Terminal node to Output REPLY.

c) Flip the node

Right click on the Output REPLY node and select **Flip node**.

d) Set the properties for the node

Right click on the Output Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**.

6) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the Input RAW node and select **Connect > Out**. Move the connection line to the TU_F_SIGNON node and left click. This adds a control connection and a map (Map1 node) between the two nodes. Refer to Figure 118 on page 157.

2) Add a data connection from the out terminal on the SYS_LU_LOGON node to the Map1 node.

3) Add a control connection from the first out terminal (TU_F_COMP_SCR_screen) on the TU_F_SIGNON node to the TU_F_CMAV node. This auto-adds a Map2 node on the control connection line.

4) Add a Map node (Map3) between the TU_F_SIGNON node and the Output REPLY node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel). Position the cursor between the nodes and release the mouse button.

5) Add a control connection from the second out terminal (Unknown) on the TU_F_SIGNON node to Map3 node and then add a control connection from the Map3 node to the Output REPLY node.
6) Add a control connection from the first out terminal (TU_F_CUST_SCR_screen) on the TU_F_CMAV node to the Output REPLY node. This auto-adds a Map4 node on the control connection line.
7) Add a control connection from the second out terminal (labeled Unknown) on the TU_F_CMAV node to the Map3 node.
8) Save your workspace by selecting **File > Save Workspace** from the menubar.

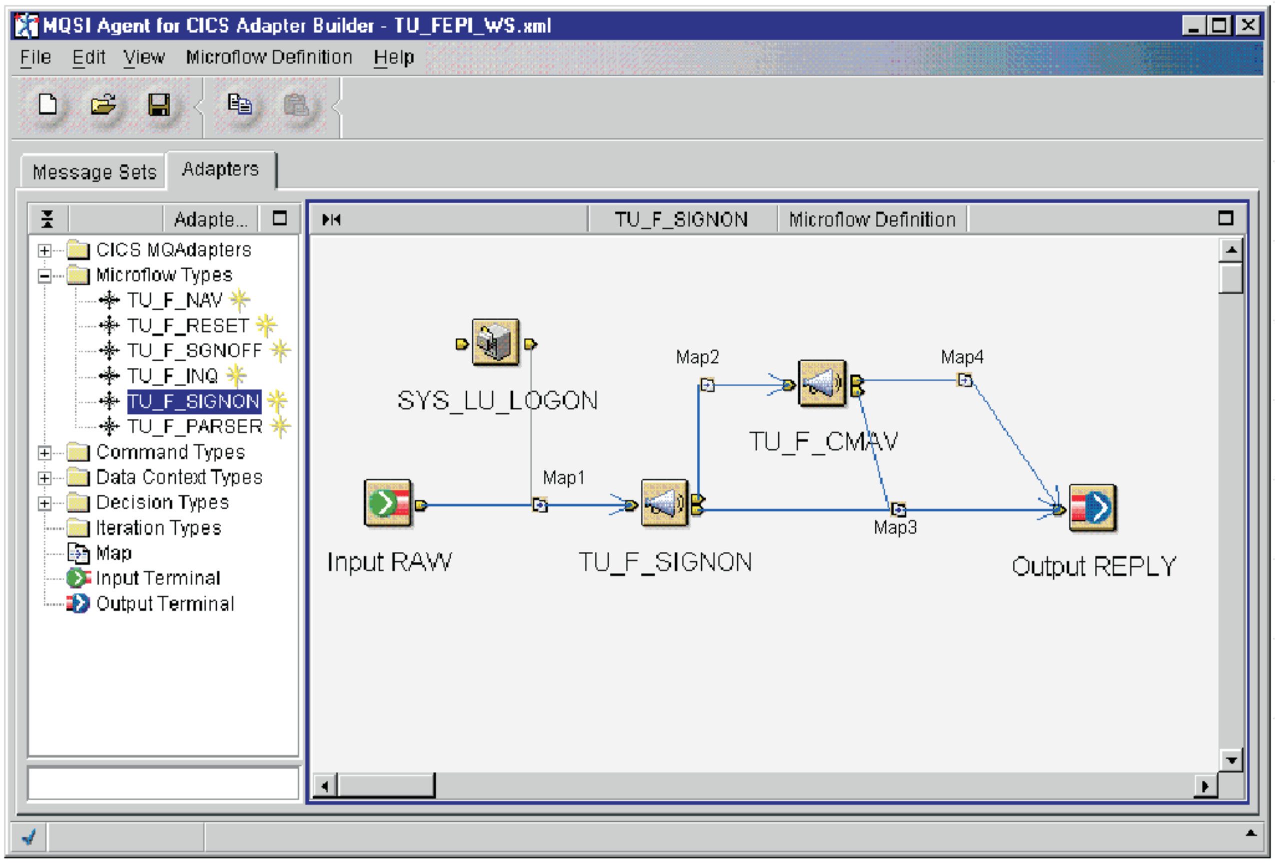


Figure 118. TU_F_SIGNON

c. **Map your subflow**

You are now ready to perform the data mappings for the TU_F_SIGNON subflow. Mapping models data transformation via a Map node between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

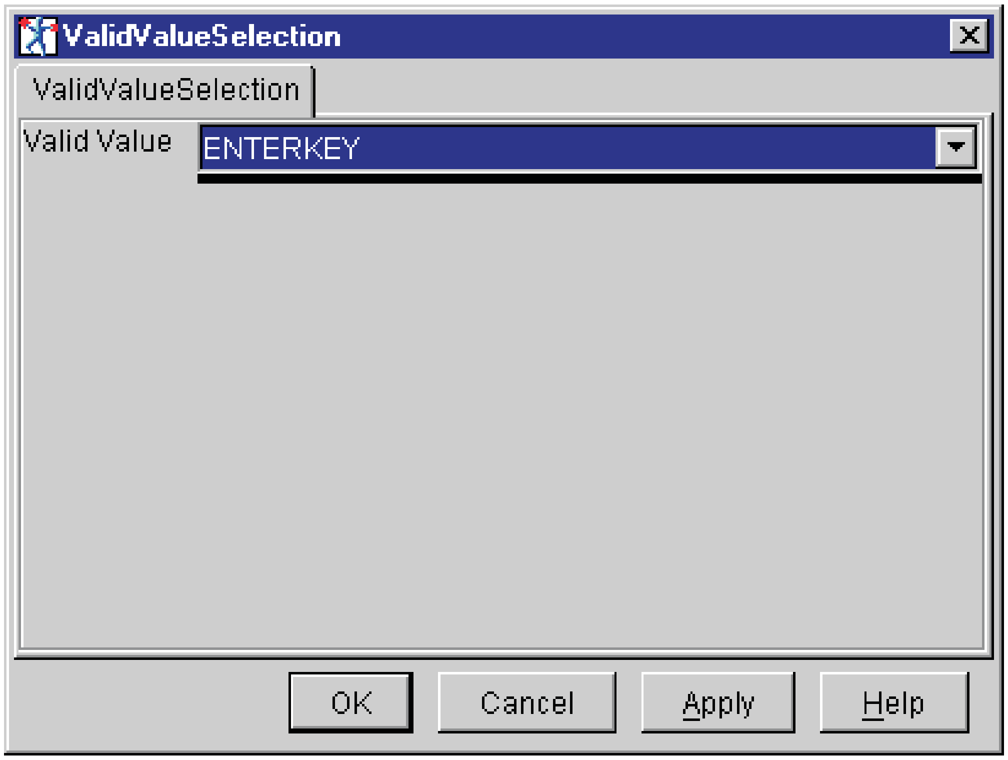
Note: Valid values will assign the appropriate PF key "value" to the Aid key.

1) Perform the mapping for the Map1 node as listed in Table 54 and shown in Figure 120 on page 159. Map1 maps the LU_OWNER, PASSTICKET, and ENTERKEY for the CICS Signon screen.

   Right click on the Map1 node (the Map node that appears between the Input RAW and TU_F_SIGNON nodes) and select **Properties**. Click the DataMappingExpression tab.

   a) Perform a valid value mapping. Right click on the AIDKEY field in the TU_F_SIGNON_SCR_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select ENTERKEY in the Valid Value field. Click **OK**.

      Note: Do not type the word ENTERKEY, but rather make sure you select it from the drop down menu for the Valid Value field.

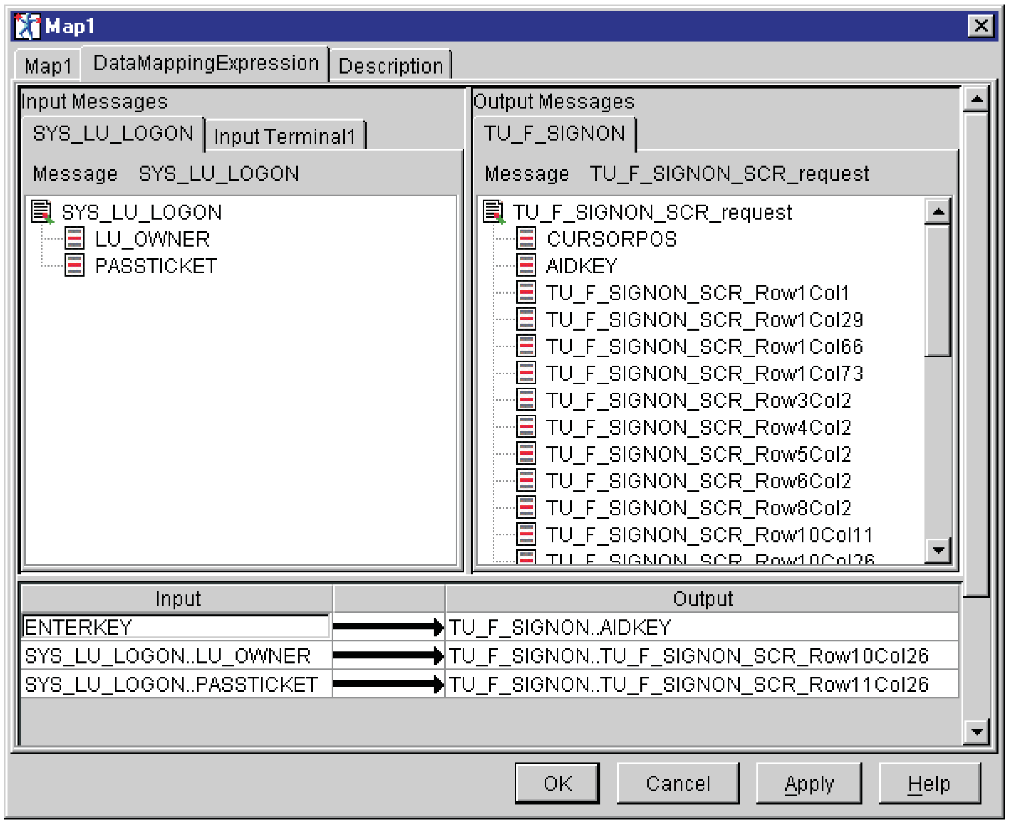


Valid Value: ENTERKEY

Figure 119. Valid Values dialog

Table 54. Mapping fields for Map1 node (SYS_LU_LOGON message to TU_F_SIGNON_SCR_request message)

Input Field	Output Field	Description
ENTERKEY	AIDKEY	Maps the enter key for signon
LU_OWNER	TC_F_SIGNON_SCR_Row10Col26	Maps screen position that corresponds to the LU_OWNER field.
PASSTICKET	TC_F_SIGNON_SCR_Row11Col26	Maps screen position that corresponds to the PASSTICKET field.



Message SYS_LU_LOGON

- SYS_LU_LOGON
  - LU_OWNER
  - PASSTICKET

Message TU_F_SIGNON_SCR_request

- TU_F_SIGNON_SCR_request
  - CURSORPOS
  - AIDKEY
  - TU_F_SIGNON_SCR_Row1Col1
  - TU_F_SIGNON_SCR_Row1Col29
  - TU_F_SIGNON_SCR_Row1Col66
  - TU_F_SIGNON_SCR_Row1Col73
  - TU_F_SIGNON_SCR_Row3Col2
  - TU_F_SIGNON_SCR_Row4Col2
  - TU_F_SIGNON_SCR_Row5Col2
  - TU_F_SIGNON_SCR_Row6Col2
  - TU_F_SIGNON_SCR_Row8Col2
  - TU_F_SIGNON_SCR_Row10Col11
  - TU_F_SIGNON_SCR_Row10Col26

Input		Output
ENTERKEY	→	TU_F_SIGNON..AIDKEY
SYS_LU_LOGON..LU_OWNER	→	TU_F_SIGNON..TU_F_SIGNON_SCR_Row10Col26
SYS_LU_LOGON..PASSTICKET	→	TU_F_SIGNON..TU_F_SIGNON_SCR_Row11Col26

Figure 120. Mapping for Map1 node

b) Perform the LU_OWNER mapping. Click in the LU_OWNER field under the SYS_LU_LOGON message and drag to the TU_F_SIGNON_SCR_Row10Col26 field under the TU_F_SIGNON_SCR_request message.

c) Perform the PASSTICKET mapping. Click in the PASSTICKET field under the SYS_LU_LOGON message and drag to the TU_F_SIGNON_SCR_Row11Col26 field under the TU_F_SIGNON_SCR_request message. Click **OK**.

2) Perform the mapping for the Map2 node as listed in Table 55 on page 160 and shown in Figure 121 on page 160. Map2 maps the CLEARKEY for initially clearing the screen, the 'CMAV' transaction ID for bringing up the Customer screen, and the ENTERKEY.

Right click on the Map2 node (the Map node that appears between the TU_F_SIGNON and TU_F_CMAV nodes) and select **Properties**. Click the DataMappingExpression tab.

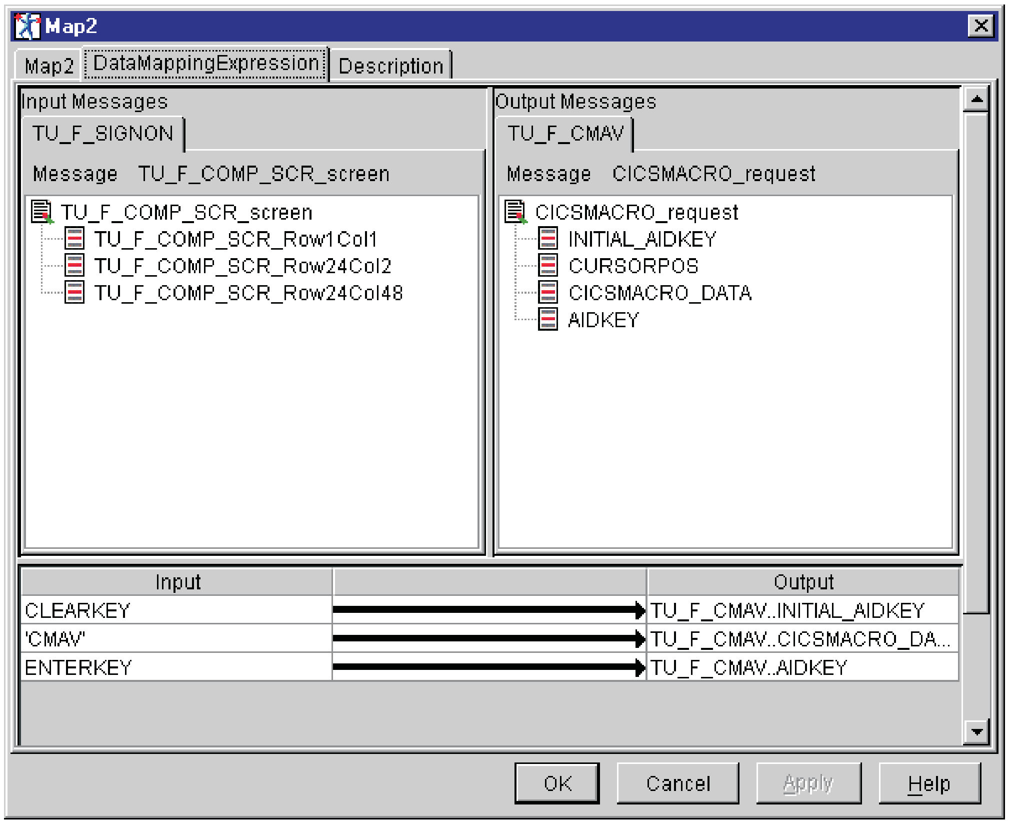
a) Perform a valid value mapping. Right click on the destination field for the literal (the INITIAL_AIDKEY field in the CICSMACRO_request Output Message) and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select CLEARKEY in the Valid Value field.

b) Perform a literal mapping. Right click on the destination field for the literal (the CICSMACRO_DATA field in the CICSMACRO_request Output Message) and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to 'CMAV' (quotes must be used around the CMAV string).

c) Perform a valid value mapping. Right click on the AIDKEY field in the CICSMACRO_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select ENTERKEY in the Valid Value field. Click **OK**.

Table 55. Mapping fields for Map2 node (TU_F_COMP_SCR_screen message to CICSMACRO_request message)

Input Field	Output Field	Description
CLEARKEY	INITIAL_AIDKEY	Maps the key to initial clear the screen
'CMAV'	CICSMACRO_DATA	Maps the 'CMAV' transaction that displays the Customer screen
ENTERKEY	AIDKEY	Maps the enter key



Input		Output
CLEARKEY	→	TU_F_CMAV..INITIAL_AIDKEY
'CMAV'	→	TU_F_CMAV..CICSMACRO_DA...
ENTERKEY	→	TU_F_CMAV..AIDKEY

Figure 121. Mapping for Map2 node

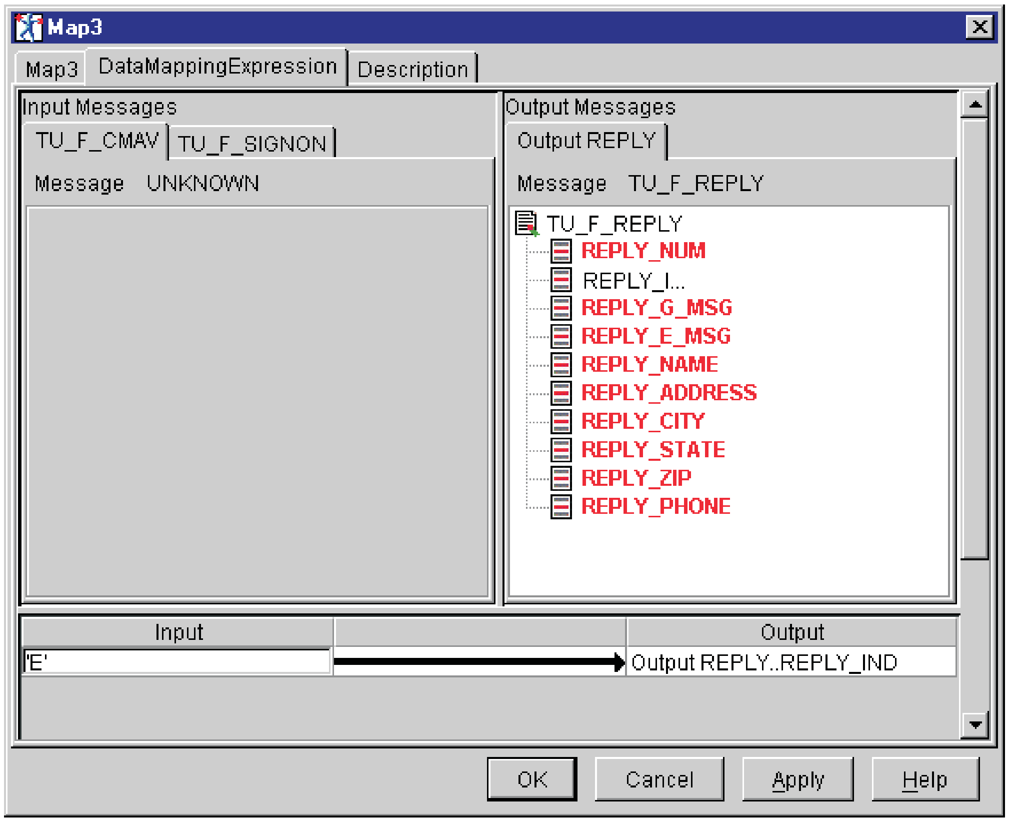
3) Perform the mapping for the Map3 node as listed in Table 56 on page 161 and shown in Figure 122 on page 161. This map sets the REPLY_IND to 'E' to indicate a bad signon.

Perform a literal mapping. Right click on the REPLY_IND field in the TU_F_REPLY Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

Double click on LITERAL field and rename it to 'E' (quotes must be used around the string). Click **OK**.

Table 56. Mapping fields for Map3 node (Unknown message to TU_F_REPLY message)

Input Field	Output Field	Description
'E'	REPLY_IND	Sets REPLY_IND to 'E' to indicate a bad signon



Message UNKNOWN

Message TU_F_REPLY

TU_F_REPLY
REPLY_NUM
REPLY_I...
REPLY_G_MSG
REPLY_E_MSG
REPLY_NAME
REPLY_ADDRESS
REPLY_CITY
REPLY_STATE
REPLY_ZIP
REPLY_PHONE

Input		Output
'E'	→	Output REPLY..REPLY_IND

Figure 122. Mapping for Map3 node

4) Perform the mapping for the Map4 node as listed in Table 57. This map sets the REPLY_IND to 'G' to indicate a good signon.

Perform a literal mapping. Right click on the REPLY_IND field in the Output REPLY Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

Double click on LITERAL field and rename it to 'G' (quotes must be used around the string). Click **OK**.

Table 57. Mapping fields for Map4 node (TU_F_CMAV message to Output REPLY message)

Input Field	Output Field	Description
'G'	REPLY_IND	Sets REPLY_IND to 'G' to indicate a good signon

Build an adapter that supports a FEPI interface

You just modelled the Signon subflow.

In your model, you have coded the instructions on how the subflow is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

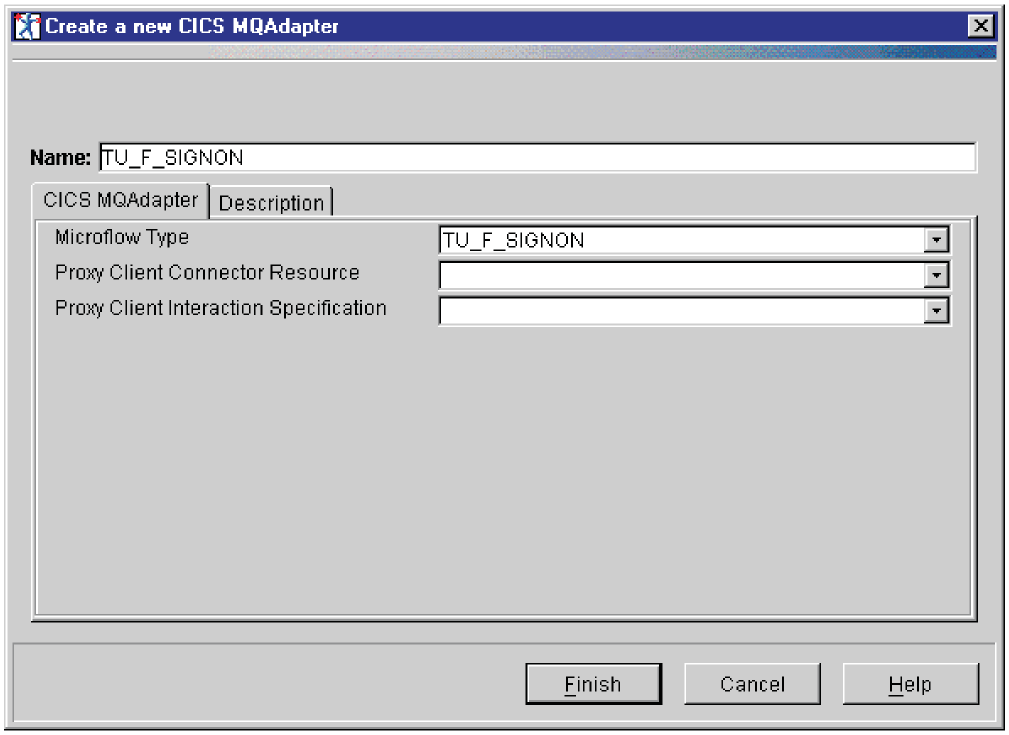
3. **Assign the model of the subflow to a CICS MQAdapter**.

   In this step you will associate the TU_F_SIGNON subflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual implementation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**.

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_SIGNON for the Name and use the drop down menu to select TU_F_SIGNON for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TU_F_SIGNON

Microflow Type	TU_F_SIGNON
Proxy Client Connector Resource	
Proxy Client Interaction Specification	

Figure 123. Creating an CICS MQAdapter

You have completed the TU_F_SIGNON subflow and setup your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

Now you are ready to generate your adapter.

Note: The copybooks were previously generated during the parser subflow (TU_F_PARSER) adapter modeling section (see step2a on page 148).

4. **Generate adapter Code**.

   To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

   Note: You must generate the adapter code in the same directory where you generated the copybooks.

   a. Right click on the TU_F_SIGNON adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination **<mqiac_tutorials>\fepi** in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

      The generated adapter code will be output to the destination path directory.

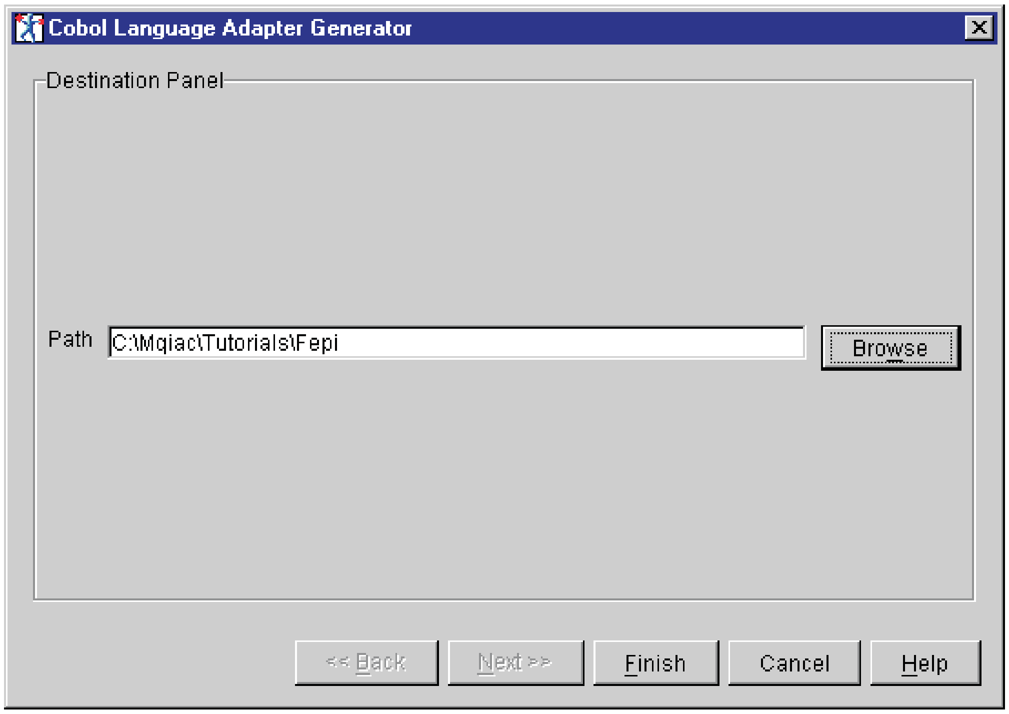


Figure 124. Specifying pathname for adapter code generation output

Create the Inquiry subflow

The Inquiry subflow is used to perform an inquiry on Customer information.

Follow these steps to create the Inquiry subflow:

1. **Create the component types for use in the Parser subflow**

   This step is made of the following tasks:

   - Create a Command Type
   - Create a Decision Type
   - Create a Microflow Type

   a. **Create the TU_F_CUST Command type.**

      You will need to create a Command type which processes the Customer information screen.

Build an adapter that supports a FEPI interface

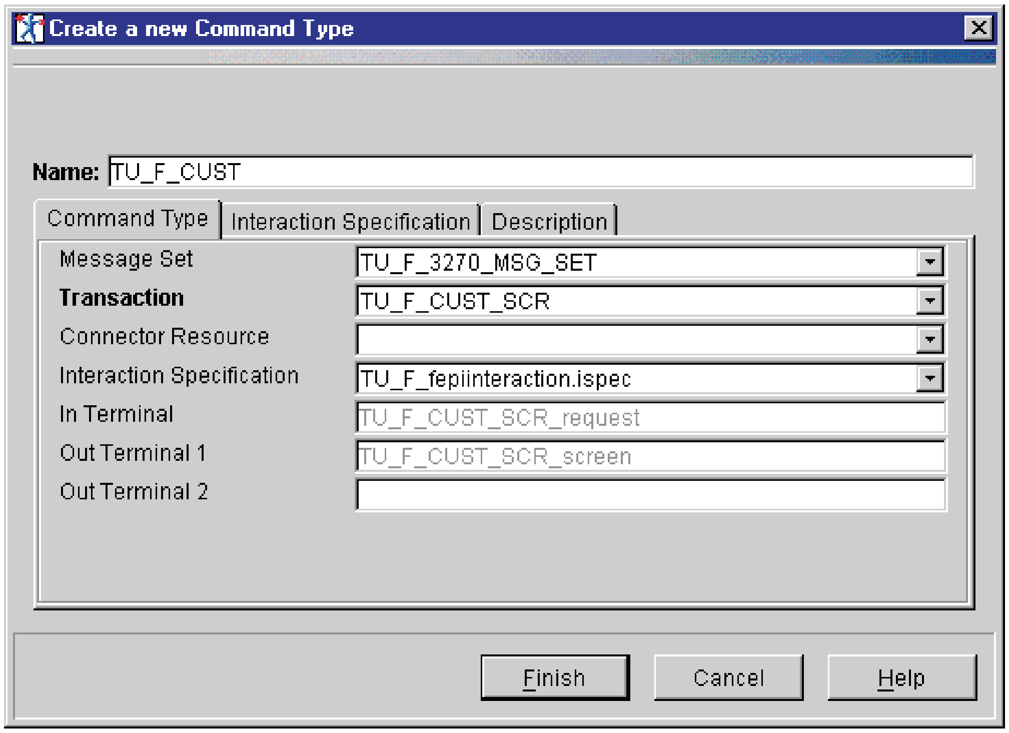
Click on the Adapters tab to switch to the Adapters view.

1) Right click on the Command Types folder and select **Create > Command Type**. Enter *TU_F_CUST* in the Name field.

2) Using the drop down menus, set the following field property values:

Table 58. TU_F_CUST Command property values

Field	Value
Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_CUST_SCR
Interaction Specification	TU_F_fepiinteraction.ispec



Name: TU_F_CUST

Message Set	TU_F_3270_MSG_SET
Transaction	TU_F_CUST_SCR
Connector Resource	
Interaction Specification	TU_F_fepiinteraction.ispec
In Terminal	TU_F_CUST_SCR_request
Out Terminal 1	TU_F_CUST_SCR_screen
Out Terminal 2	

Figure 125. Creating a TU_F_CUST Command type

Click **Finish** to apply the property values.

b. **Create a Decision Type**

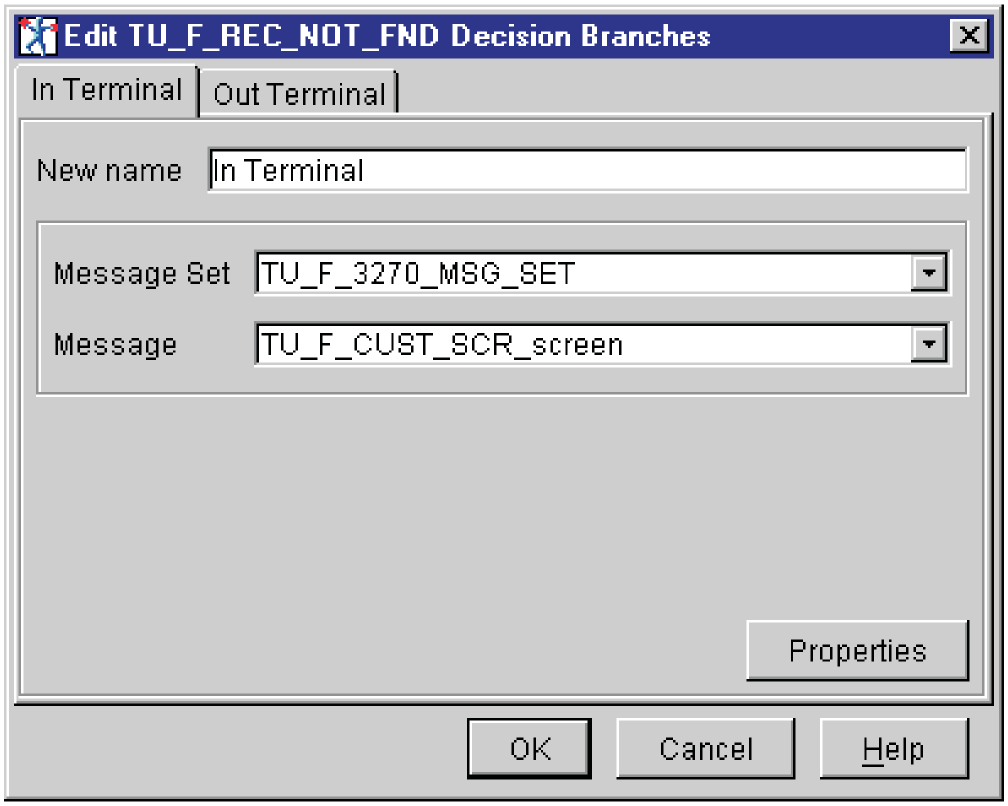
You will need to create a Decision type which will test to see if the selected record specified in the input message is in the file.

1) Right click on the Decision Types folder and select **Create > Decision Type**. Enter *TU_F_REC_NOT_FND* in the Name field and click **Finish**.

2) Associate a message set and message with the In Terminal on the TU_F_REC_NOT_FND Decision type.

a) Right click on the TU_F_REC_NOT_FND Decision type under the Decision Types folder and select **Decision Branch**. Make sure the In Terminal tab is selected.

b) Using the drop down menus, select TU_F_3270_MSG_SET for the Message Sets field and TU_F_CUST_SCR_screen for the Messages field.



New name: In Terminal

Message Set: TU_F_3270_MSG_SET

Message: TU_F_CUST_SCR_screen

Figure 126. Editing the In Terminal on the Decision type

3) Create an Out Terminal for the Record Not Found decision. On the Edit TU_F_REC_NOT_FND Decision Branches, select the Out Terminal tab. Click Out Terminal in the terminal list box. and click **Rename**. Enter *REC_NOT_FND* in the New name field and click **Finish**. Click **OK**.

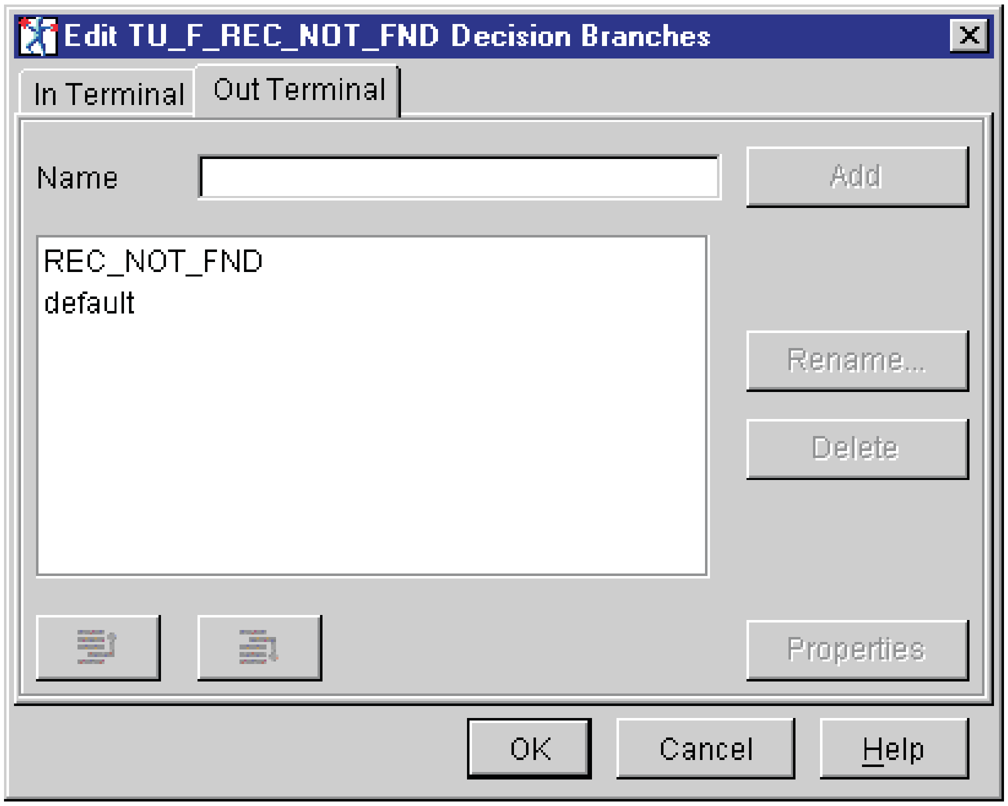
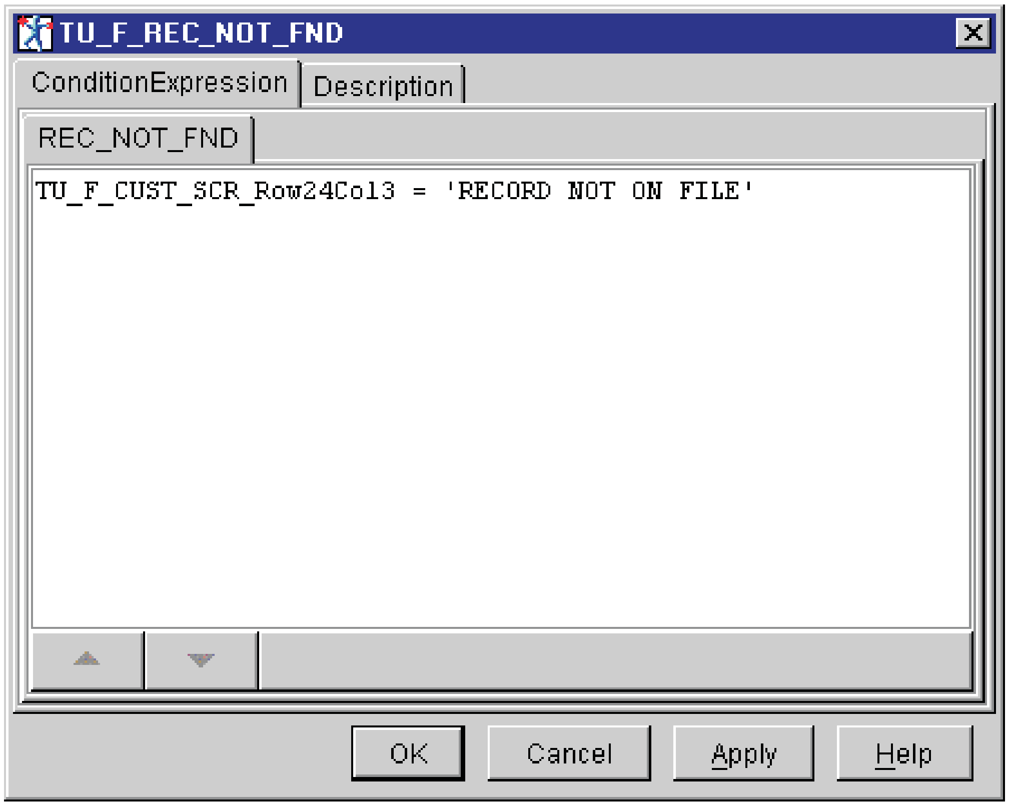


Figure 127. Editing the Out Terminal on the Decision type

4) Right click on the TU_F_REC_NOT_FND Decision type and select **Properties** on the pop up menu. Make sure the ConditionExpression tab is selected. Click in the REC_NOT_FND test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_F_CUST_SCR_screen message that we associated with the TU_F_REC_NOT_FND Decision type). Select the *TU_F_CUST_SCR_Row24Col3* field to add this to the ConditionExpression area.

   You should add the code shown in Figure 128 on page 167 for the REC_NOT_FND terminal test condition.

   Click **OK**.



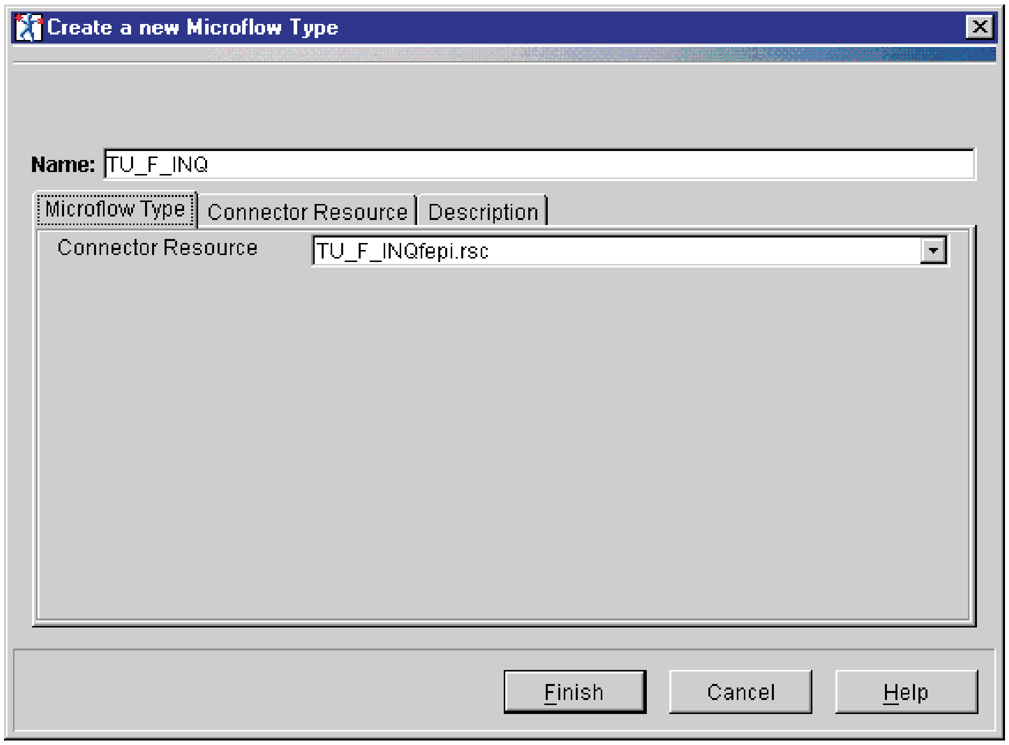
```
TU_F_CUST_SCR_Row24Col3 = 'RECORD NOT ON FILE'
```

Figure 128. Code for the REC_NOT_FND Terminal

c. **Create a microflow type**

Create a microflow that will model the processing of the customer data request.

1) Right click on the Microflow Types folder and select **Create > Microflow Type**.
2) Enter TU_F_INQ in the Name field.
3) Use the drop down menu in the Connector Resource field to select TU_F_INQfepi.rsc as the Connector Resource file and then click **Finish**.



Name: TU_F_INQ

Connector Resource: TU_F_INQfepi.rsc

Figure 129. Creating a TU_F_INQ Microflow Type

4) Save your workspace by selecting **File > Save Workspace** from the menubar.

You just created all of the component types that you will need to model the Inquiry subflow.

2. **Model the Inquiry subflow**.

   In this step you will perform a set of tasks to define and model the Inquiry subflow's functionality. The model represents the behavior of this subflow at run time.

   The subflow processing determines whether or not the requested customer record is found. You will add the nodes shown in Figure 131 on page 171.

   This step is made of the following tasks:

   - Adding subflow nodes
   - Connecting the subflow nodes
   - Defining the mappings

   a. **Add nodes to the subflow**

      In this task, you will drag the component types that you created in 1 on page 163. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

1) **Add the Input Terminal node**

An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

a) Drag the node on to the Microflow Definition pane.

In the Microflow Types folder, select the TU_F_INQ microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

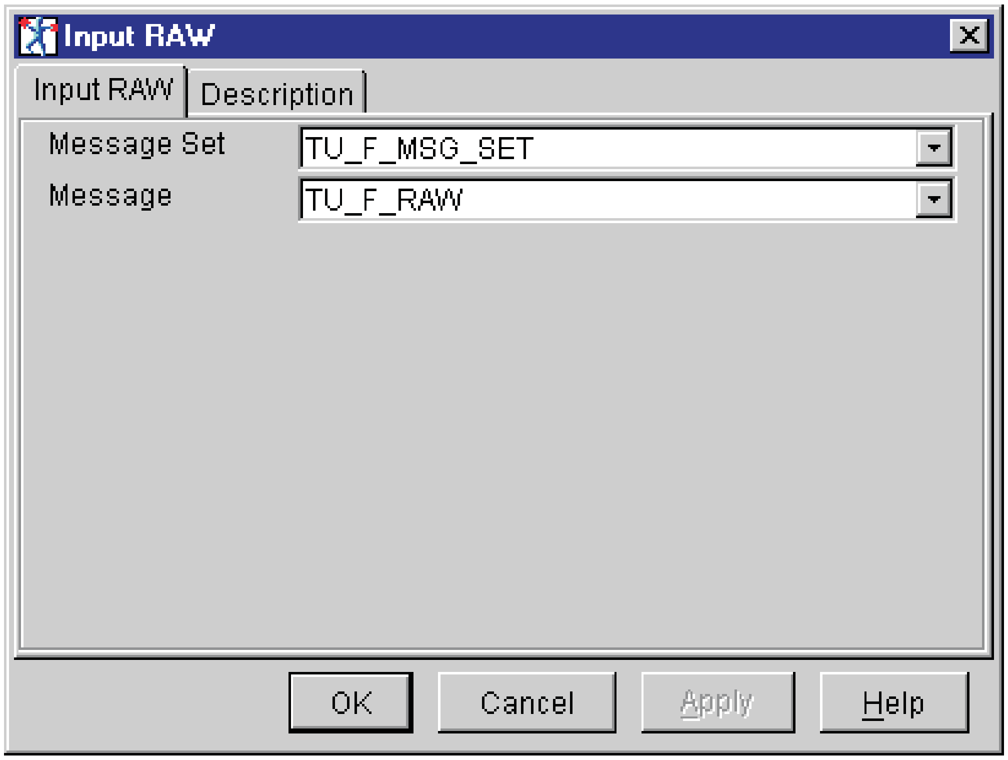
Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

b) Rename the node

Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input RAW and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_RAW in the Messages field. Click **OK**.



Message Set: TU_F_MSG_SET
Message: TU_F_RAW

Figure 130. Configuring the Input RAW Input Terminal node properties

2) **Add the Command node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_CUST Command type from the Adapter Tree View to the workspace. Place the node to the right on the Input RAW node. This command processes the Customer information screen.

b) Rename the node

Right click on the TU_F_CUST1 Command node and select **Rename**.

Modify TU_F_CUST1 in the New name field to the name TU_F_CUST and click **Finish**.

3) **Add the Decision node**

The TU_F_REC_NOT_FND Decision Node tests to see if the requested data record is found.

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_REC_NOT_FND Decision type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_CUST node.

b) Rename the node

Right click on the TU_F_REC_NOT_FND1 Command node and select **Rename**.

Modify TU_F_REC_NOT_FND1 in the New name field to the name TU_F_REC_NOT_FND and click **Finish**.

4) **Add the Output terminal node**

a) Drag the node on to the Microflow Definition pane

Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_REC_NOT_FND node.

b) Rename the node

Rename the Output Terminal node to Output REPLY. Click **Finish**

c) Flip the node

Right click on the Output REPLY node and select **Flip node**.

d) Set the properties for the node

Right click on the Output Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**.

5) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the Input RAW node and select **Connect > Out**. Move the connection line to the TU_F_CUST node and left click. This adds a control connection and a map (Map1 node) between the two nodes. Refer to Figure 131 on page 171.

2) Add a control connection from the second out terminal (Unknown) on the TU_F_CUST node to the Output REPLY node.

3) Add a Map node (Map2) to the control connection between the TU_F_CUST node and the Output REPLY node. To create a Map node,

drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before you release the mouse button.

4) Add a control connection from the first out terminal (TU_F_CUST_SCR_screen) on the TU_F_CUST node to the TU_F_REC_NOT_FND node.
5) Add a control connection from the first out terminal (REC_NOT_FND) on the TU_F_REC_NOT_FND node to the Output REPLY node.
6) Add a Map node (Map3) to the control connection between the TU_F_REC_NOT_FND node and the Output REPLY node.
7) Add a control connection from the second out terminal (default) on the TU_F_REC_NOT_FND node to the Output REPLY node.
8) Add a Map node (Map4) to the control connection between the TU_F_REC_NOT_FND node and the Output REPLY node.
9) Add a data connection from the Input RAW node to the Map3 node.
10) Save your workspace by selecting **File > Save Workspace** from the menubar.

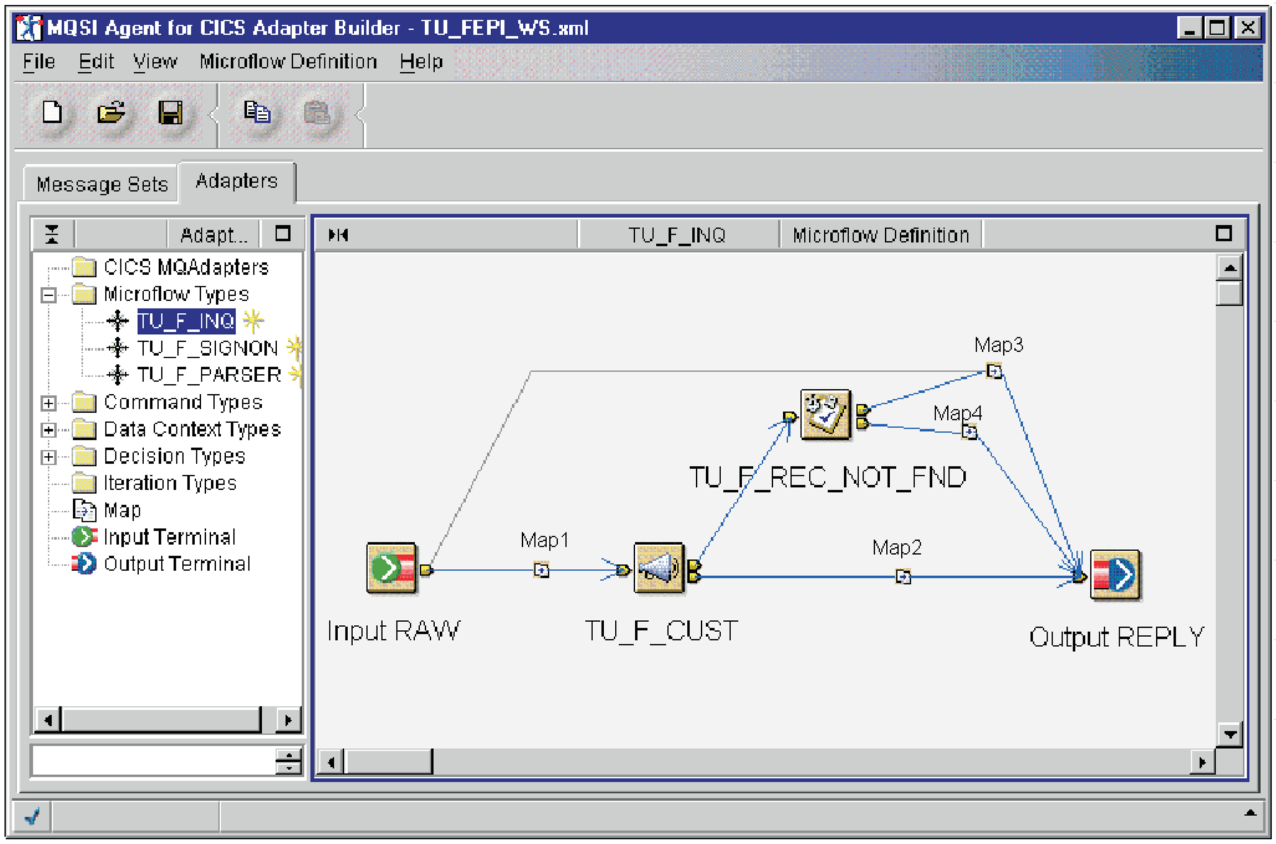


Figure 131. TU_F_INQ

c. **Map your subflow**

You are now ready to perform the data mappings for the TU_F_INQ subflow. Mapping models data transformation via a Map node between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.

- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 59 and shown in Figure 132 on page 173. This map sets the customer number field from the Customer screen to the customer number from the Input RAW record, hard codes a ’1’ for an Inquiry transaction and passes along the ENTERKEY.

   Right click on the Map1 node (the Map node that appears between the Input RAW and TU_F_CUST nodes) and select **Properties**. Click the DataMappingExpression tab.

   a) Left click on the CUST_NUM field under the TU_F_RAW message (view input message on right of panel) and drag the mouse cursor to the TU_F_CUST_SCR_Row3Col19 field under the TU_F_CUST_SCR_request message (view output message on left of panel). This will create a mapping between the two fields.

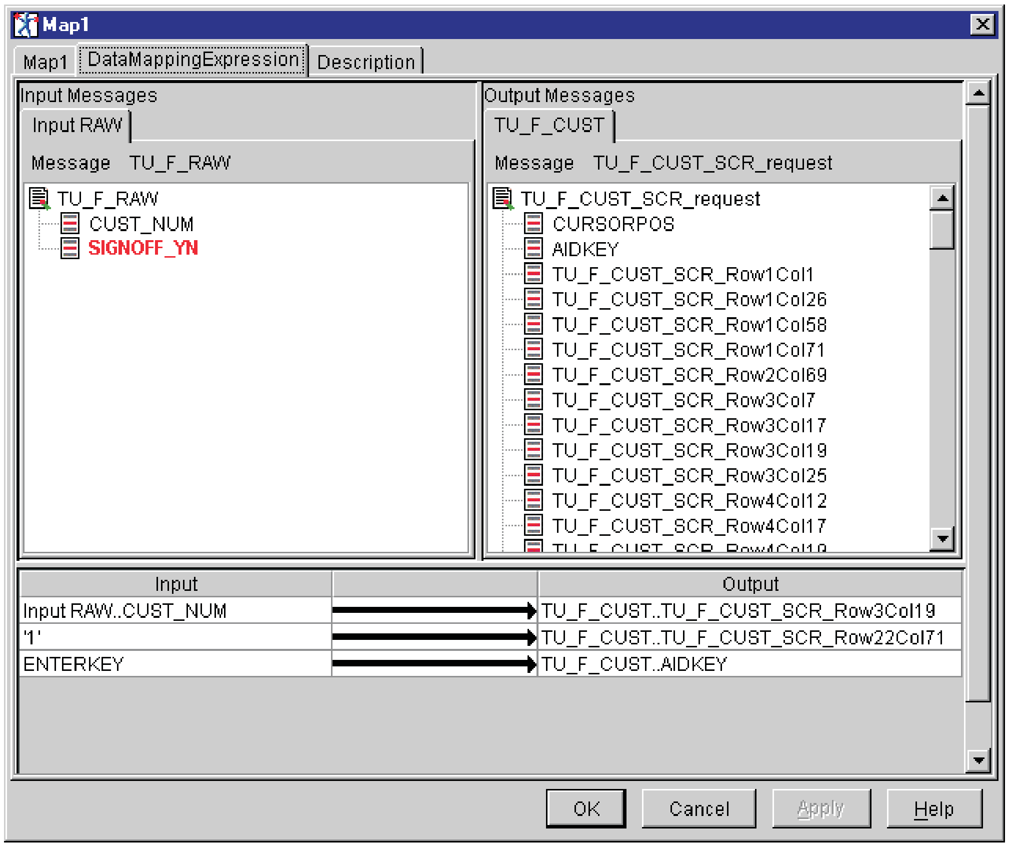
   b) Perform a literal mapping. Right click on the TU_F_CUST_SCR_Row22Col71 field in the TU_F_CUST_SCR_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ’1’ (quotes must be used).

   c) Perform a valid value mapping. Right click on the AIDKEY field in the TU_F_CUST_SCR_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select ENTERKEY in the Valid Value field.

Table 59. Mapping fields for Map1 node (TU_F_RAW message to TU_F_CUST_SCR_request message)

Input Field	Output Field	Description
CUST_NUM	TU_F_CUST_SCR_Row3Col19	Customer number for input record
’1’	TU_F_CUST_SCR_Row22Col71	Inquiry transaction
ENTERKEY	AIDKEY	Maps the enter key

   d) Click **OK** on the Map 1 dialog box.



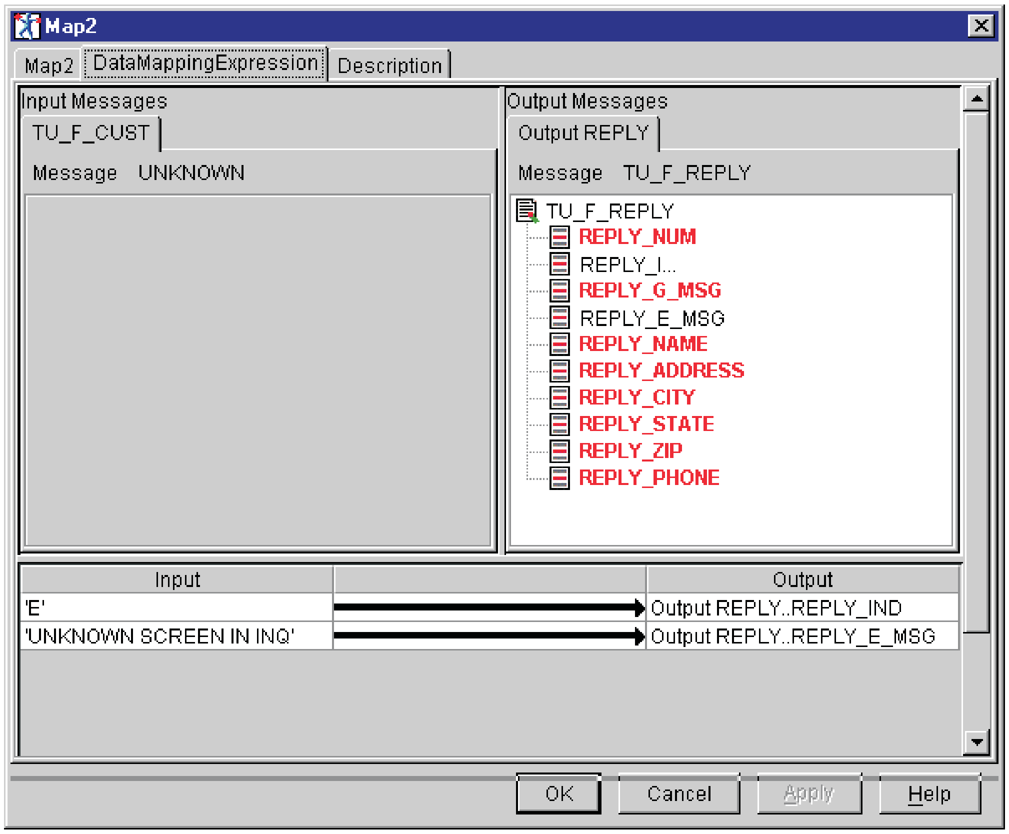
Input		Output
Input RAW..CUST_NUM	→	TU_F_CUST..TU_F_CUST_SCR_Row3Col19
'1'	→	TU_F_CUST..TU_F_CUST_SCR_Row22Col71
ENTERKEY	→	TU_F_CUST..AIDKEY

Figure 132. Mapping for Map1 node

2) Perform the mapping for the Map2 node as listed in Table 60 and shown in Figure 133 on page 174. This mapping occurs in the case where an Unknown screen is received from the TU_F_CUST command. These mappings set the REPLY_IND to 'E' and creates an error message. Use literal mappings to perform these field mappings, as was done in step 2c1c on page 172 of the **Define the Mappings** task.

Table 60. Mapping fields for Map2 node (Unknown message to TU_F_REPLY message)

Input Field	Output Field	Description
'E'	REPLY_IND	Sets REPLY_IND to 'E' to indicate an error
'UNKNOWN SCREEN IN INQ'	REPLY_E_MSG	Error message



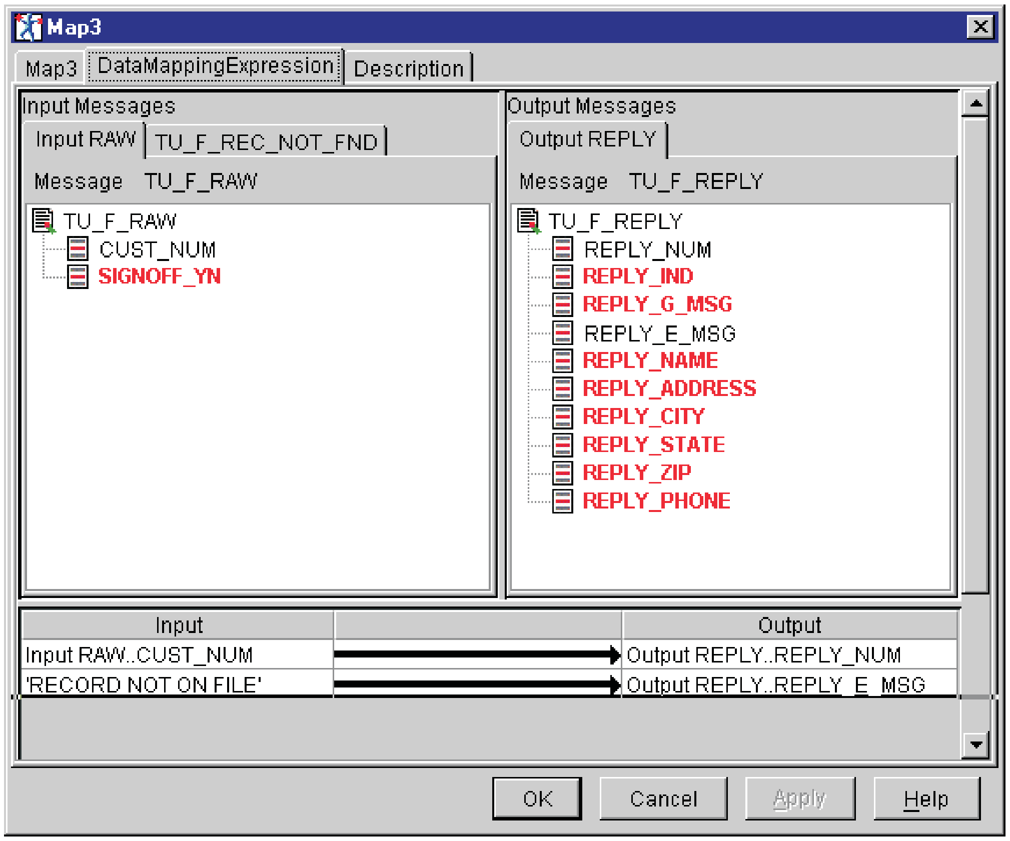
Input		Output
'E'	→	Output REPLY..REPLY_IND
'UNKNOWN SCREEN IN INQ'	→	Output REPLY..REPLY_E_MSG

Figure 133. Mapping for Map2 node

3) Click **OK** on the Map2 dialog box.
4) Perform the mapping for the Map3 node as listed in Table 61 and shown in Figure 134 on page 175. On a REC_NOT_FND condition, the mapper sets an error message and passes along the customer number to the REPLY record. Use literal mappings to perform these field mappings.

Table 61. Mapping fields for Map3 node (TU_F_RAW message to TU_F_REPLY message)

Input Field	Output Field	Description
CUST_NUM	REPLY_NUM	Passes the customer number to the reply
'RECORD NOT ON FILE'	REPLY_E_MSG	Error message



Input		Output
Input RAW..CUST_NUM	→	Output REPLY..REPLY_NUM
'RECORD NOT ON FILE'	→	Output REPLY..REPLY_E_MSG

Figure 134. Mapping for Map3 node

5) Click **OK** on the Map3 dialog box.
6) Perform the mapping for the Map4 node as listed in Table 62 and shown in Figure 135 on page 176. When a record is present, this mapper sets all the customer data to the REPLY record.

Table 62. Mapping fields for Map4 node (TU_F_CUST_SCR_screen message to TU_F_REPLY message)

Input Field	Output Field	Description
TU_F_CUST_SCR_Row3Col19	REPLY_NUM	Customer number
TU_F_CUST_SCR_Row4Col19	REPLY_NAME	Customer name
TU_F_CUST_SCR_Row5Col19	REPLY_ADDRESS	Customer address
TU_F_CUST_SCR_Row6Col19	REPLY_CITY	Customer city
TU_F_CUST_SCR_Row6Col49	REPLY_STATE	Customer state
TU_F_CUST_SCR_Row6Col64	REPLY_ZIP	Customer zip
TU_F_CUST_SCR_Row7Col21	REPLY_PHONE*	Telephone area code
TU_F_CUST_SCR_Row7Col28	REPLY_PHONE*	Telephone local exchange
TU_F_CUST_SCR_Row7Col34	REPLY_PHONE*	Telephone number (last 4 digits)
* This field is a concatenation of three input fields.		

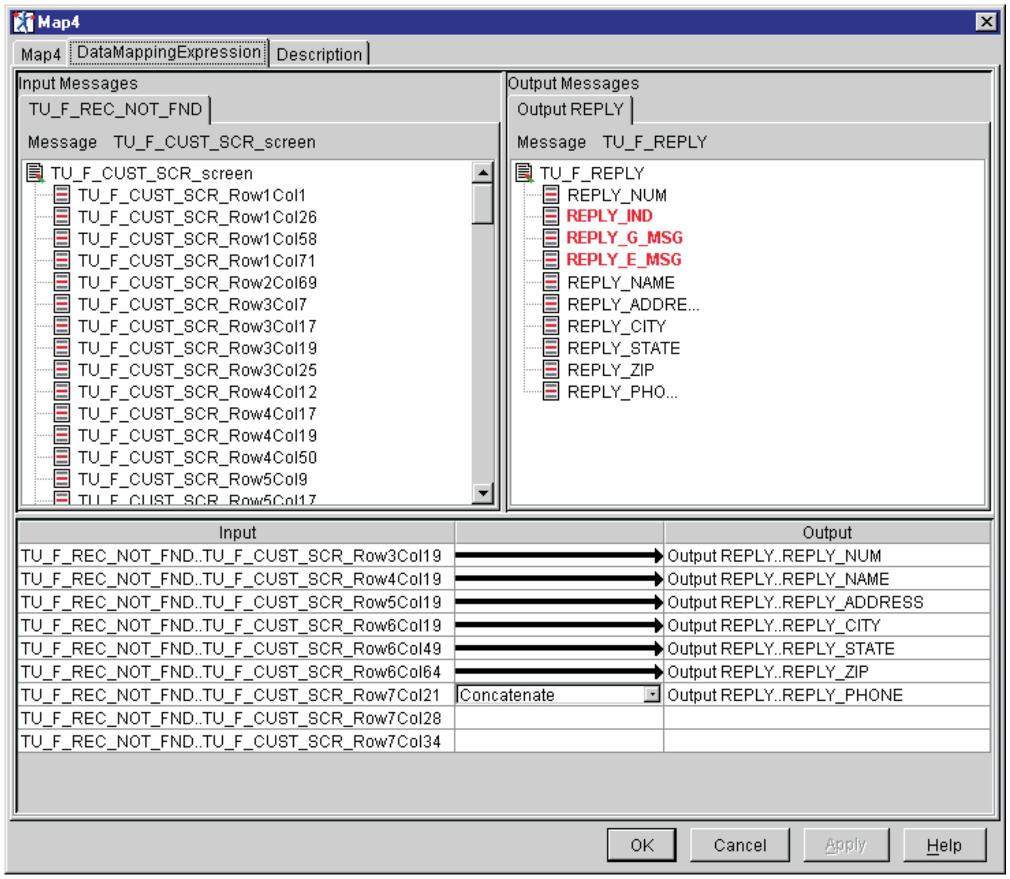
The REPLY_PHONE output field is a concatenation of three input fields. To perform this mapping:

a) In the Input section, left click the TU_F_CUST_SCR_Row7Col21 field and drag to the REPLY_PHONE field in the Output section.

b) In the Input section, drag a second input data field (TU_F_CUST_SCR_Row7Col28) and drop it on the previously mapped input data field (TU_F_CUST_SCR_Row7Col21) in the Inputs section. The mapping arrow will change to a drop down menu after the second field is mapped.

c) Select **Concatenate** from the drop down menu.

d) In the Input section, drag a third input data field (TU_F_CUST_SCR_Row7Col34) and drop it on the previously mapped input data field (TU_F_CUST_SCR_Row7Col28) in the Inputs section. This is done to concatenate the fields in the proper sequence.



Input		Output
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row3Col19	→	Output REPLY..REPLY_NUM
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row4Col19	→	Output REPLY..REPLY_NAME
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row5Col19	→	Output REPLY..REPLY_ADDRESS
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row6Col19	→	Output REPLY..REPLY_CITY
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row6Col49	→	Output REPLY..REPLY_STATE
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row6Col64	→	Output REPLY..REPLY_ZIP
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row7Col21	Concatenate	Output REPLY..REPLY_PHONE
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row7Col28		
TU_F_REC_NOT_FND..TU_F_CUST_SCR_Row7Col34		

Figure 135. Mapping for Map4 node

e) Click **OK**.

You just modelled the Inquiry subflow.

In your model, you have coded the instructions on how the subflow is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

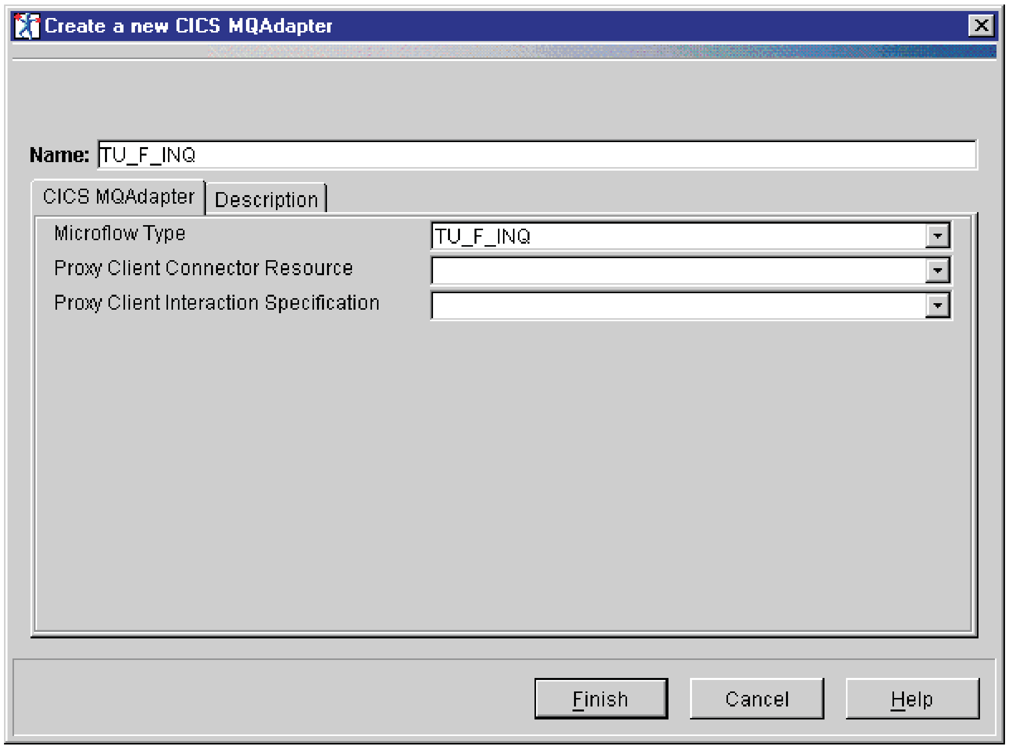
3. **Assign the model of the subflow to a CICS MQAdapter**.

   In this step you will associate the Inquiry subflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual generation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_INQ for the Name and use the drop down menu to select TU_F_INQ for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank.



Name: TU_F_INQ

Microflow Type: TU_F_INQ

Proxy Client Connector Resource:

Proxy Client Interaction Specification:

Figure 136. Creating an CICS MQAdapter

   c. Click **Finish**.

   You have completed the Inquiry subflow and setup of this segment of your adapter.

   Save your workspace by selecting **File > Save Workspace** from the menubar.

   Now you are ready to generate your adapter.

   Note: The copybooks for the Inquiry microflow were previously generated during the parser subflow (TU_F_PARSER) adapter modeling section (see 2a on page 148).

   Now you are ready to generate your adapter.

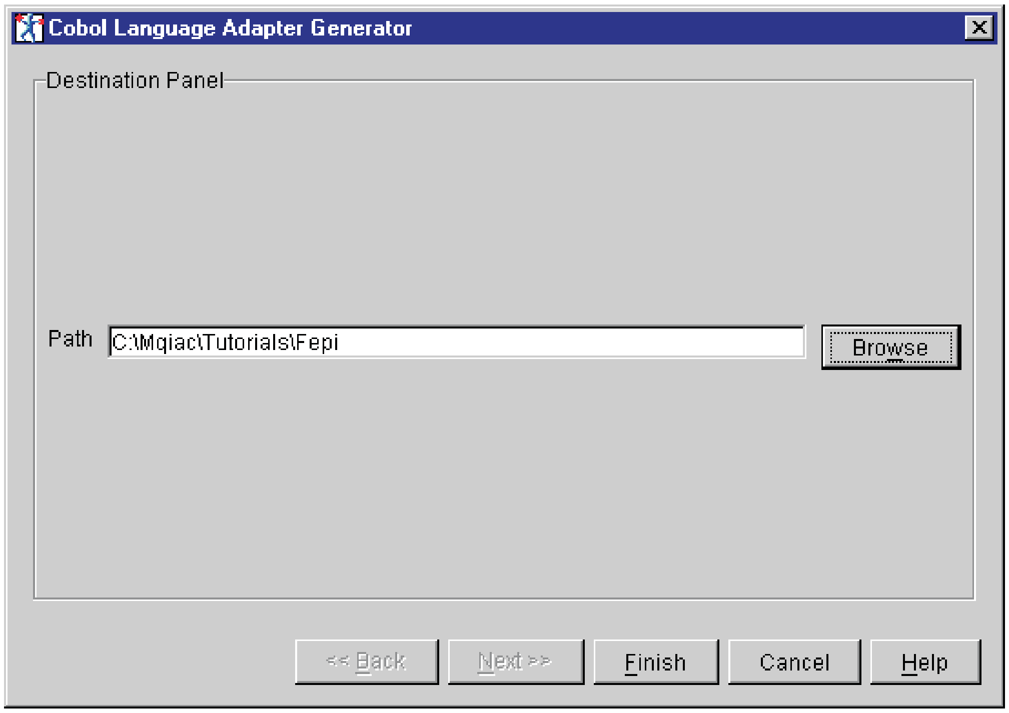
4. **Generate adapter Code**.

   To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

   Note: You must generate the adapter code in the same directory where you generated the copybooks.

   a. Right click on the TU_F_INQ adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

      The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\Tutorials\Fepi

Figure 137. Specifying pathname for adapter code generation output

Create the Signoff subflow

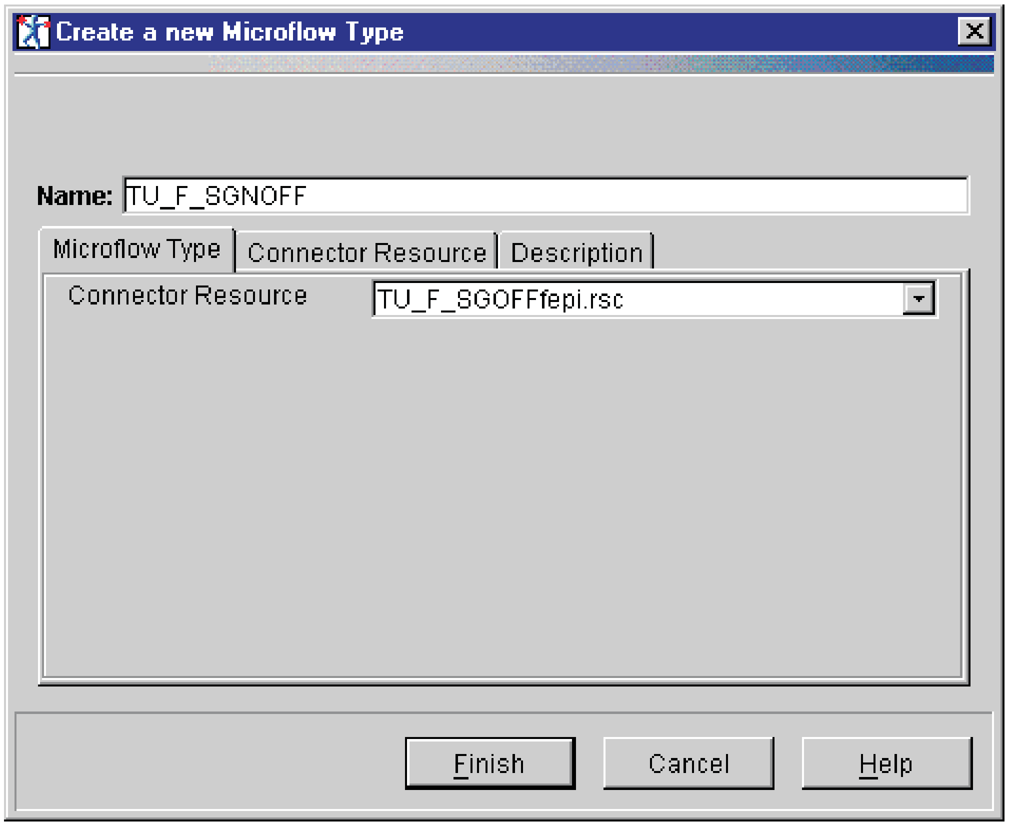
The Signoff subflow is used to sign a user off the Customer application and put the user on a blank CICS screen.

Follow these steps to create the Signoff subflow:

1. **Create the microflow component type for use in the Signoff subflow**.

   Create a microflow that will model the processing for the user signoff request.

   a. Right click on the Microflow Types folder and select **Create > Microflow Type**.
   b. Enter TU_F_SGNOFF in the Name field.
   c. Use the drop down menu in the Connector Resource field to select TU_F_SGOFFfepi.rsc as the Connector Resource file and then click **Finish**.



Name: TU_F_SGNOFF

Connector Resource: TU_F_SGOFFfepi.rsc

Figure 138. Creating a TU_F_SGNOFF Microflow Type

You just created the one component type that you will need to model the Signoff subflow.

2. **Model the Signoff subflow**

   In this step you will perform a set of tasks to define and model the Signoff subflow's functionality. The model represents the behavior of this subflow at run time.

   This step is made of the following tasks:
   - Adding the subflow nodes
   - Connecting the subflow nodes
   - Defining the mappings

   a. **Add nodes to the subflow**

      1) **Add the Input Terminal node**

         An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

a) Drag the node on to the Microflow Definition pane.

In the Microflow Types folder, select the TU_F_SGNOFF microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

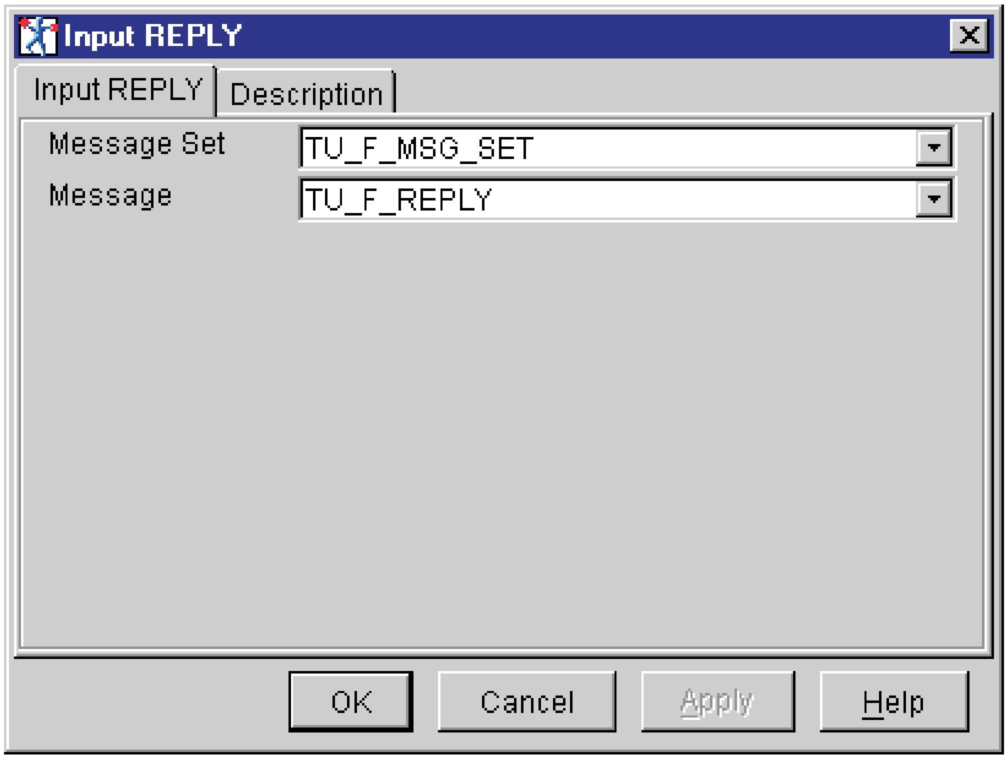
Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

b) Rename the node

Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input REPLY and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**.



Message Set: TU_F_MSG_SET
Message: TU_F_REPLY

Figure 139. Configuring the Input REPLY Input Terminal node properties

2) **Add the TU_F_CUST Command node**

In this step you will add the command node that processes the Customer information screen.

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_CUST Command type from the Adapter Tree View to the workspace. Place the node to the right on the Input REPLY node.

b) Rename the node

Right click on the TU_F_CUST1 Command node and select **Rename**.

Modify TU_F_CUST1 in the New name field to the name TU_F_CUST and click **Finish**.

3) **Add the TU_F_CMAV Command node**

   In this step you will add the command node that processes the CESF Logoff transaction.

   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_CMAV Command type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_CUST node.

   b) Rename the node

      Right click on the TU_F_CMAV1 Command node and select **Rename**

      Modify TU_F_CMAV1 in the New name field to the name CESF_LOGOFF and click **Finish**

4) **Add the Output terminal node**

   a) Drag the node on to the Microflow Definition pane

      Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the CESF_LOGOFF node.

   b) Rename the node

      Rename the Output Terminal node to Output REPLY.

   c) Flip the node

      Right click on the Output REPLY node and select **Flip node**.

   d) Set the properties for the node

      Right click on the Output REPLY and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**

5) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

   In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

   1) Right click on the Input REPLY node and select **Connect > Out**. Move the connection line to the TU_F_CUST node and left click. This adds a control connection and a map (Map1 node) between the two nodes. Refer to Figure 140 on page 182.
   2) Add a Map node (Map2) between the TU_F_CUST node and the CESF_LOGOFF node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before releasing the mouse button.
   3) Add a control connection from the second out terminal (Unknown) on the TU_F_CUST node to the Map2 node and from the Map2 node to the CESF_LOGOFF node.
   4) Add a control connection from the first out terminal (TU_F_CUST_SCR_screen) on the TU_F_CUST node to the Output REPLY node. The auto-adds a Map3 node on the connection.

5) Add a control connection from the first out terminal (TU_F_CUST_SCR_screen) on the CESF_LOGOFF node to the Map3 node.
6) Add a Map node (Map4) between the CESF_LOGOFF node and the Output REPLY node.
7) Add a control connection from the second out terminal (Unknown) on the CESF_LOGOFF node to the Map4 node and from the Map4 node to the Output REPLY node.
8) Add a data connection from the Input REPLY node to the Map3 node.
9) Add a data connection from the Input REPLY node to the Map4 node.
10) Save your workspace by selecting **File > Save Workspace** from the menubar.

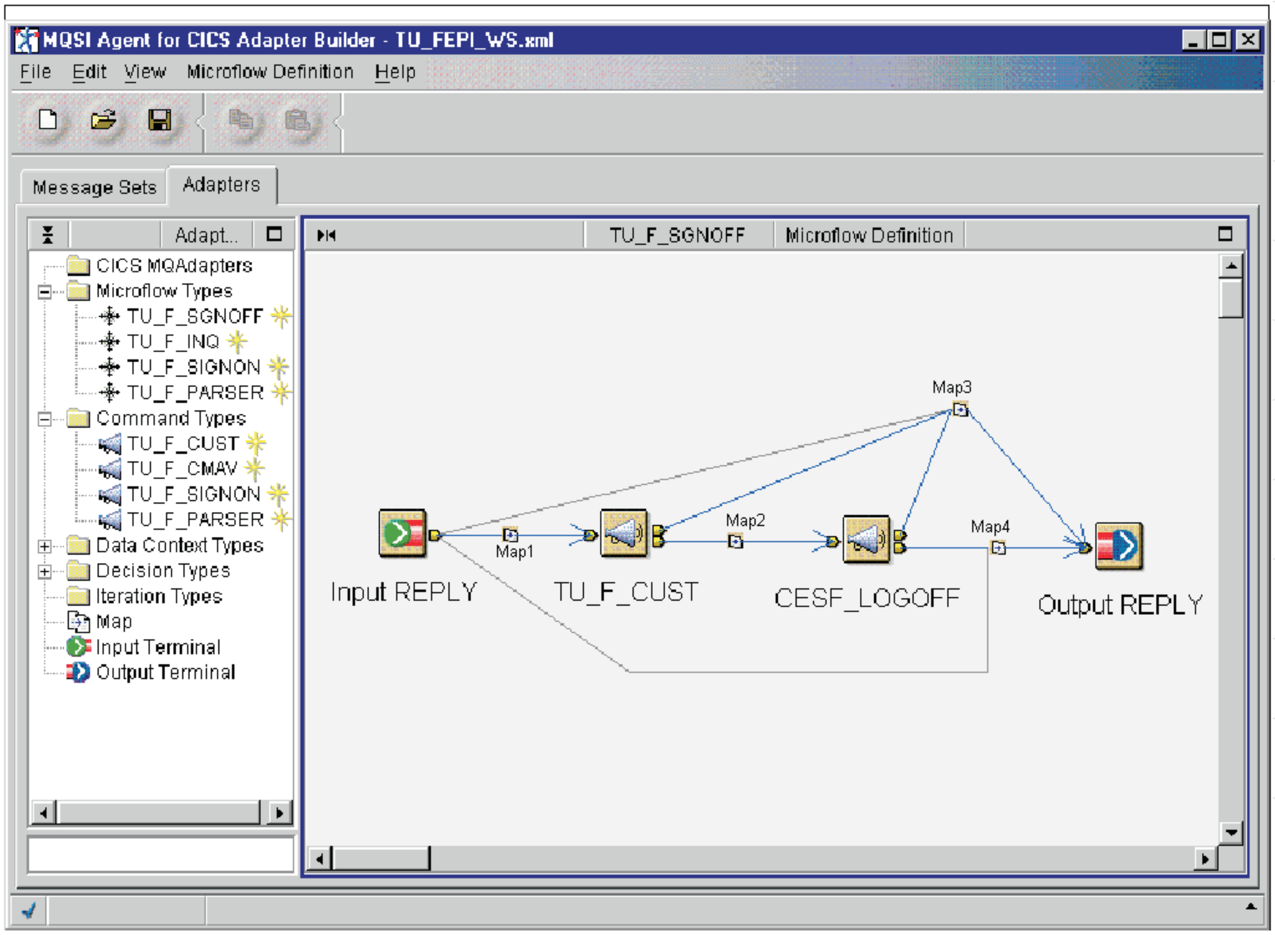


Figure 140. TU_F_SGNOFF

c. **Map your subflow**

You are now ready to perform the data mappings for the TU_F_SGNOFF subflow. Mapping models data transformation via a Map node between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

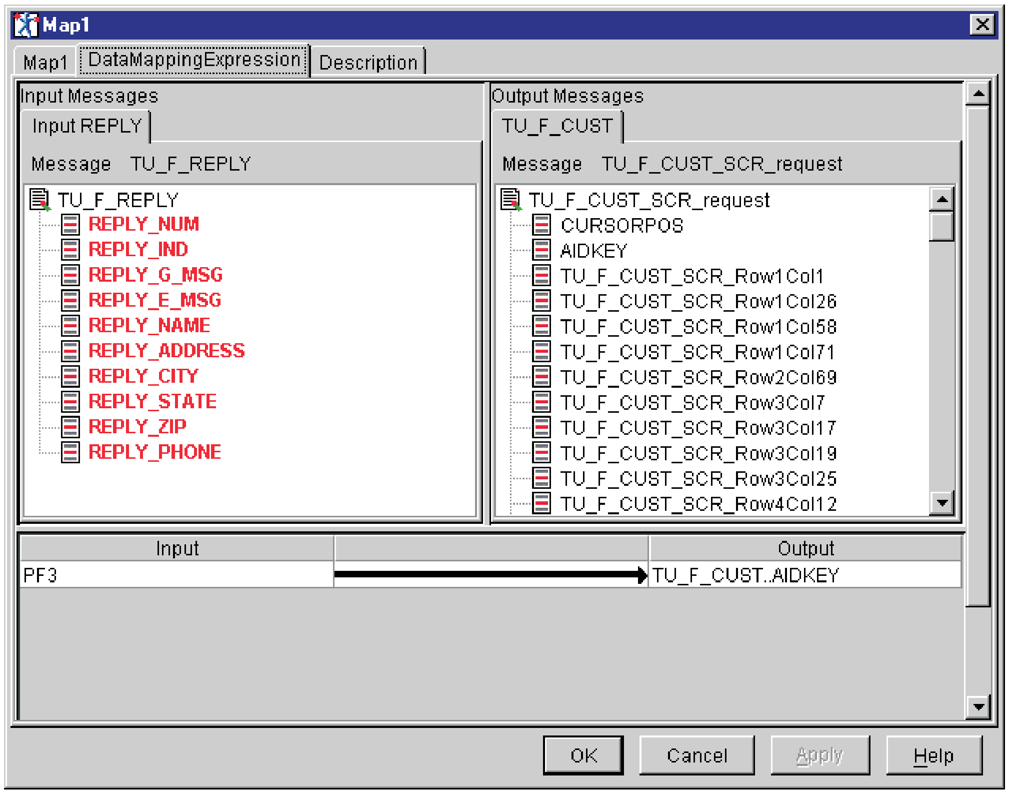
- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.

- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 63 and shown in Figure 141. This map passes a PF3 key to the Customer information screen to logoff from the application and return to a blank CICS screen.
   a) Right click on the Map1 node (the Map node that appears between the Input REPLY and TU_F_CUST nodes) and select **Properties**. Click the DataMappingExpression tab.
   b) Perform a valid value mapping.
      - Right click on the destination field for the AIDKEY field in the TU_F_CUST_SCR_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.
      - Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select PF3 in the Valid Value field. Click **OK**

Table 63. Mapping fields for Map1 node (TU_F_REPLY message to TU_F_CUST_SCR_request message)

Input Field	Output Field	Description
PF3	AIDKEY	Maps the PF3 key



Message TU_F_REPLY

- TU_F_REPLY
  - REPLY_NUM
  - REPLY_IND
  - REPLY_G_MSG
  - REPLY_E_MSG
  - REPLY_NAME
  - REPLY_ADDRESS
  - REPLY_CITY
  - REPLY_STATE
  - REPLY_ZIP
  - REPLY_PHONE

Message TU_F_CUST_SCR_request

- TU_F_CUST_SCR_request
  - CURSORPOS
  - AIDKEY
  - TU_F_CUST_SCR_Row1Col1
  - TU_F_CUST_SCR_Row1Col26
  - TU_F_CUST_SCR_Row1Col58
  - TU_F_CUST_SCR_Row1Col71
  - TU_F_CUST_SCR_Row2Col69
  - TU_F_CUST_SCR_Row3Col7
  - TU_F_CUST_SCR_Row3Col17
  - TU_F_CUST_SCR_Row3Col19
  - TU_F_CUST_SCR_Row3Col25
  - TU_F_CUST_SCR_Row4Col12

Input		Output
PF3	→	TU_F_CUST..AIDKEY

Figure 141. Mapping for Map1 node

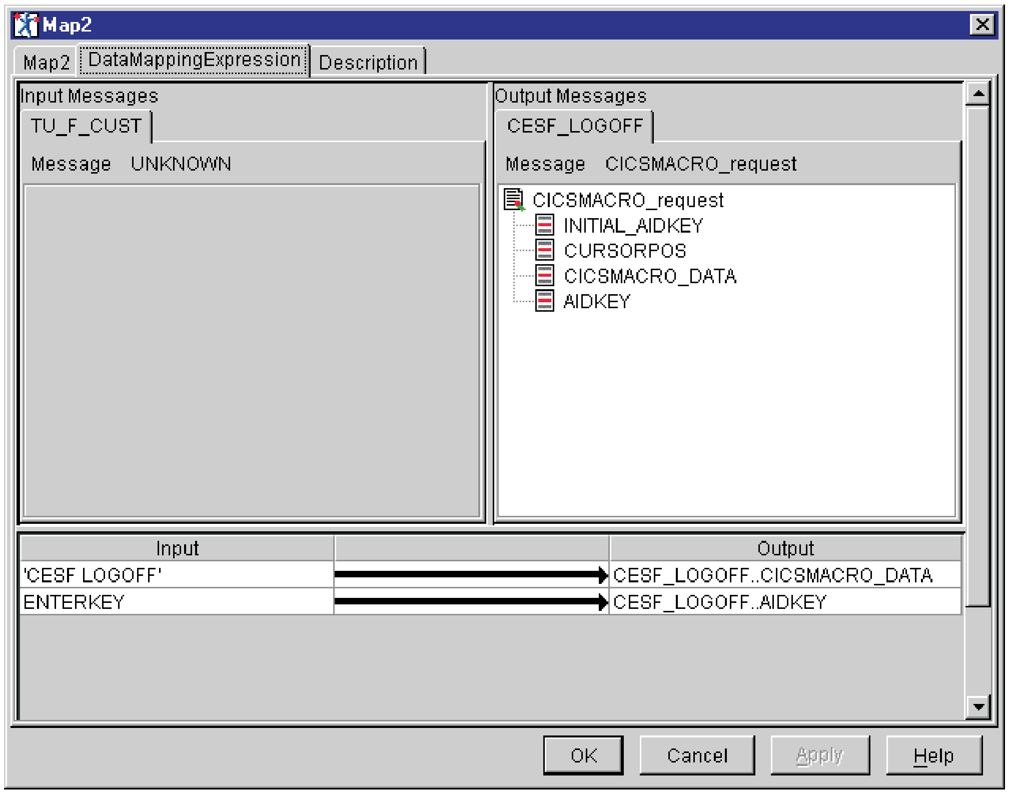
2) Perform the mapping for the Map2 node as listed in Table 64 and shown in Figure 142 on page 185. This mapping occurs in the case where an Unknown screen is received from the TU_F_CUST command. The mapping sets the REPLY_IND to ’E’ and creates an error message.

   Right click on the Map2 node (the Map node that appears between the TU_F_CUST and CESF_LOGOFF nodes) and select **Properties**. Click the DataMappingExpression tab.

   a) Perform a literal mapping.
      i. Right click on the destination field for the CICSMACRO_DATA field in the CICSMACRO_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.
      ii. Double click on LITERAL field and rename it to ’CESF LOGOFF’ (quotes must be used around the CESF LOGOFF string).
   b) Perform a valid value mapping.
      i. Right click on the destination field for the AIDKEY field in the CICSMACRO_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.
      ii. Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select ENTERKEY in the Valid Value field. Click **OK**.

Table 64. Mapping fields for Map2 node (Unknown message to CICSMACRO_request message)

Input Field	Output Field	Description
’CESF LOGOFF’	CICSMACRO_DATA	Maps the ’CESF’ transaction to signoff of CICS system
ENTERKEY	AIDKEY	Maps the enter key



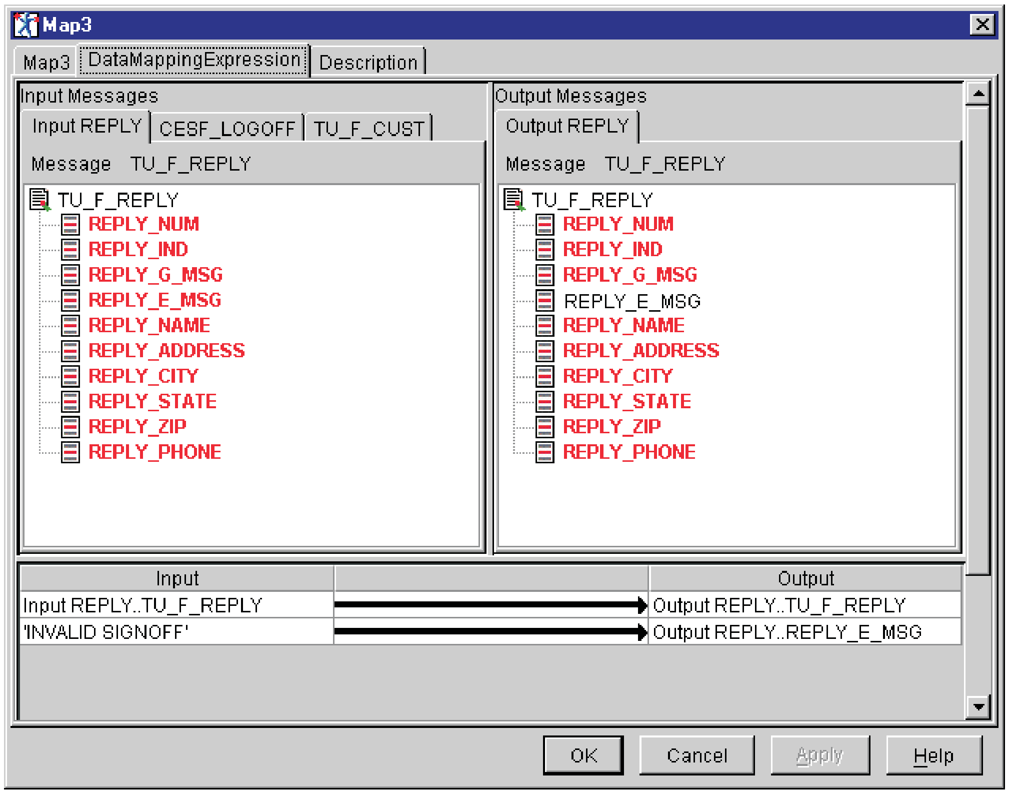
Input		Output
'CESF LOGOFF'	→	CESF_LOGOFF..CICSMACRO_DATA
ENTERKEY	→	CESF_LOGOFF..AIDKEY

Figure 142. Mapping for Map2 node

3) Perform the mapping for the Map3 node as listed in Table 65 and shown in Figure 143 on page 186. On an Invalid Signoff condition, the mapper passes along the REPLY record and sets a message stating INVALID SIGNOFF.

Table 65. Mapping fields for Map3 node (TU_F_REPLY message to TU_F_REPLY message)

Input Field	Output Field	Description
TU_F_REPLY	TU_F_REPLY	Passes the REPLY record
'INVALID SIGNOFF'	REPLY_E_MSG	Error message



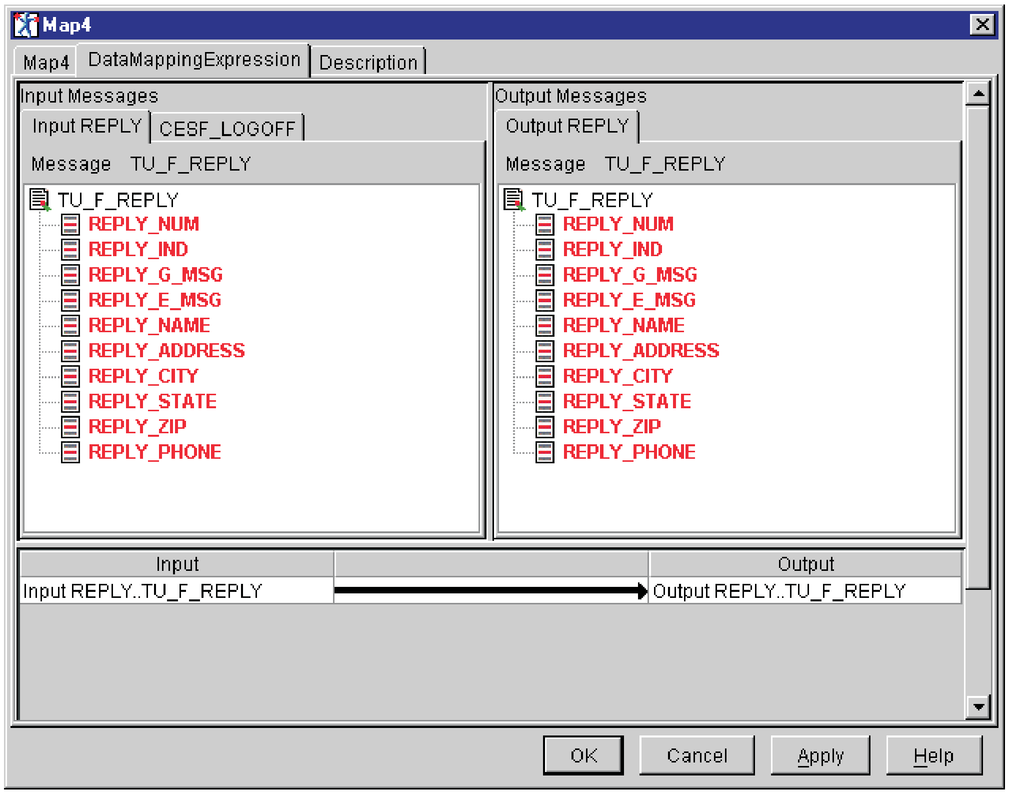
Input		Output
Input REPLY..TU_F_REPLY	→	Output REPLY..TU_F_REPLY
'INVALID SIGNOFF'	→	Output REPLY..REPLY_E_MSG

Figure 143. Mapping for Map3 node

4) Perform the mapping for the Map4 node as listed in Table 66 and shown in Figure 144 on page 187. On an Valid Signoff condition, the mapper passes along the REPLY record.

Table 66. Mapping fields for Map4 node (TU_F_REPLY message to TU_F_REPLY message)

Input Field	Output Field	Description
TU_F_REPLY	TU_F_REPLY	Passes the REPLY record



Message TU_F_REPLY

TU_F_REPLY
- REPLY_NUM
- REPLY_IND
- REPLY_G_MSG
- REPLY_E_MSG
- REPLY_NAME
- REPLY_ADDRESS
- REPLY_CITY
- REPLY_STATE
- REPLY_ZIP
- REPLY_PHONE

Input		Output
Input REPLY..TU_F_REPLY	→	Output REPLY..TU_F_REPLY

Figure 144. Mapping for Map4 node

You just modelled the Signoff subflow.

In your model, you have coded the instructions on how the subflow is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

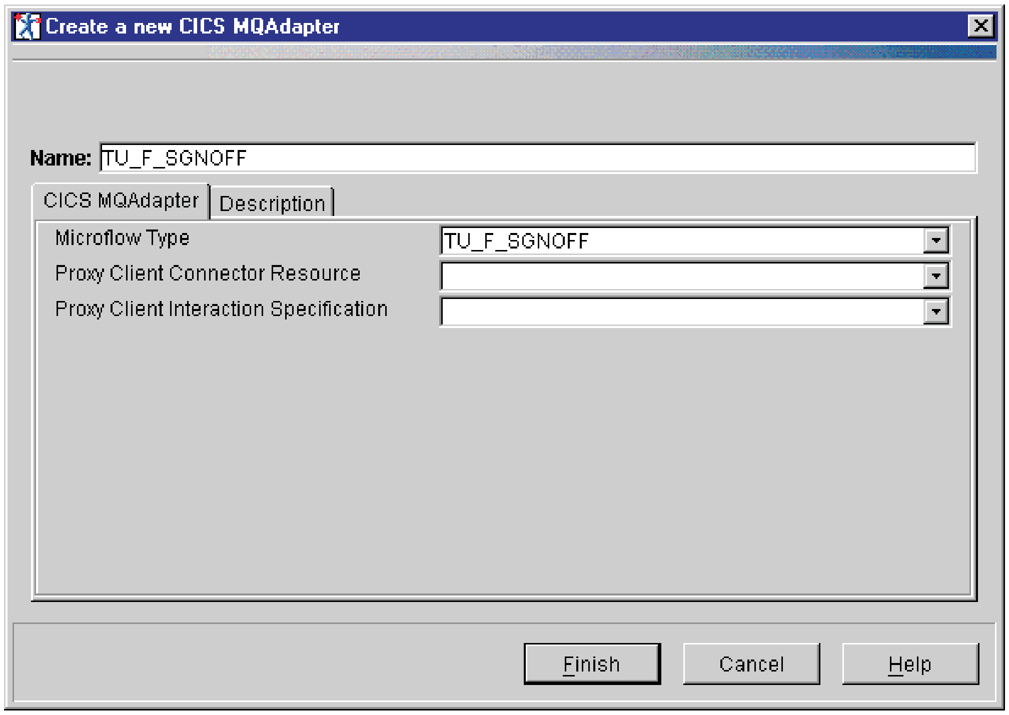
3. **Assign the model of the subflow to a CICS MQAdapter**.

   In this step you will associate the Signoff subflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual implementation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_SGNOFF for the Name and use the drop down menu to select TU_F_SGNOFF for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TU_F_SGNOFF

Microflow Type: TU_F_SGNOFF

Proxy Client Connector Resource:

Proxy Client Interaction Specification:

Figure 145. Creating an CICS MQAdapter

You have completed the microflow and setup your adapter. Save your workspace by selecting **File > Save Workspace** from the menubar. Now you are ready to generate your adapter.

Note: The copybooks for the SIGNOFF microflow were previously generated during the parser subflow (TU_F_PARSER) adapter modeling section (see 2a on page 148).

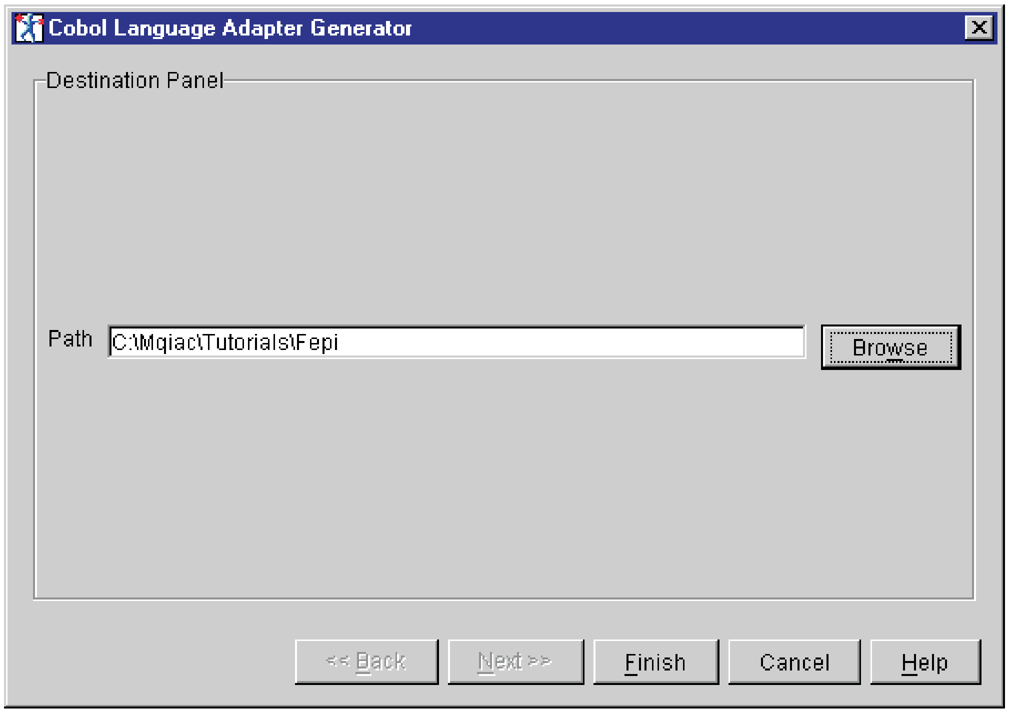
4. **Generate adapter Code**.

   To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

   Note: You must generate the adapter code in the same directory where you generated the copybooks.

   a. Right click on the TU_F_SGNOFF adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination **<mqiac_tutorials>\fepi** in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

      The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\Tutorials\Fepi

Figure 146. Specifying pathname for adapter code generation output

Create the Reset subflow

The Reset subflow is used to reset the Customer screen for inputting the next customer number.

Follow these steps to create the Reset subflow:

1. **Create the microflow component type for use in the Reset subflow**

   Create a microflow that will model the processing for the Customer screen reset.

   a. Right click on the Microflow Types folder and select **Create > Microflow Type**.

   b. Enter TU_F_RESET in the Name field.

   c. Use the drop down menu in the Connector Resource field to select TU_F_RESETfepi.rsc as the Connector Resource file and then click **Finish**.

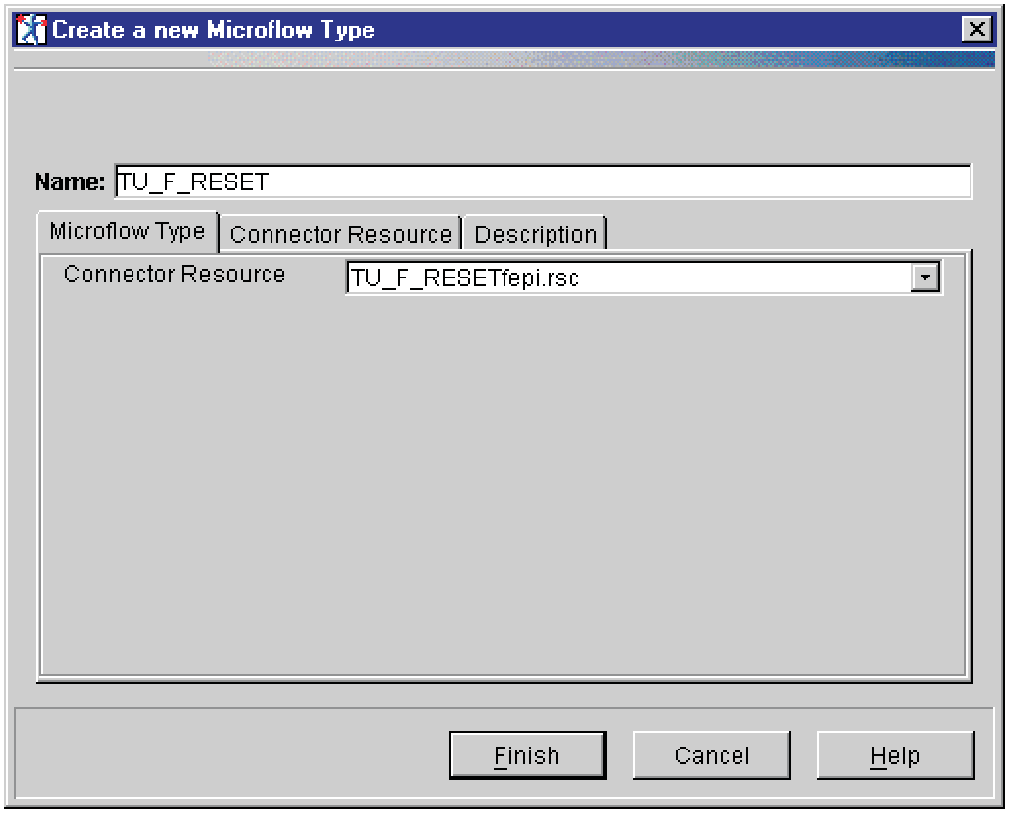


Figure 147. Creating a TU_F_RESET Microflow Type

You just created the one component type that you will need to model the Reset subflow.

2. **Model the Reset subflow**

   Now you will begin to define and model the Reset subflow's functionality.

   The subflow processing determines whether or not to reset the Customer screen.

   This step is made of the following tasks:

   - Adding subflow nodes
   - Connecting the subflow nodes
   - Defining the mappings

   a. **Add nodes to the subflow**

      You will add the nodes shown in Figure 149 on page 193.

      1) **Add the Input Terminal node**

         An *Input Terminal* serves as an entry point for the microflow. The Input Terminal can make a connection to any terminal that resides within the microflow.

a) Drag the node on to the Microflow Definition pane.

In the Microflow Types folder, select the TU_F_RESET microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

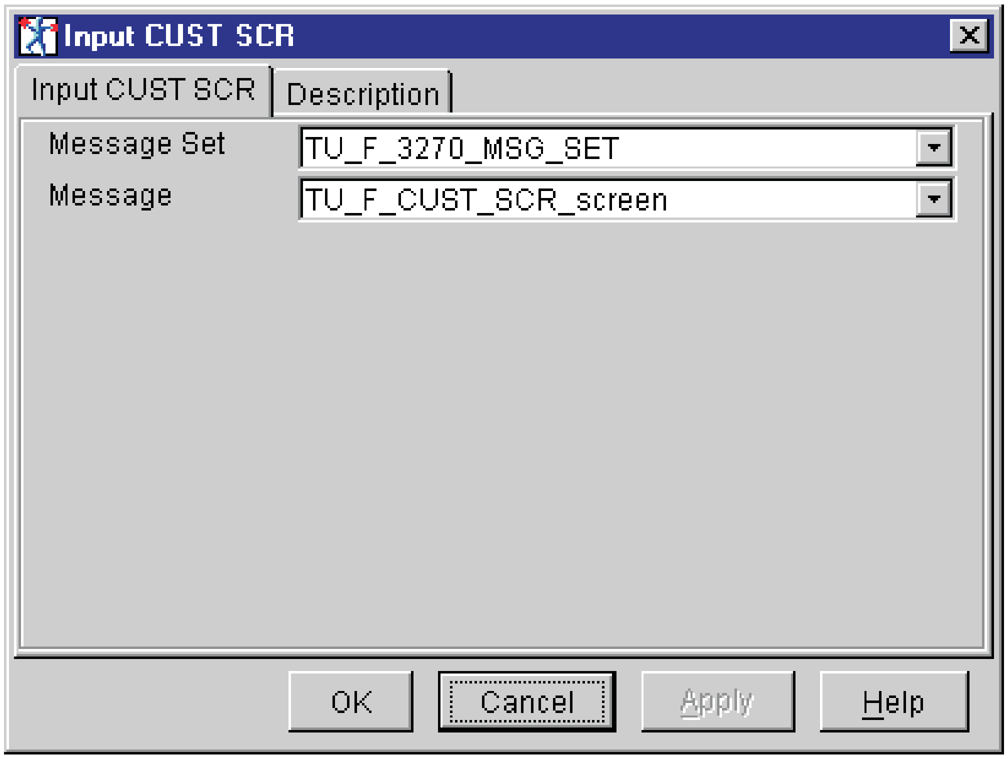
Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

b) Rename the node

Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input CUST SCR and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_3270_MSG_SET in the Message Sets field and select TU_F_CUST_SCR_screen in the Messages field. Click **OK**.



Message Set: TU_F_3270_MSG_SET

Message: TU_F_CUST_SCR_screen

Figure 148. Configuring the Input CUST SCR Input Terminal node properties

2) **Add the Command node**

This command processes the Customer information screen.

a) Drag the node on to the Microflow Definition pane

From the Command Types folder in the Adapter Tree View, select a TU_F_CUST Command type.

Left click and hold on the TU_F_DCUST Command type to drag it to the Microflow Definition pane. Place the node to the right on the Input CUST SCR node.

b) Rename the node

Right click on the TU_F_CUST1 Command node and select **Rename**. Modify TU_F_CUST1 in the New name field to the name TU_F_CUST and click **Finish**.

3) **Add the Output terminal node**

a) Drag the node on to the Microflow Definition pane

Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_CUST node.

b) Rename the node

Rename the Output Terminal node to Output REPLY.

c) Flip the node

Right click on the Output REPLY node and select **Flip node**

d) Set the properties for the node

Right click on the Output REPLY and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**.

.

4) **Add a second Output terminal node**

a) Drag the node on to the Microflow Definition pane

Drag a second Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_CUST node and above the Output REPLY node.

b) Rename the node

Rename the Output Terminal node to Output CUST SCR.

c) Flip the node

Right click on the Output CUST SCR node and select **Flip node**

d) Set the properties for the node

Right click on the Output CUST SCR and select **Properties**. From the drop down menus, select TU_F_3270_MSG_SET in the Message Sets field and select TU_F_CUST_SCR_screen in the Messages field. Click **OK**.

5) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the Input CUST SCR node and select **Connect > Out**. Move the connection line to the TU_F_CUST node and left click. This adds a control connection and a map (Map1 node) between the two nodes. Refer to Figure 149 on page 193.

2) Add a control connection from the first out terminal (TU_F_CUST_SCR_screen) on the TU_F_CUST node to the Output CUST SCR node.
3) Add a Map node (Map2) between the TU_F_CUST node and the Output REPLY node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor slightly below the two nodes before releasing the mouse button.
4) Add a control connection from the second out terminal (Unknown) on the TU_F_CUST node to the Map2 node and from the Map2 node to the Output REPLY node.
5) Save your workspace by selecting **File > Save Workspace** from the menubar.

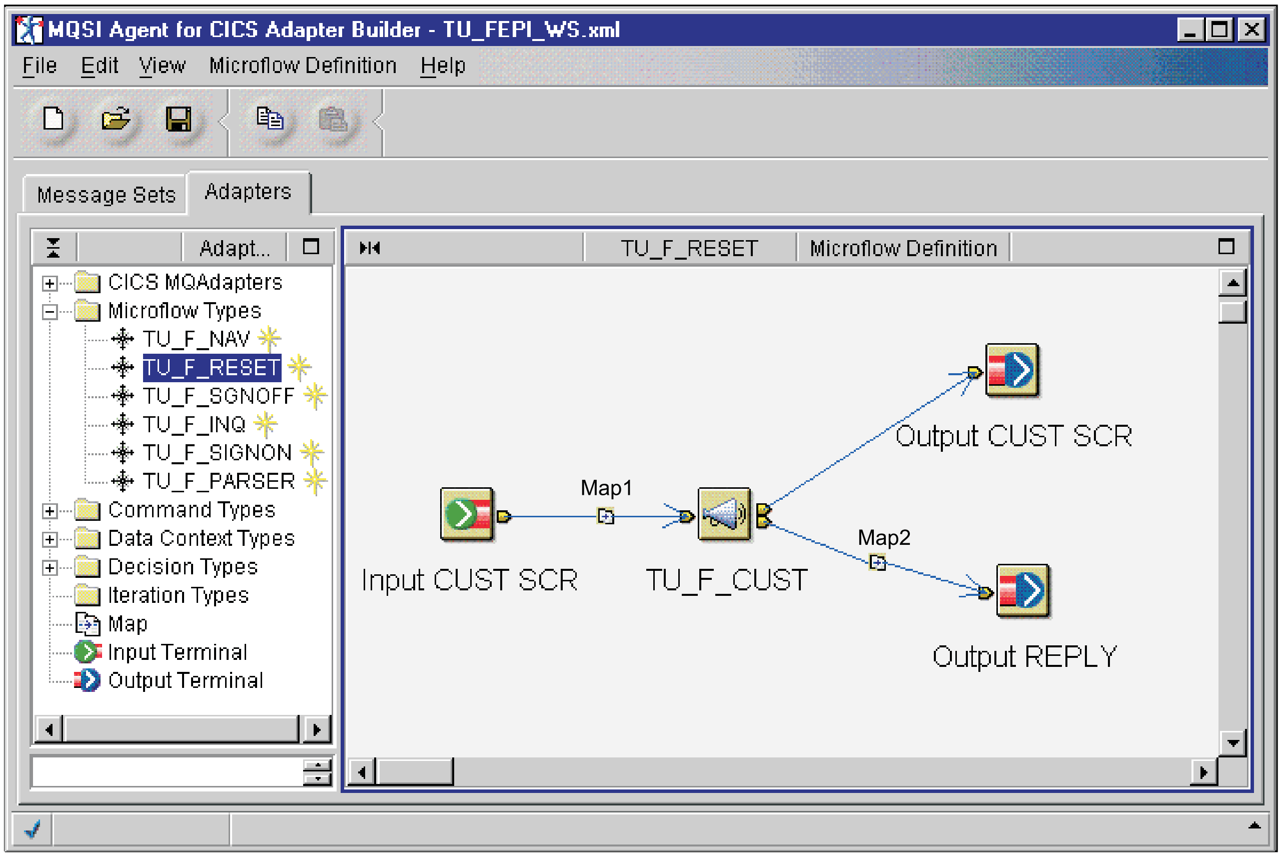


Figure 149. TU_F_RESET Microflow

c. **Map your subflow**

You are now ready to perform the data mappings for the TU_F_RESET subflow. Mapping models data transformation via a Map node between an output terminal on one node and an input terminal on another node. Data transformation can include a variety of functions:

- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.

- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as listed in Table 67 and shown in Figure 150. This maps the PF12 key to prepare the Customer information screen for input of the next Customer number.

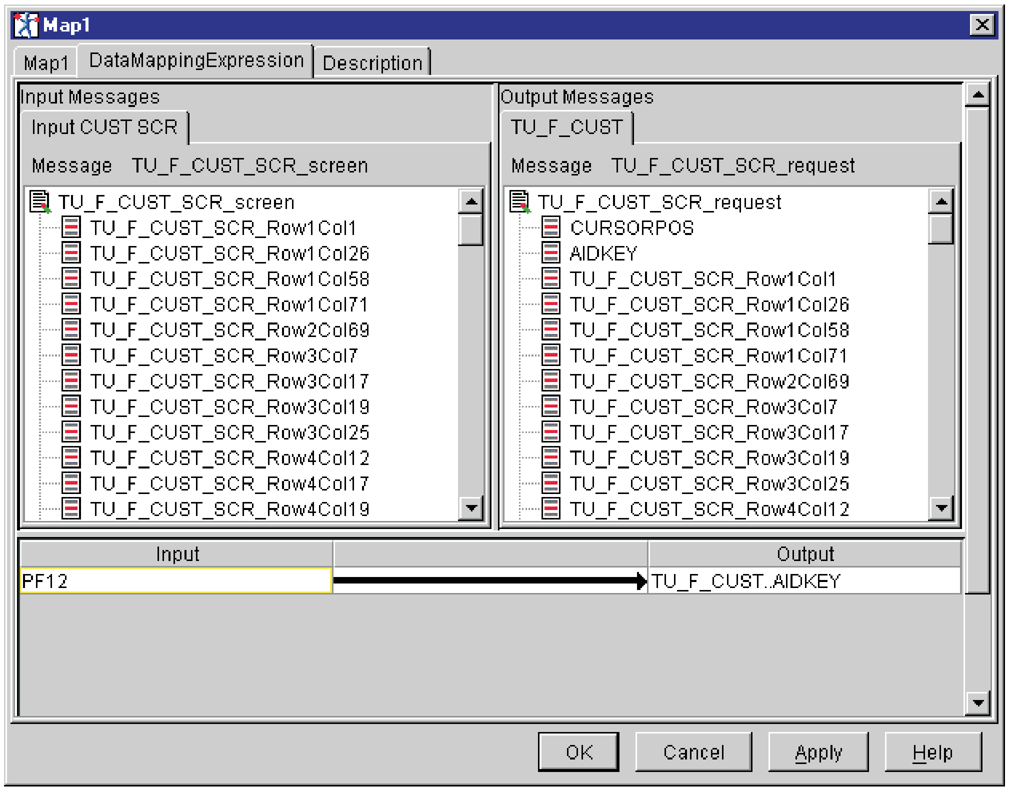
   Right click on the Map1 node (the Map node that appears between the Input CUST SCR and TU_F_CUST nodes) and select **Properties**. Click the DataMappingExpression tab.

   Perform a valid value mapping.

   - Right click on the destination field AIDKEY field in the TU_F_CUST_SCR_request Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.
   - Right click on the LITERAL field and select **Valid Values** from the pop up menu. Use the pull down menu to select PF12 in the Valid Value field. Click **OK**.

Table 67. Mapping fields for Map1 node (TU_F_CUST_SCR_screen message to TU_F_CUST_SCR_request message)

Input Field	Output Field	Description
PF12	AIDKEY	Maps the PF12 key



Input		Output
PF12	→	TU_F_CUST.AIDKEY

Figure 150. Mapping for Map1 node

2) Perform the mapping for the Map2 node as listed in Table 68 on page 195 and shown in Figure 151 on page 195. This mapping occurs in the case where an Unknown screen is received from the TU_F_CUST command. The mapping sets the REPLY_IND to 'E' and creates an error message.

Right click on the Map2 node (the Map node that appears between the TU_F_CUST and Output REPLY nodes) and select **Properties**. Click the DataMappingExpression tab.

a) Perform a literal mapping.

   i. Right click on the destination field REPLY_IND in the TU_F_REPLY Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

   ii. Double click on LITERAL field and rename it to 'E' (quotes must be used around the E string).

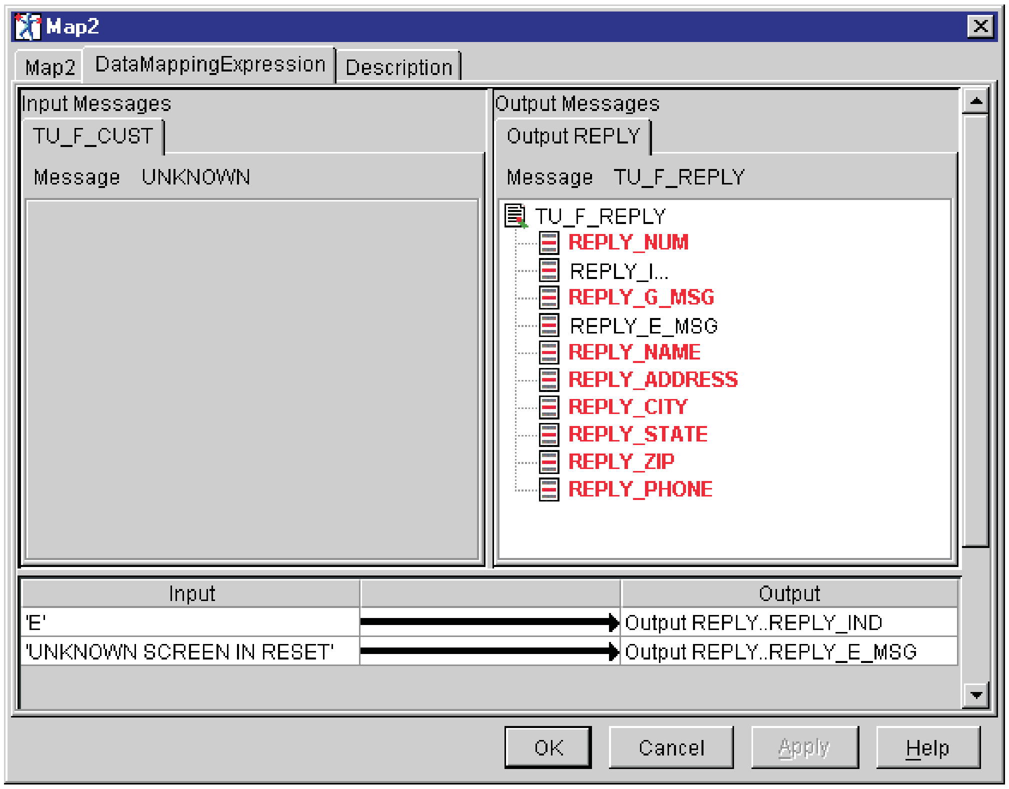
b) Perform a literal mapping.

   i. Right click on the destination field the REPLY_E_MSG field in the TU_F_REPLY Output Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

   ii. Double click on LITERAL field and rename it to 'UNKNOWN SCREEN IN RESET' (quotes must be used around the E string). Click **OK**.

Table 68. Mapping fields for Map2 node (Unknown message to TU_F_REPLY message)

Input Field	Output Field	Description
'E'	REPLY_IND	Sets REPLY_IND to 'E' to indicate an error
'UNKNOWN SCREEN IN RESET'	REPLY_E_MSG	Error message



Input		Output
'E'	→	Output REPLY..REPLY_IND
'UNKNOWN SCREEN IN RESET'	→	Output REPLY..REPLY_E_MSG

Figure 151. Mapping for Map2 node

You just modelled the Reset subflow.

In your model, you have coded the instructions on how the subflow is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

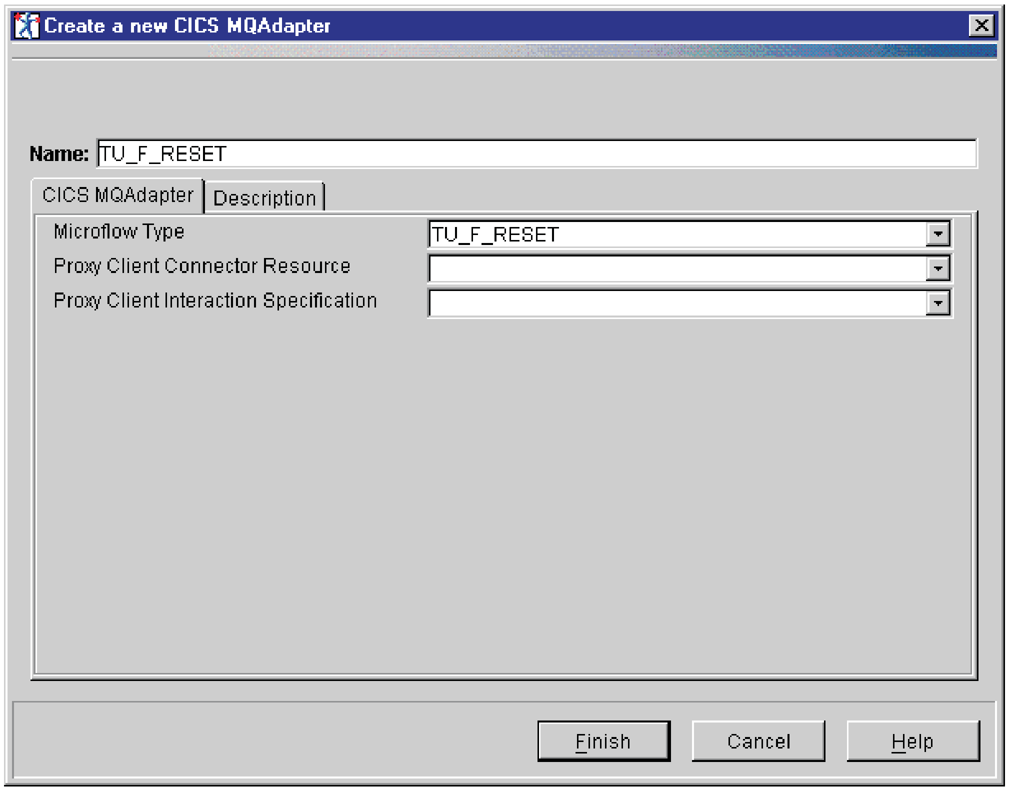
3. **Assign the model of the subflow to a CICS MQAdapter**

   In this step you will associate the Reset subflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual implementation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_RESET for the Name and use the drop down menu to select TU_F_RESET for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.


Create a new CICS MQAdapter

Name: TU_F_RESET

Microflow Type	TU_F_RESET
Proxy Client Connector Resource	
Proxy Client Interaction Specification	

Figure 152. Creating an CICS MQAdapter

You have completed the Reset subflow and setup of this segment of your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

Now you are ready to generate your adapter.

Note: The copybooks were previously generated during the parser subflow (TU_F_PARSER) adapter modeling section (see 2a on page 148).

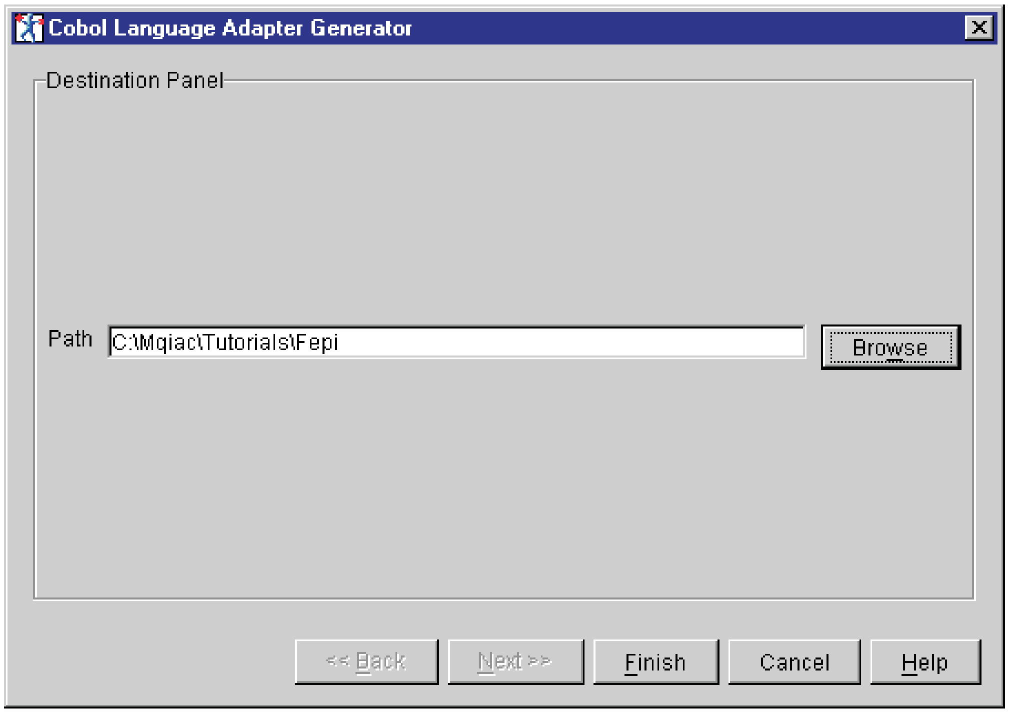
4. **Generate adapter Code**.

   To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

   Note: You must generate the adapter code in the same directory where you generated the copybooks.

   a. Right click on the TU_F_RESET adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination **<mqiac_tutorials>\fepi** in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

      The generated adapter code will be output to the destination path directory.



Path: C:\Mqiac\Tutorials\Fepi

Figure 153. Specifying pathname for adapter code generation output

Create the Navigator microflow

The Navigator microflow is the parent microflow for the FEPI interface. The Navigator microflow calls the FEPI subflows.

This flow represents the Navigator in the run time environment. See Figure 81 on page 115.

Follow these steps to create the Navigator microflow:

1. **Create the component types for use in the Navigator microflow**

   This step is made up of the following tasks:

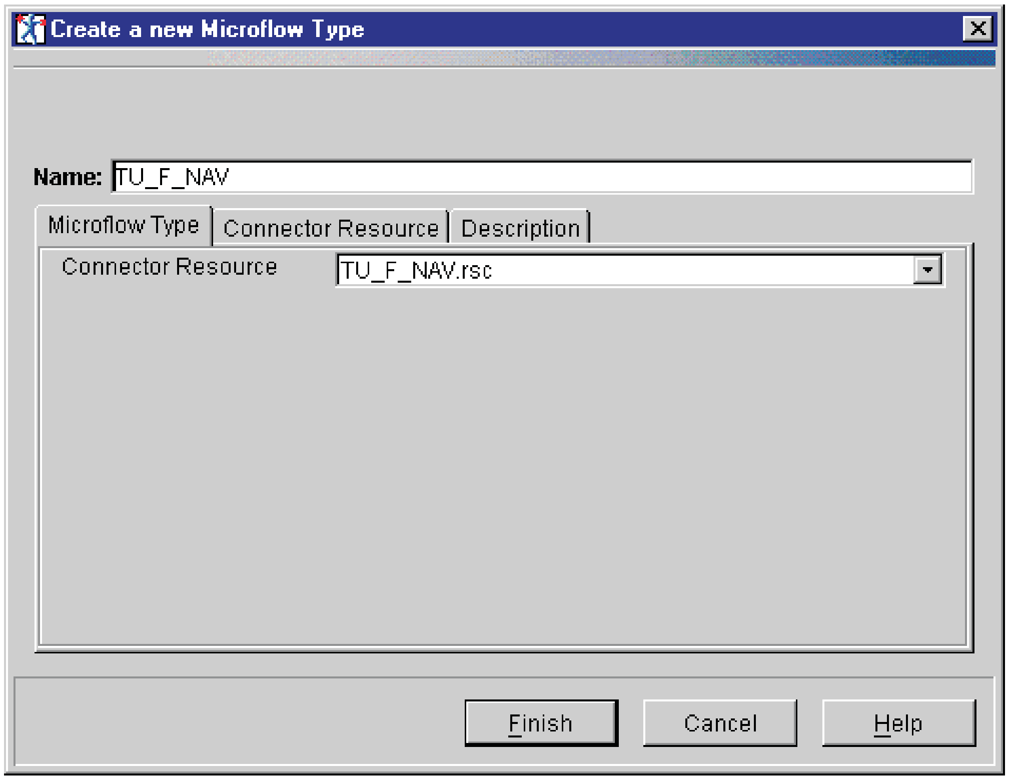
Build an adapter that supports a FEPI interface

- Create a Microflow Type
- Create Decision Types
- Create Data Context Types

a. **Create a microflow type**

Create a microflow that will model the controlling navigation processing for the FEPI interface.

1) Click on the Adapters tab to switch to the Adapters view.
2) Right click on the Microflow Types folder and select **Create > Microflow Type**.
3) Enter TU_F_NAV in the Name field.
4) Use the drop down menu in the Connector Resource field to select TU_F_NAV.rsc as the Connector Resource file and then click **Finish**.



Name: TU_F_NAV

Connector Resource: TU_F_NAV.rsc

Figure 154. Creating a TU_F_NAV Microflow Type

b. **Create Decision Types**

Now it is time to create the Decision types that will be used in the Navigator microflow. You will need to create the following Decision types:

- TU_F_SCR_ID — Tests to see what screen the user is on after the Parser subflow processing completes.
- TU_F_GOOD_SIGNON — Tests for a valid signon.
- TU_F_SIGNOFF — Tests the data in the RAW record to determine whether to release the LU connection.

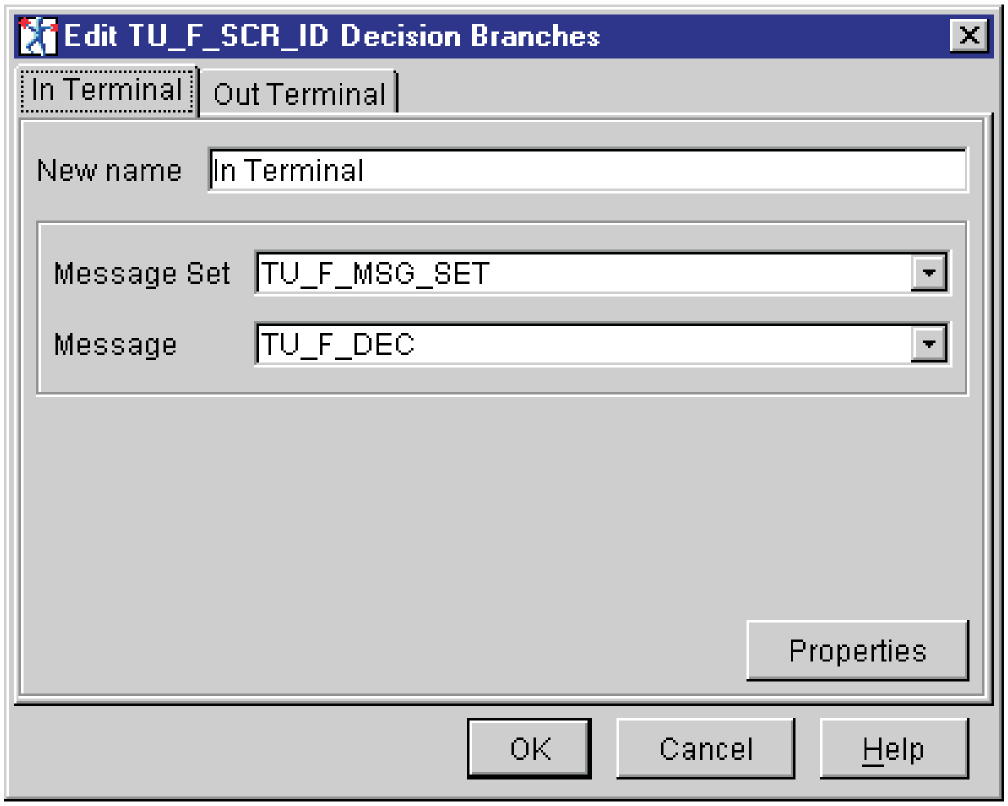
1) *Create the TU_F_SCR_ID Decision type.*

   Right click on the Decision Types folder and select **Create > Decision Type**. Enter *TU_F_SCR_ID* in the Name field and click **Finish**.

a) Associate a message set and message with the In Terminal on the TU_F_SCR_ID Decision type.

   i. Right click on the TU_F_SCR_ID Decision type under the Decision Types folder and select **Decision Branch**. Make sure the In Terminal tab is selected.

   ii. Using the drop down menus, select TU_F_MSG_SET for the Message Sets field and TU_F_DEC for the Messages field. Click **OK**.



New name: In Terminal

Message Set: TU_F_MSG_SET

Message: TU_F_DEC

Figure 155. Editing the In Terminal on the Decision type

b) Create Out Terminals for the SIGNON and CUSTOMER decisions.

   The TU_F_SCR_ID Decision type will determine which of these actions to take based on the screen identity that is returned from the Parser subflow.

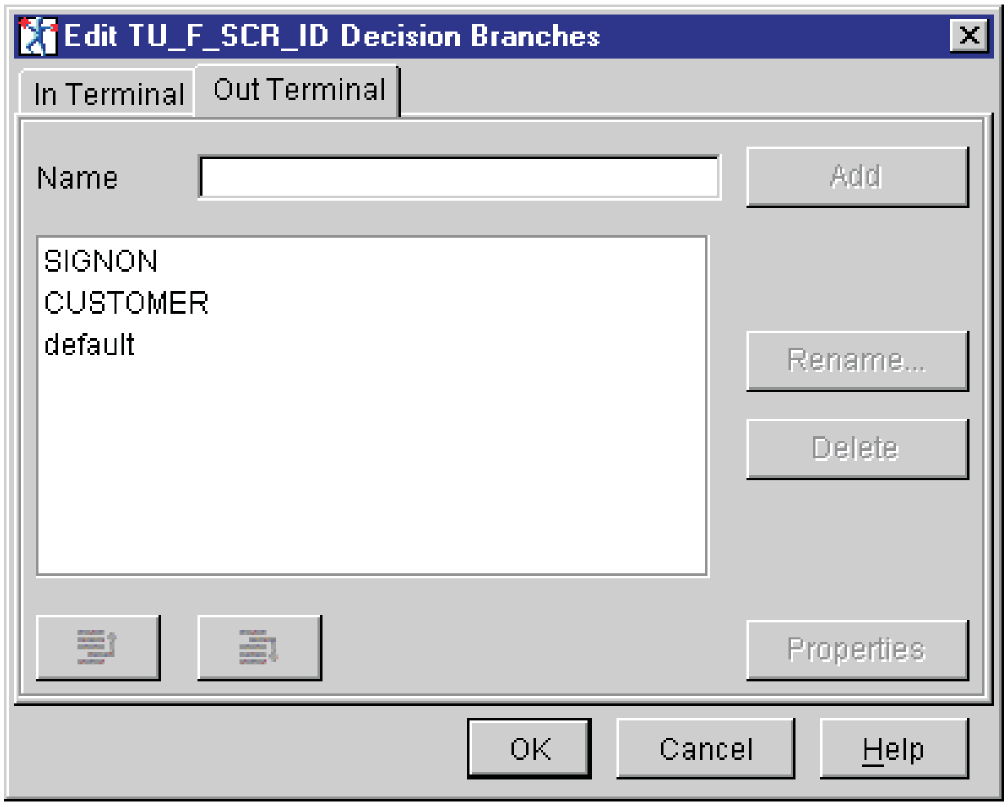
   i. Right click on the TU_F_SCR_ID Decision type under the Decision Types folder and select **Decision Branch**.

      i) Make sure the Out Terminal tab is selected. Click Out Terminal in the terminal list box and click **Rename**.

      ii) Enter *SIGNON* in the New name field.

      iii) Click **Finish**.

   ii. Enter *CUSTOMER* in the Name field and click **Add**. Click **OK**



SIGNON
CUSTOMER
default

Figure 156. Editing the Out Terminal on the Decision type

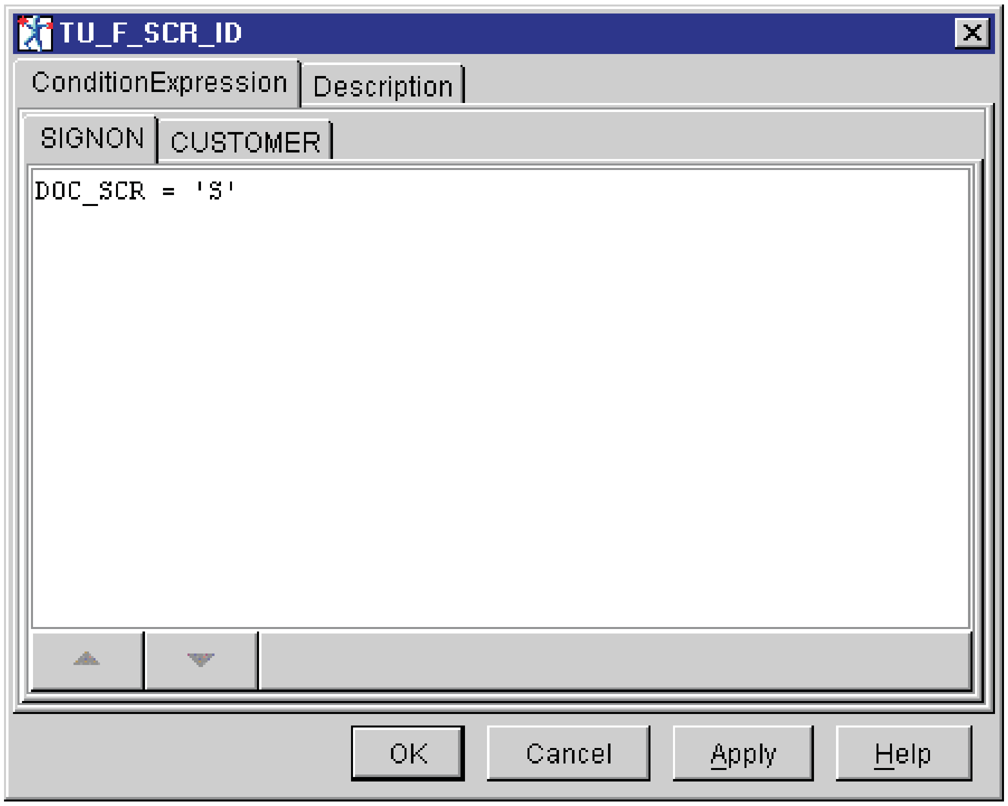
iii. Right click on the TU_F_SCR_ID Decision type and select **Properties** on the pop up menu.

i) Make sure the ConditionExpression tab is selected and the SIGNON tab is selected.

ii) Click in the SIGNON test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_F_DEC message that we associated with the TU_F_SCR_ID Decision type).

iii) Select the *DOC_SCR* field to add this to the ConditionExpression area.

iv) You should add the code shown in Figure 157 on page 201 for the SIGNON terminal test condition. The letter 'S' for the DOC_SCR field is based on the screen indicator codes that are defined for the decision message (TU_F_DEC). The *DOC_SCR* field is returned from the parser subflow indicating the current screen that the user is on.



```
DOC_SCR = 'S'
```

Figure 157. Code for the SIGNON Terminal

iv. In a similar manner, add the test condition code for the CUSTOMER terminal remaining terminals. See Table 69 for the code to add. When finished, click **OK**

Table 69. Code for the Out Terminal actions for the TU_F_SCR_ID Decision type

Terminal	Code	Description
SIGNON	DOC_SCR = ’S’	S - SIGNON screen identified by Parser subflow
CUSTOMER	DOC_SCR = ’C’	C - CUSTOMER screen identified by Parser subflow

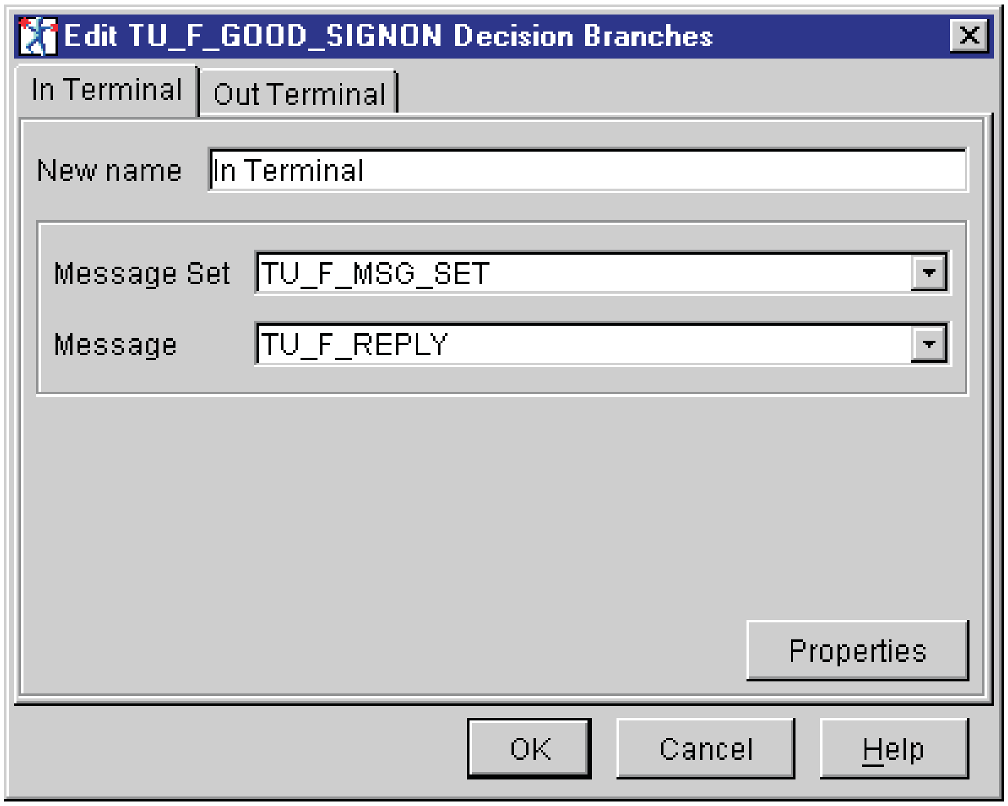
2) *Create the TU_F_GOOD_SIGNON Decision type.*

Right click on the Decision Types folder and select **Create > Decision Type**. Enter *TU_F_GOOD_SIGNON* in the Name field and click **Finish**.

Associate a message set and message with the In Terminal on the TU_F_GOOD_SIGNON Decision type.

a) Right click on the TU_F_GOOD_SIGNON Decision type under the Decision Types folder and select **Decision Branch**.

b) Make sure the In Terminal tab is selected. Using the drop down menus, select TU_F_MSG_SET for the Message Sets field and TU_F_REPLY for the Messages field. Click **OK**.



New name: In Terminal

Message Set: TU_F_MSG_SET

Message: TU_F_REPLY

Figure 158. Editing the In Terminal on the Decision type

c) Create Out Terminals for the GOOD_SIGNON decision.

The TU_F_GOOD_SIGNON Decision type will determine which action to take based on whether a valid signon occurs.

i. Right click on the TU_F_GOOD_SIGNON Decision type under the Decision Types folder and select **Decision Branch**.

ii. Make sure the Out Terminal tab is selected. Click Out Terminal in the terminal list box and click **Rename**.

iii. Enter *GOOD_SIGNON* in the New name field and click **Finish**.

iv. Click **OK**.

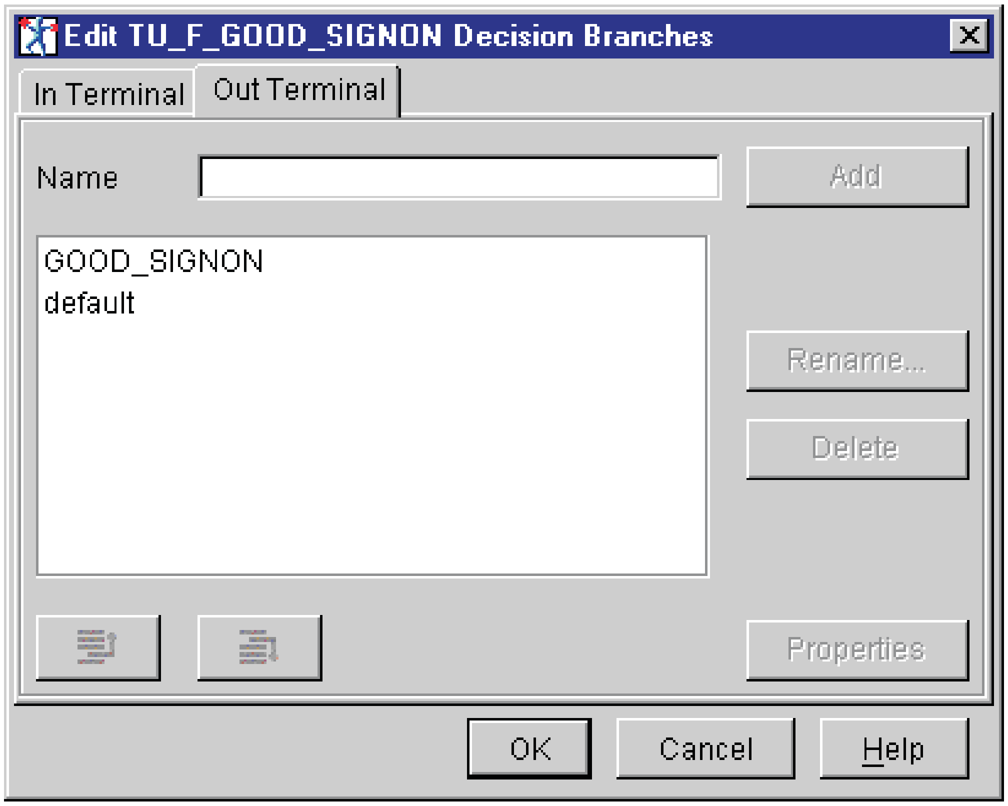


Figure 159. Editing the Out Terminal on the TU_F_GOOD_SIGNON Decision type

v. Right click on the TU_F_GOOD_SIGNON Decision type and select **Properties** on the pop up menu.

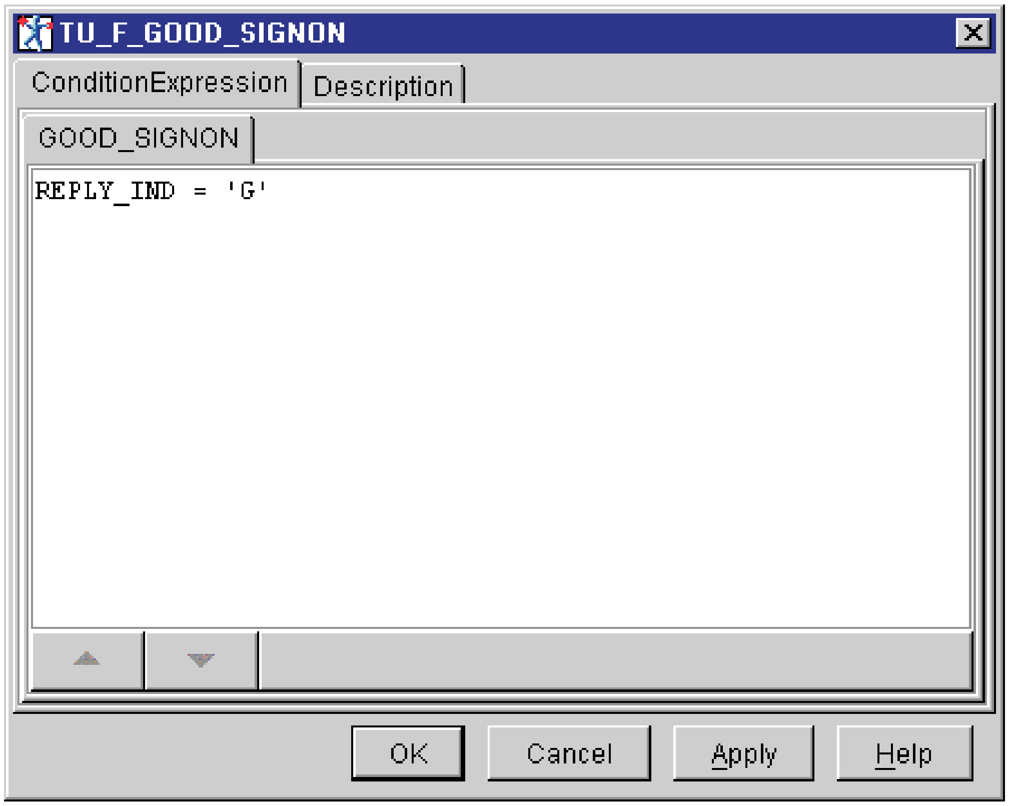
i) Make sure the ConditionExpression tab is selected.

ii) Click in the GOOD_SIGNON test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_F_REPLY message that we associated with the TU_F_GOOD_SIGNON Decision type).

iii) Select the *REPLY_IND* field to add this to the ConditionExpression area.

iv) You should add the code shown in Figure 160 on page 204 for the GOOD_SIGNON terminal test condition. The letter 'G' for the REPLY_IND field indicates that a valid signon has occurred.

v) Click **OK**.



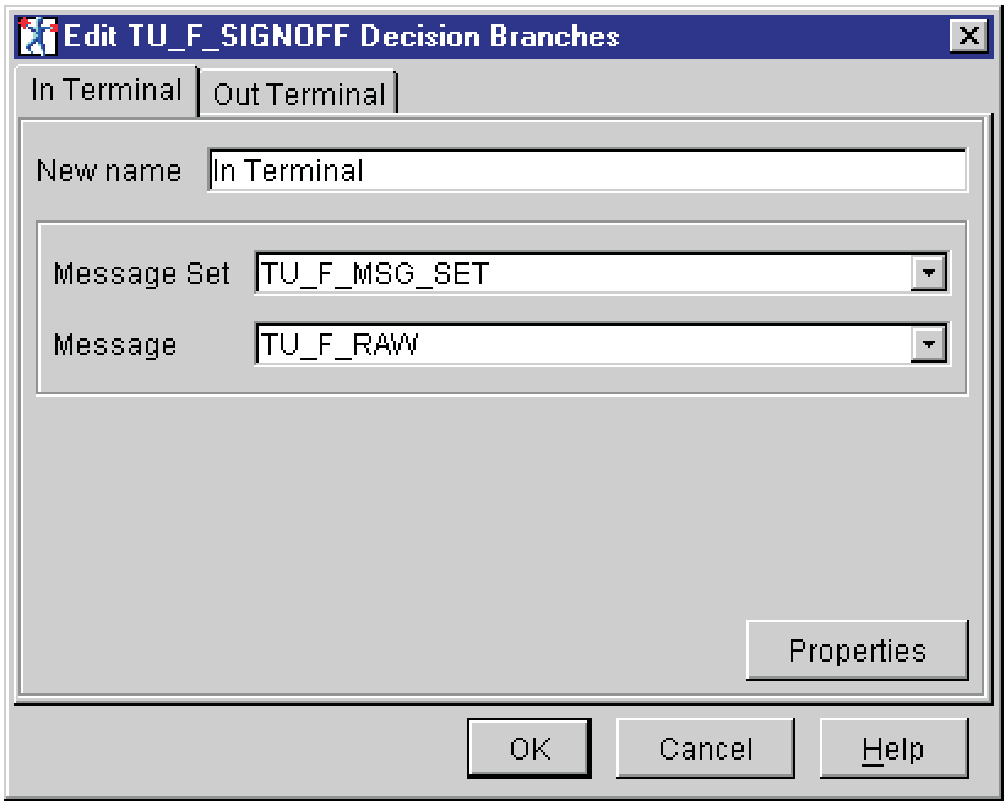
```
REPLY_IND = 'G'
```

Figure 160. Code for the GOOD_SIGNON Terminal

3) *Create the TU_F_SIGNOFF Decision type.*

   The TU_F_SIGNOFF Decision type will determine whether to release the LU connection.

   a) Right click on the Decision Types folder and select **Create > Decision Type**.
   b) Enter *TU_F_SIGNOFF* in the Name field and click **Finish**.
   c) Associate a message set and message with the In Terminal on the TU_F_SIGNOFF Decision type. Right click on the TU_F_SIGNOFF Decision type under the Decision Types folder and select **Decision Branch**.
   d) Make sure the In Terminal tab is selected.
   e) Using the drop down menus, select TU_F_MSG_SET for the Message Sets field and TU_F_RAW for the Messages field. Click **OK**.



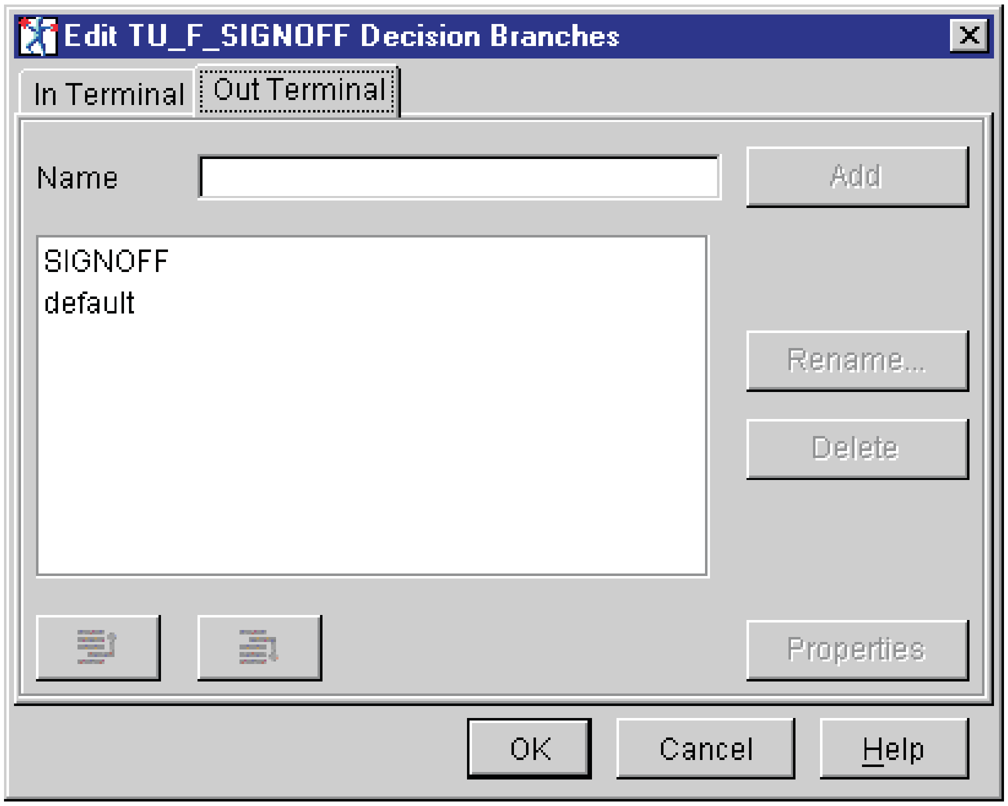
New name: In Terminal

Message Set: TU_F_MSG_SET

Message: TU_F_RAW

Figure 161. Editing the In Terminal on the TU_F_SIGNOFF Decision type

f) Create an Out Terminal for the SIGNOFF decision.
   i. Right click on the TU_F_SIGNOFF Decision type under the Decision Types folder and select **Decision Branch**.
   ii. Make sure the Out Terminal tab is selected. Click Out Terminal in the terminal list box and click **Rename**.
   iii. Enter *SIGNOFF* in the New name field and click **Finish**.
   iv. Click **OK**.



SIGNOFF
default

Figure 162. Editing the Out Terminal on the TU_F_SIGNOFF Decision type

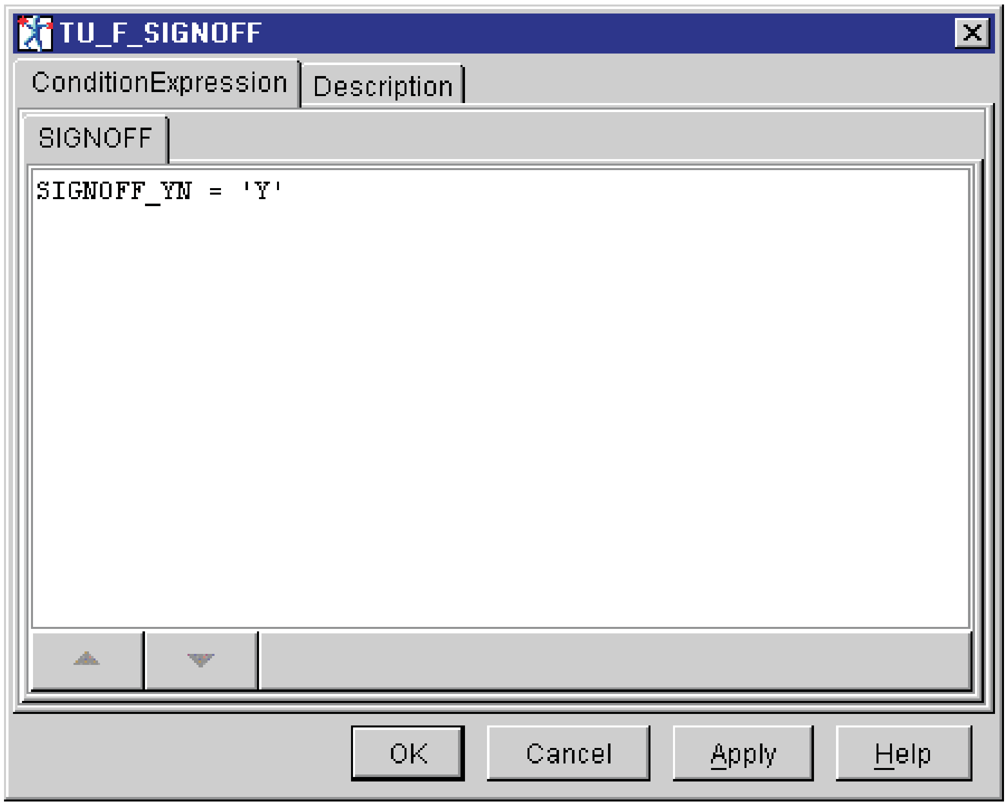
g) Set the Decision type properties.

i. Right click on the TU_F_SIGNOFF Decision type and select **Properties** on the pop up menu.

ii. Make sure the ConditionExpression tab is selected. Click in the SIGNOFF test condition input area and press CTRL-SHIFT to display a list of available message fields (these fields are from the TU_F_RAW message that we associated with the TU_F_SIGNOFF Decision type).

iii. Select the *SIGNOFF_YN* field to add this to the ConditionExpression area.

iv. You should add the code shown in Figure 163 on page 207 for the SIGNOFF terminal test condition. The letter 'Y' for the SIGNOFF_YN field indicates that signoff should occur.



```
SIGNOFF_YN = 'Y'
```

Figure 163. Code for the SIGNOFF Terminal

v. Click **OK**.

c. **Create Data Context Types.**

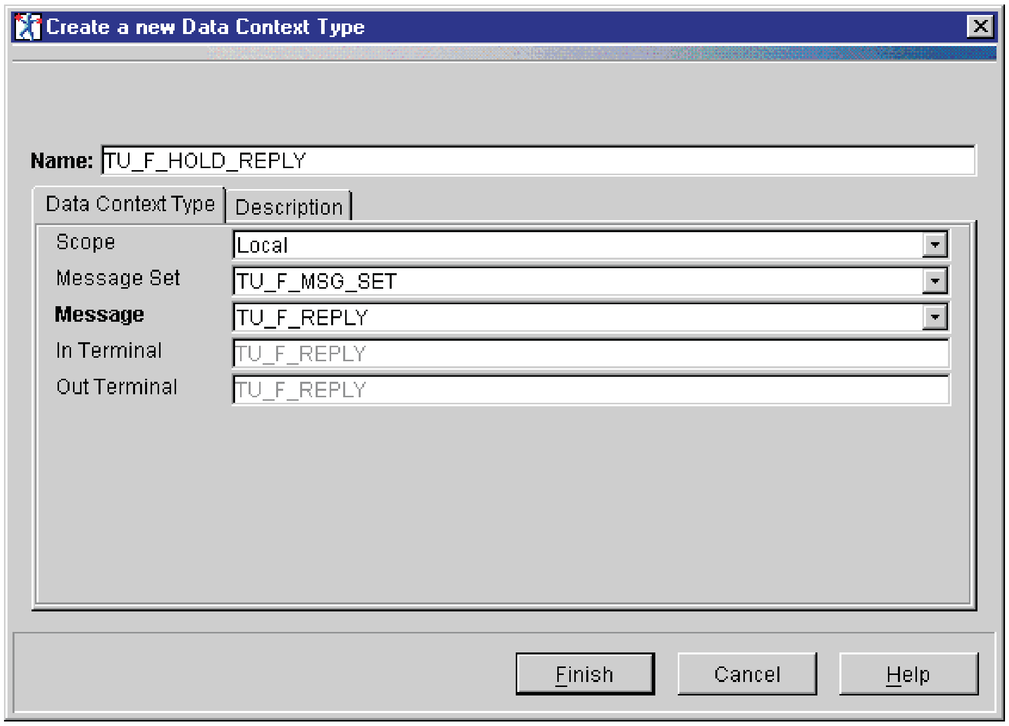
You will create the following two Data Context types to use in the Navigator microflow:

- TU_F_HOLD_REPLY — Holds the data in the REPLY record for processing later in the flow.
- SYS_FEPI_OVERRIDES — Holds the override values for FEPI processing.

1) Create the TU_F_HOLD_REPLY Data Context type.

a) Right click on the Data Context Types folder and select **Create > Data Context Type**. Enter *TU_F_HOLD_REPLY* in the Name field.

b) Using the drop down menus, set the field property values shown in Figure 164 on page 208.



Name: TU_F_HOLD_REPLY

Scope: Local
Message Set: TU_F_MSG_SET
Message: TU_F_REPLY
In Terminal: TU_F_REPLY
Out Terminal: TU_F_REPLY

Figure 164. Creating a TU_D_HOLD_REPLY Data Context type

Click **Finish** to apply the property values.

2) Right click on the Data Context types folder and select **Add to Workspace > Data Context Type**. Select the SYS_FEPI_OVERRIDES Data Context type and click **Finish**.

2. **Model the Navigator microflow**

   In this step you will perform a set of tasks to define and model the Navigator flow's functionality.

   This step is made of the following tasks:

   - Adding microflow nodes
   - Connecting the nodes
   - Defining the mappings

   a. **Add nodes to the Navigator microflow**

      In this task, you will drag the component types that you created in step 1 on page 197, onto the Microflow Definition pane. When you drag a component type onto the Microflow Definition pane, it is instantiated and referred to as a *microflow node*. A single component type can be used to create one or more microflow nodes (instances) as part of the same microflow.

      1) **Add the Input Terminal node**

         The Input Terminal can make a connection to any terminal that resides within the microflow.

         a) Drag the node on to the Microflow Definition pane

            In the Microflow Types folder, select the TU_F_NAV microflow you created.

Note: To model your adapter in the workspace (Microflow Definition pane), you must make sure the microflow is selected in the Microflow Types folder.

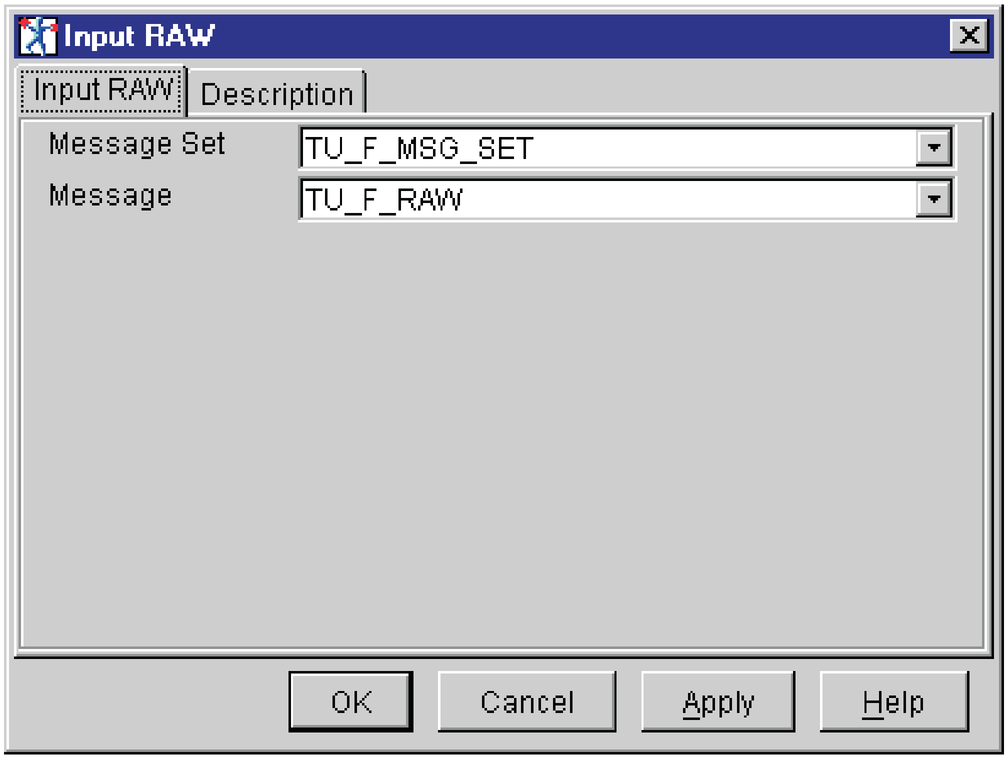
Drag an Input Terminal type from the Adapter Tree View to the workspace (Left click and hold on the Input Terminal to drag it to the workspace).

b) Rename the node

Right click on the Input Terminal and select **Rename**. Rename the Input Terminal node to Input RAW and click **Finish**.

c) Set the properties for the node

Right click on the Input Terminal and select **Properties**. From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_RAW in the Messages field. Click **OK**.



Message Set: TU_F_MSG_SET
Message: TU_F_RAW

Figure 165. Configuring the Input RAW Input Terminal node properties

2) **Add the Parser node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_PARSER Microflow type from the Adapter Tree View to the workspace. Place the node to the right on the Input RAW node.

b) Rename the node

Right click on the TU_F_PARSER1 Microflow node and select **Rename**.

Modify TU_F_PARSER1 in the New name field to the name TU_F_PARSER and click **Finish**.

3) **Add the Decision node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_SCR_ID Decision type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_PARSER node.

b) Rename the node

Right click on the TU_F_SCR_ID1 Decision node and select **Rename**

Modify TU_F_SCR_ID1 in the New name field to the name TU_F_SCR_ID and click **Finish**.

4) **Add a Reset node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_RESET Microflow type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_SCR_ID node.

b) Rename the node

Right click on the TU_F_RESET1 Microflow node and select **Rename**.

Modify TU_F_RESET1 in the New name field to the name TU_F_RESET and click **Finish**.

5) **Add the Output terminal node**

a) Drag the node on to the Microflow Definition pane

Drag an Output Terminal type from the Adapter Tree View to the workspace and place the node to the right of the TU_F_RESET node.

b) Rename the node

Rename the Output Terminal node to Output REPLY.

c) Flip the node

Right click on the Output REPLY node and select **Flip node**.

d) Set the properties for the node

Right click on the Output REPLY and select **Properties**.

From the drop down menus, select TU_F_MSG_SET in the Message Sets field and select TU_F_REPLY in the Messages field. Click **OK**.

6) **Add the Signon node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_SIGNON Microflow type from the Adapter Tree View to the workspace. Place the node above and to the right of the TU_F_SCR_ID node.

b) Rename the node

Right click on the TU_F_SIGNON1 Microflow node and select **Rename**.

Modify TU_F_SIGNON1 in the New name field to the name TU_F_SIGNON and click **Finish**.

7) **Add the Good Signon Decision node**

a) Drag the node on to the Microflow Definition pane

Drag a TU_F_GOOD_SIGNON Decision type from the Adapter Tree View to the workspace. Place the node above and to the right of the TU_F_SIGNON node.

b) Rename the node

Right click on the TU_F_GOOD_SIGNON1 Decision node and select **Rename**.

Modify TU_F_GOOD_SIGNON1 in the New name field to the name TU_F_GOOD_SIGNON and click **Finish**.

8) **Add the Inquiry Microflow node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_INQ Microflow type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_GOOD_SIGNON node.
   b) Rename the node

      Right click on the TU_F_INQ1 Microflow node and select **Rename**.

      Modify TU_F_INQ1 in the New name field to the name TU_F_INQ and click **Finish**.
9) **Add the Signoff Decision node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_SIGNOFF Decision type from the Adapter Tree View to the workspace. Place the node to the right of the TU_F_INQ node.
   b) Rename the node

      Right click on the TU_F_SIGNOFF1 Decision node and select **Rename**.

      Modify TU_F_SIGNOFF1 in the New name field to the name TU_F_SIGNOFF and click **Finish**.
10) **Add the Signoff Microflow node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_SGNOFF Microflow type from the Adapter Tree View to the workspace. Place the node to the right on the TU_F_SIGNOFF node and above and to the left of the Output REPLY node.
   b) Rename the node

      Right click on the TU_F_SGNOFF1 Microflow node and select **Rename**.

      Modify TU_F_SGNOFF1 in the New name field to the name TU_F_SGNOFF and click **Finish**.
11) **Add the Hold Reply Data context node**
   a) Drag the node on to the Microflow Definition pane

      Drag a TU_F_HOLD_REPLY Data Context type from the Adapter Tree View to the workspace. Place the node above and between the TU_F_INQ and TU_F_SIGNOFF nodes.
   b) Rename the node

      Right click on the TU_F_HOLD_REPLY1 Data Context node and select **Rename**.

      Modify TU_F_HOLD_REPLY1 in the New name field to the name TU_F_HOLD_REPLY and click **Finish**.
12) **Add the FEPI Overrides Data context node**
   a) Drag the node on to the Microflow Definition pane

      Drag a SYS_FEPI_OVERRIDES Data Context type from the Adapter Tree View to the workspace. Place the node above the TU_F_SGNOFF Microflow node.

b) Rename the node

Right click on the SYS_FEPI_OVERRIDES1 Data Context node and select **Rename**.

Modify SYS_FEPI_OVERRIDES1 in the New name field to the name SYS_FEPI_OVERRIDES and click **Finish**.

13) Save your workspace by selecting **File > Save Workspace** from the menubar.

b. **Connect the microflow nodes**

In this task you will connect the microflow nodes that are on the Microflow Definition pane. You will do this by creating *connections*. A connection is a wire that connects an output terminal of one microflow node to the input terminal of another. There are two types of connections (control connection and data connection). For a detailed description of the different types of connections, see the section on composing microflows in the *MQSeries Integrator Agent for CICS Transaction Server Using the Control Center* book.

1) Right click on the Input RAW Input Terminal node and select **Connect > Out**. Move the connection line to the TU_F_PARSER Command node and left click. This adds a control connection. Refer to Figure 166 on page 215.
2) Add a control connection from the TU_F_PARSER Microflow node to the TU_F_SCR_ID node.
3) Add a control connection from the first out terminal (SIGNON) on the TU_F_SCR_ID Decision node to the TU_F_SIGNON node.
4) Add a control connection from the TU_F_SIGNON Microflow node to the TU_F_GOOD_SIGNON Decision node.
5) Add a Map node (Map1) between the TU_F_GOOD_SIGNON Decision node and the TU_F_INQ Microflow node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the node before releasing the mouse button.
6) Add a control connection from the first out terminal (GOOD_SIGNON) on the TU_F_GOOD_SIGNON Decision node to the Map1 node and from the Map1 node to the TU_F_INQ Microflow node.
7) Add a control connection from the TU_F_INQ Microflow node to the TU_F_SIGNOFF Decision node. This adds a control connection and a map (Map2 node) between the two nodes.
8) Add a Map node (Map3) between the TU_F_SIGNOFF Decision node and the TU_F_SGNOFF Microflow node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before releasing the mouse button.
9) Add a control connection from the first output terminal (SIGNOFF) on the TU_F_SIGNOFF Decision node to the Map3 node and from the Map3 node to the TU_F_SGNOFF Microflow node.
10) Add a control connection from the TU_F_SGNOFF Microflow node to the Output REPLY node.
11) Add a Map node (Map4) between the TU_F_SIGNOFF Decision node and the Output REPLY node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before releasing the mouse button.

12) Add a control connection from the second output terminal (default) on the TU_F_SIGNOFF Decision node to the Map4 node and from the Map4 node to the Output REPLY node.
13) Add a Map node (Map5) between the TU_F_GOOD_SIGNON Decision node and the Output REPLY node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before releasing the mouse button.
14) Add a control connection from the second output terminal (default) on the TU_F_GOOD_SIGNON Decision node to the Map5 node and from the Map5 node to the Output REPLY node.
15) Add a control connection from the second output terminal (CUSTOMER) on the TU_F_SCR_ID Decision node to the TU_F_RESET Microflow node.
16) Add a control connection from the first out terminal (TU_F_CUST_SCR) on the TU_F_RESET Microflow node to the Map1 node.
17) Add a control connection from the second out terminal (TU_F_REPLY) on the TU_F_RESET Microflow node to the Output REPLY node.
18) Add a Map node (Map6) between the TU_F_SCR_ID node and the Output REPLY node. To create a Map node, drag a Map type from the Adapters Tree View (left panel) to the Microflow Definition panel (right panel) and position the cursor between the nodes before releasing the mouse button.
19) Add a control connection from the third out terminal (default) on the TU_F_SCR_ID Decision node to the Map6 node and from the Map6 node to the Output REPLY node.
20) Add a data connection. Right click on the Input RAW node and select **Connect > Out**. Move the connection line to the Map1 node and select **In > DataConnectionType** from the pop up menu.
21) Add a data connection. Right click on the Input RAW node and select **Connect > Out**. Move the connection line to the Map2 node and select **In > DataConnectionType** from the pop up menu.
22) Add a data connection from the Map2 node to the Input Terminal of TU_F_HOLD_REPLY Data Context node.
23) Add a data connection from the Output Terminal of TU_F_HOLD_REPLY Data Context node to the Map3 node.
24) Add a data connection from the Map3 node to the Input Terminal of SYS_FEPI_OVERRIDES Data Context node.
25) Add a data connection from the Output Terminal of TU_F_HOLD_REPLY Data Context node to the Map4 node.
26) Save your workspace by selecting **File > Save Workspace** from the menubar.

Table 70. Summary of connections used in the TU_F_NAV microflow

From	To	Type of Connection
Input RAW	TU_F_PARSER	Control Connection
TU_F_PARSER	TU_F_SCR_ID	Control Connection
TU_F_SCR_ID (first out terminal — SIGNON)	TU_F_SIGNON	Control Connection
TU_F_SIGNON	TU_F_GOOD_SIGNON	Control Connection

Table 70. Summary of connections used in the TU_F_NAV microflow (continued)

From	To	Type of Connection
TU_F_GOOD_SIGNON (First terminal)	Map1	Control Connection
Map1	TU_F_INQ	Control Connection
TU_F_INQ	Map2	Control Connection
Map2	TU_F_SIGNOFF	Control Connection
TU_F_SIGNOFF (Decision) (first out terminal — SIGNOFF)	Map3	Control Connection
Map3	TU_F_SGNOFF	Control Connection
TU_F_SGNOFF	Output REPLY	Control Connection
TU_F_SIGNOFF (Decision) (second out terminal — default)	Map4	Control Connection
Map4	Output REPLY	Control Connection
TU_F_GOOD_SIGNON (second out terminal — default)	Map5	Control Connection
Map5	Output REPLY	Control Connection
TU_F_SCR_ID (second out terminal — CUSTOMER)	TU_F_RESET	Control Connection
TU_F_RESET (first out terminal — Output CUST SCR)	Map1	Control Connection
TU_F_RESET (second out terminal — Output REPLY)	Output REPLY	Control Connection
TU_F_SCR_ID (third out terminal — default)	Map6	Control Connection
Map6	Output REPLY	Control Connection
Input RAW	Map1	Data Connection
Input RAW	Map2	Data Connection
Map2	TU_F_HOLD_REPLY	Data Connection
TU_F_HOLD_REPLY	Map3	Data Connection
TU_F_HOLD_REPLY	Map4	Data Connection
Map3	SYS_FEPI_OVERRIDES	Data Connection

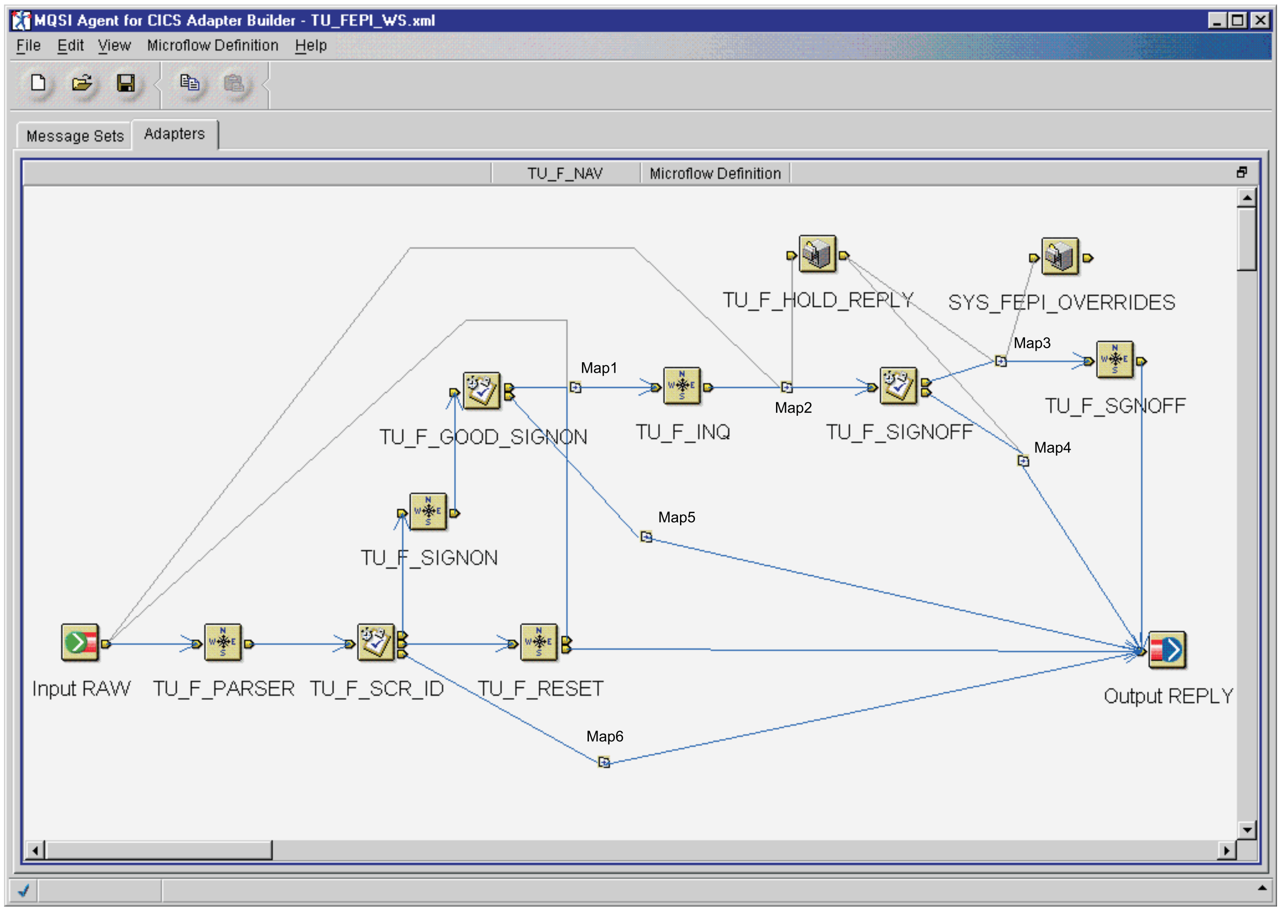


Figure 166. TU_F_NAV

c. **Map the Navigator microflow**

The act of *mapping* refers to the modeling of data transformation via a Map node, between an output terminal on one node and an input terminal on another node.

Data transformation can include a variety of functions:

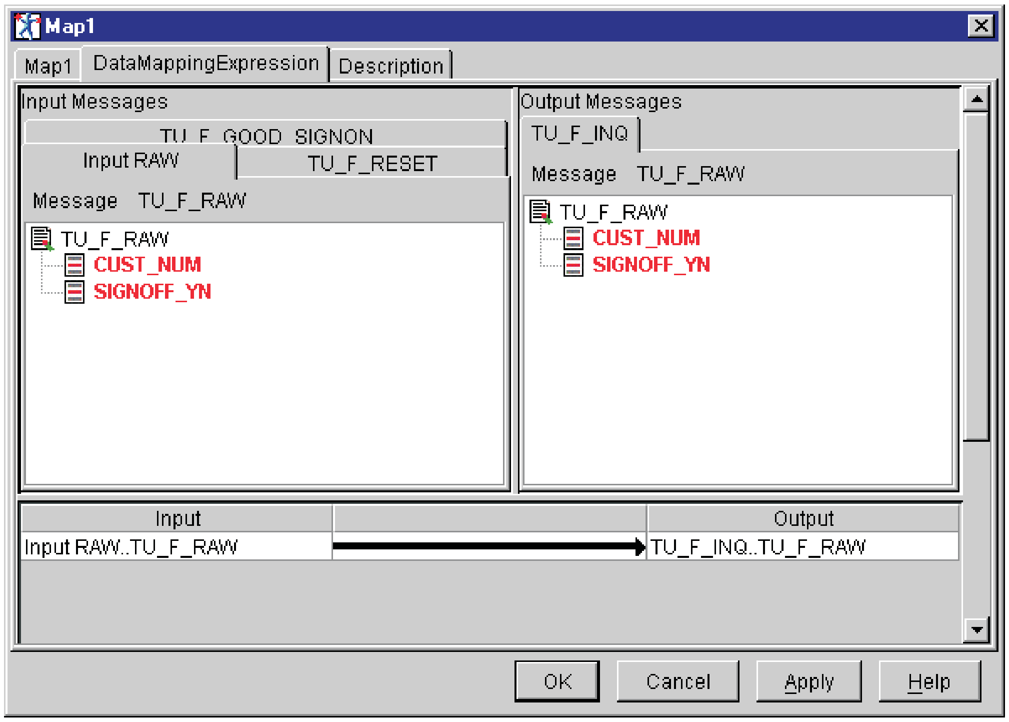
- Associating a field in one message with a field in another message.
- String mapping such as specifying pad characters.
- Date mapping, such as converting a date in one format to a date in another format.
- Putting literal data into a message.
- Adding custom code to perform other data transformation functions.

1) Perform the mapping for the Map1 node as shown in Figure 167 on page 216.

This map passes along the RAW record data for processing in the TC_F_INQ subflow.

Right click on the Map1 node (the Map node that appears between the TU_F_GOOD_SIGNON and TU_F_INQ nodes) and select **Properties**. Click the DataMappingExpression tab.

Left click on the TU_F_RAW message under the Input RAW message (view input message on left of panel) and drag the mouse cursor to the TU_F_RAW message under the TU_F_RAW output message (view output message on right of panel). This will create a mapping between the two messages. Click **OK**.



Input		Output
Input RAW..TU_F_RAW		TU_F_INQ..TU_F_RAW

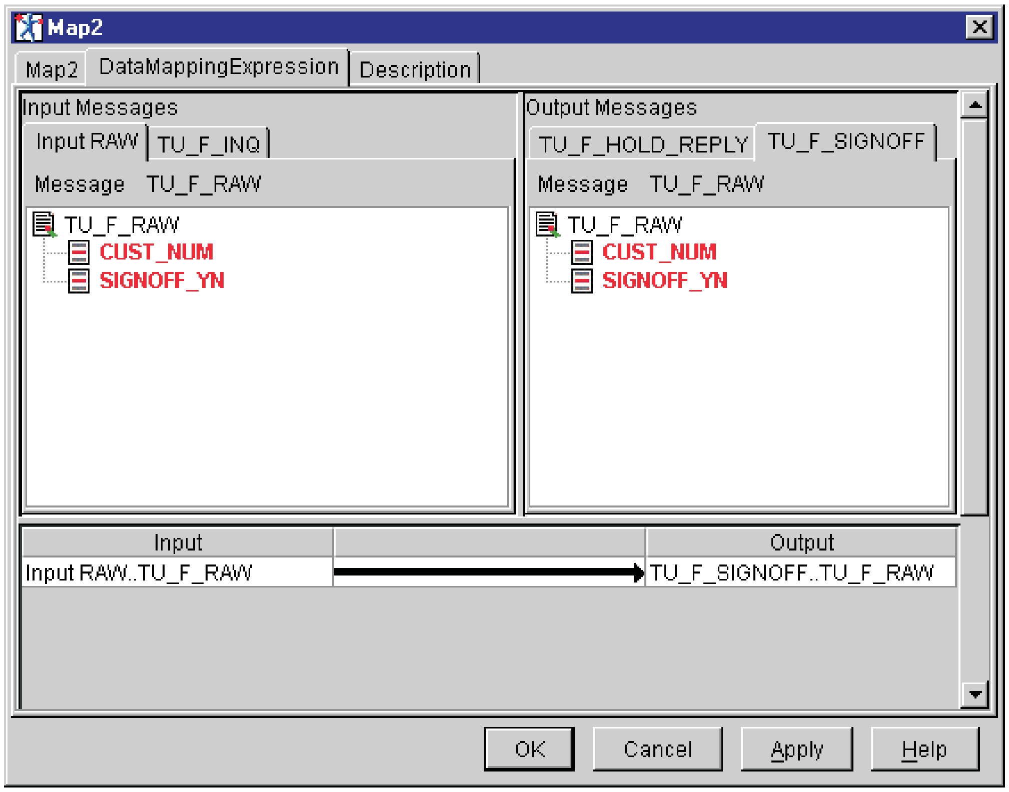
Figure 167. Mapping for Map1 node

2) Perform the mapping for the Map2 node as shown in Figure 168 on page 217.

This mapping passes along the RAW record for testing in the TU_F_SIGNOFF decision and puts the REPLY record in the TU_F_HOLD_REPLY data context.

Right click on the Map2 node (the Map node that appears between the TU_F_INQ and TU_F_SIGNOFF nodes) and select **Properties**. Click the DataMappingExpression tab.

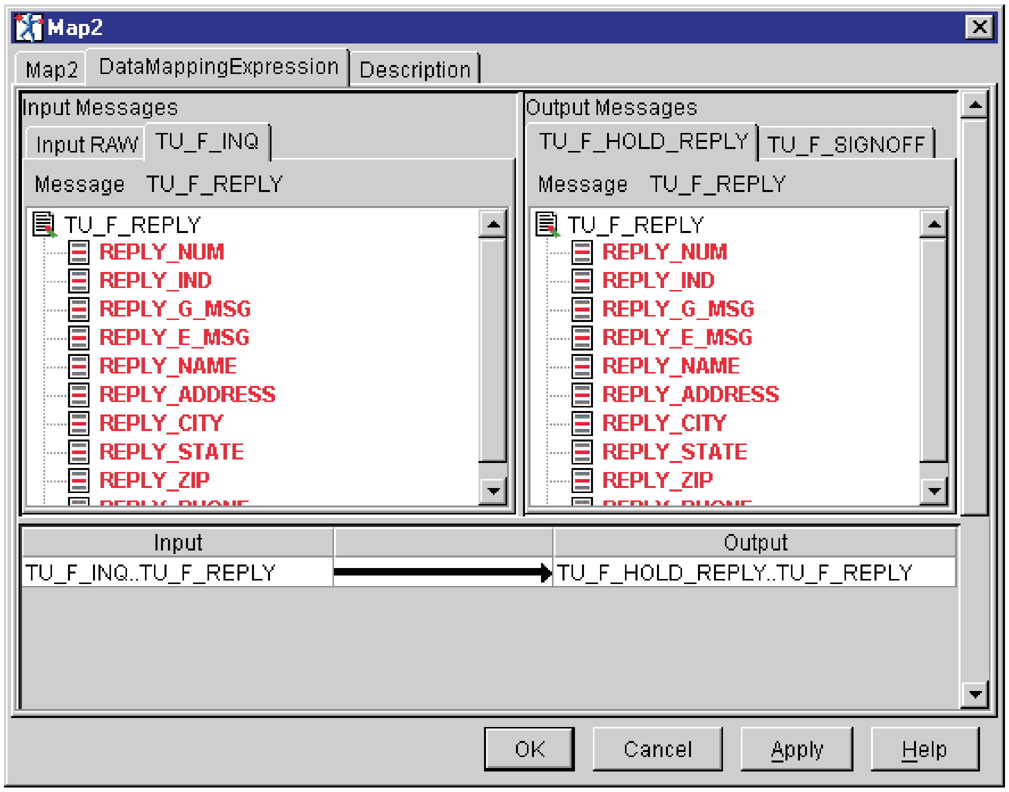
Select the Input RAW message tab (Input Messages) and TU_F_SIGNOFF message tab (Output Messages). Left click on the TU_F_RAW message under the Input RAW message tab and drag the mouse cursor to the TU_F_RAW message under the TU_F_SIGNOFF message tab. This will create a mapping between the two messages (see Figure 168 on page 217).



Input		Output
Input RAW..TU_F_RAW	→	TU_F_SIGNOFF..TU_F_RAW

Figure 168. Mapping for Map2 node

Select the TU_F_INQ message tab (Input Messages) and TU_F_HOLD_REPLY message tab (Output Messages). Left click on the TU_F_REPLY message under the TU_F_INQ message tab and drag the mouse cursor to the TU_F_REPLY message under the TU_F_HOLD_REPLY message tab. This will create a mapping between the two messages (see Figure 169 on page 218). Click **OK**.



Message TU_F_REPLY

- TU_F_REPLY
  - REPLY_NUM
  - REPLY_IND
  - REPLY_G_MSG
  - REPLY_E_MSG
  - REPLY_NAME
  - REPLY_ADDRESS
  - REPLY_CITY
  - REPLY_STATE
  - REPLY_ZIP

Message TU_F_REPLY

- TU_F_REPLY
  - REPLY_NUM
  - REPLY_IND
  - REPLY_G_MSG
  - REPLY_E_MSG
  - REPLY_NAME
  - REPLY_ADDRESS
  - REPLY_CITY
  - REPLY_STATE
  - REPLY_ZIP

Input		Output
TU_F_INQ..TU_F_REPLY	→	TU_F_HOLD_REPLY..TU_F_REPLY

Figure 169. Mapping for Map2 node

3) Perform the mapping for the Map3 node as shown in Figure 170 on page 219 .

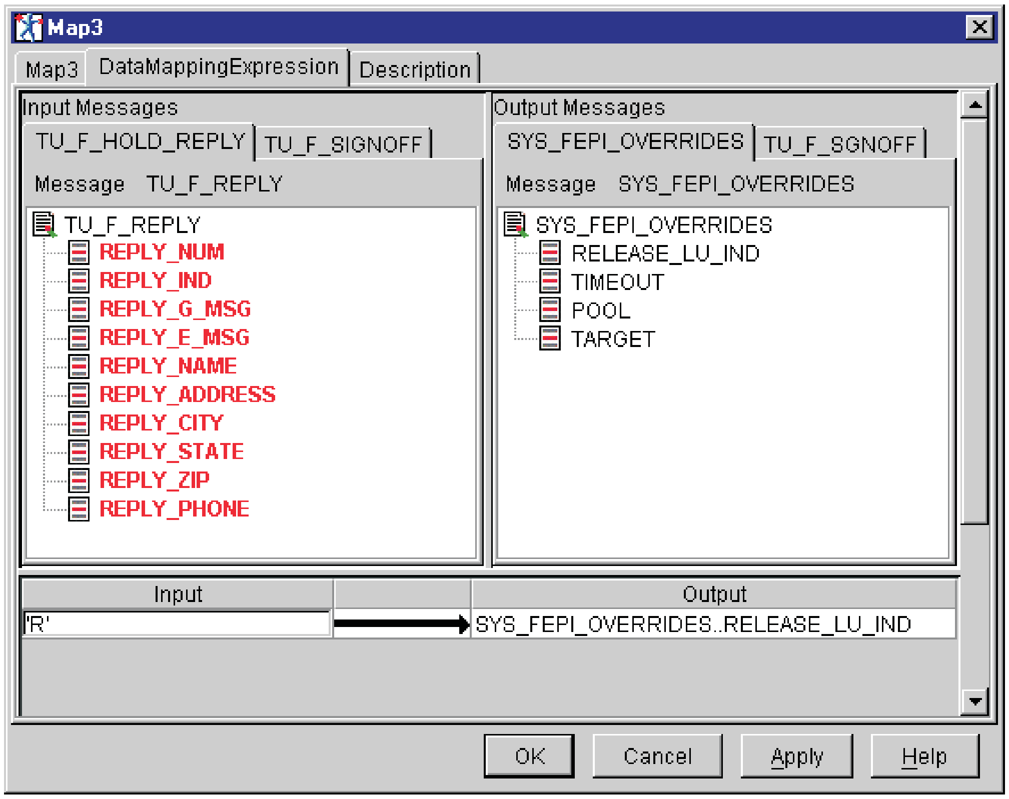
   This map sets the RELEASE_LU_IND field to ’R’ (release LU connection for signoff) and passes along the REPLY record.

   a) Perform a literal mapping. Under the SYS_FEPI_OVERRIDE Output Messages tab, right click on the RELEASE_LU_IND field and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ’R’ (quotes must be used). Refer to Figure 170 on page 219.

   b) Click **Apply**.

   c) Select the TU_F_HOLD_REPLY tab under the Input Messages and the TU_F_SGNOFF tab under the Output Messages. Map the TU_F_REPLY message to the TU_F_REPLY message

   d) Perform a literal mapping. Under the TU_F_SGNOFF Output Messages tab, right click on the REPLY_IND field and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ’L’ (quotes must be used). This indicates that the signoff path will be followed. Refer to Figure 171 on page 220.



Input Messages

TU_F_HOLD_REPLY | TU_F_SIGNOFF

Message TU_F_REPLY

TU_F_REPLY
- REPLY_NUM
- REPLY_IND
- REPLY_G_MSG
- REPLY_E_MSG
- REPLY_NAME
- REPLY_ADDRESS
- REPLY_CITY
- REPLY_STATE
- REPLY_ZIP
- REPLY_PHONE

Output Messages

SYS_FEPI_OVERRIDES | TU_F_SGNOFF

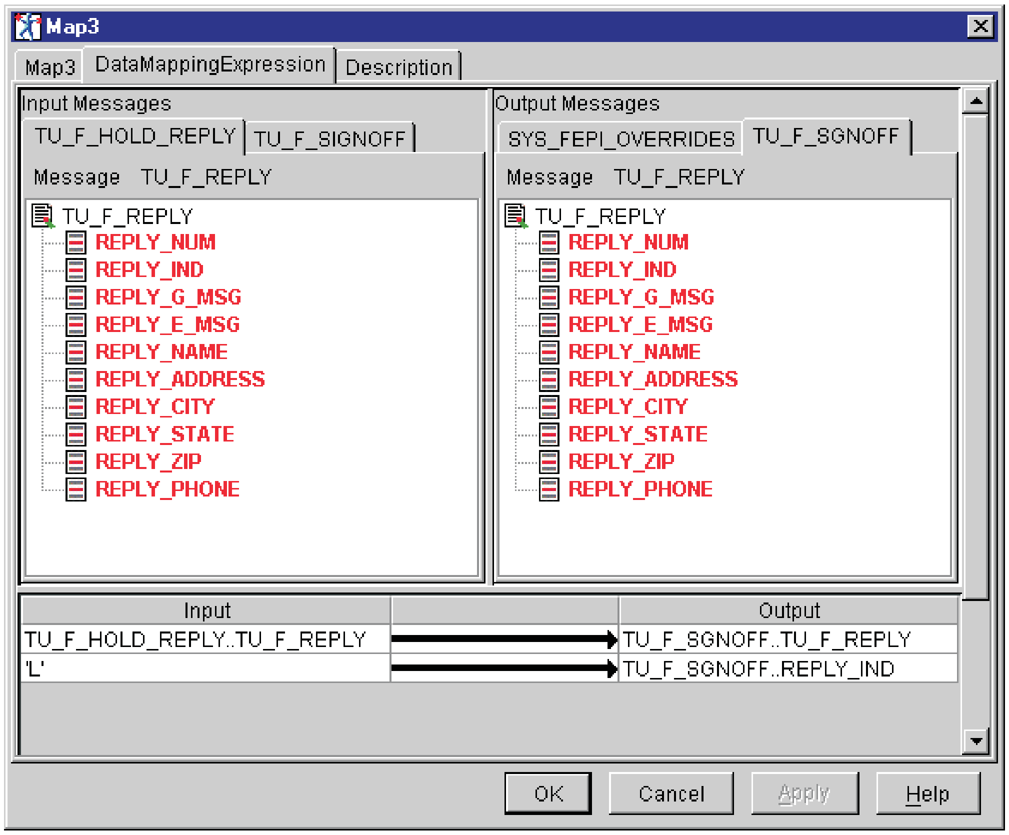
Message SYS_FEPI_OVERRIDES

SYS_FEPI_OVERRIDES
- RELEASE_LU_IND
- TIMEOUT
- POOL
- TARGET

Input		Output
'R'	→	SYS_FEPI_OVERRIDES..RELEASE_LU_IND

Figure 170. Mapping for Map3 node (SYS_FEPI_OVERRIDES message)

e) Click **OK**.



Input		Output
TU_F_HOLD_REPLY..TU_F_REPLY	→	TU_F_SGNOFF..TU_F_REPLY
'L'	→	TU_F_SGNOFF..REPLY_IND

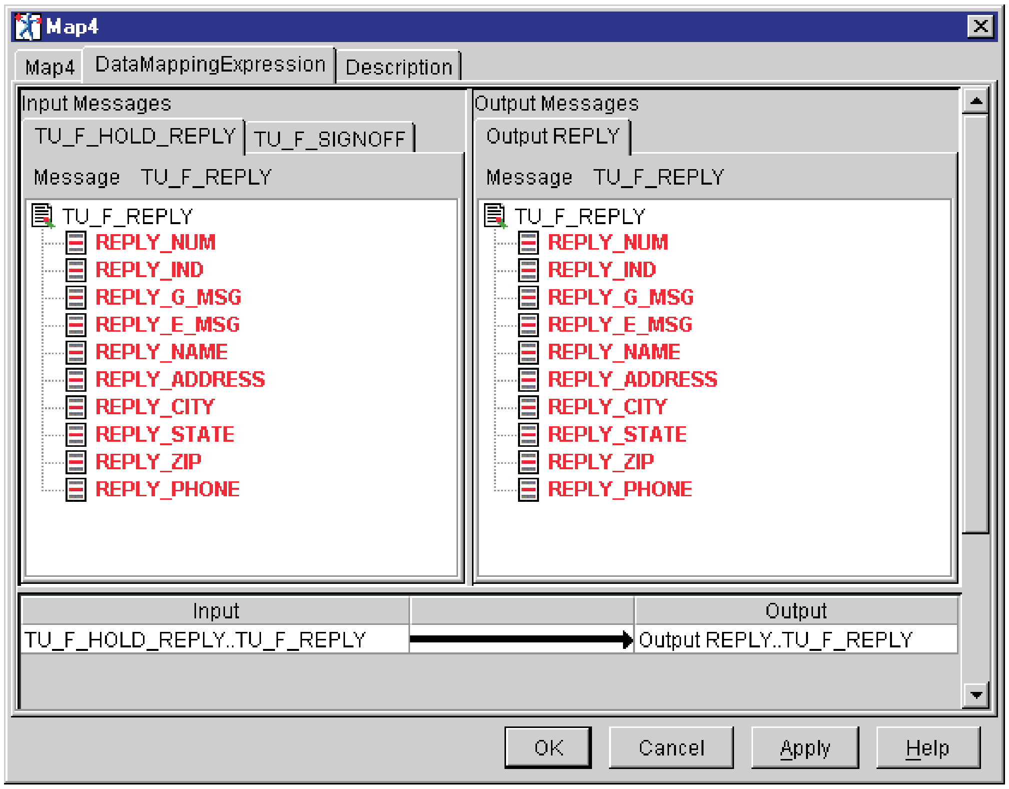
Figure 171. Mapping for Map3 node (TU_F_REPLY message to TU_F_REPLY message)

4) Perform the mapping for the Map4 node as shown in Figure 172 on page 221 .

This mapping passes along the REPLY record in the TU_F_HOLD_REPLY data context to the Output REPLY.

Right click on the Map4 node (the Map node that appears between the TU_F_SIGNOFF and Output REPLY nodes) and select **Properties**. Click the DataMappingExpression tab.

Select the TU_F_HOLD_REPLY message tab (Input Messages) and Output REPLY message tab (Output Messages). Left click on the TU_F_REPLY message under the TU_F_HOLD_REPLY message tab and drag the mouse cursor to the TU_F_REPLY message under the Output REPLY message tab. This will create a mapping between the two messages (see Figure 172 on page 221). Click **OK**.



Message TU_F_REPLY

TU_F_REPLY
- REPLY_NUM
- REPLY_IND
- REPLY_G_MSG
- REPLY_E_MSG
- REPLY_NAME
- REPLY_ADDRESS
- REPLY_CITY
- REPLY_STATE
- REPLY_ZIP
- REPLY_PHONE

Message TU_F_REPLY

TU_F_REPLY
- REPLY_NUM
- REPLY_IND
- REPLY_G_MSG
- REPLY_E_MSG
- REPLY_NAME
- REPLY_ADDRESS
- REPLY_CITY
- REPLY_STATE
- REPLY_ZIP
- REPLY_PHONE

Input		Output
TU_F_HOLD_REPLY..TU_F_REPLY	$\longrightarrow$	Output REPLY..TU_F_REPLY

Figure 172. Mapping for Map4 node

5) Perform the mapping for the Map5 node as shown in Figure 173 on page 222 .

   This mapping passes along the REPLY record in the TU_F_GOOD_SIGNON decision to the Output REPLY.

   Right click on the Map5 node (the Map node that appears between the TU_F_GOOD_SIGNON and Output REPLY nodes) and select **Properties**. Click the DataMappingExpression tab.

   Left click on the TU_F_REPLY message under the TU_F_GOOD_SIGNON message tab and drag the mouse cursor to the TU_F_REPLY message under the Output REPLY message tab. This will create a mapping between the two messages (see Figure 173 on page 222). Click **OK**.

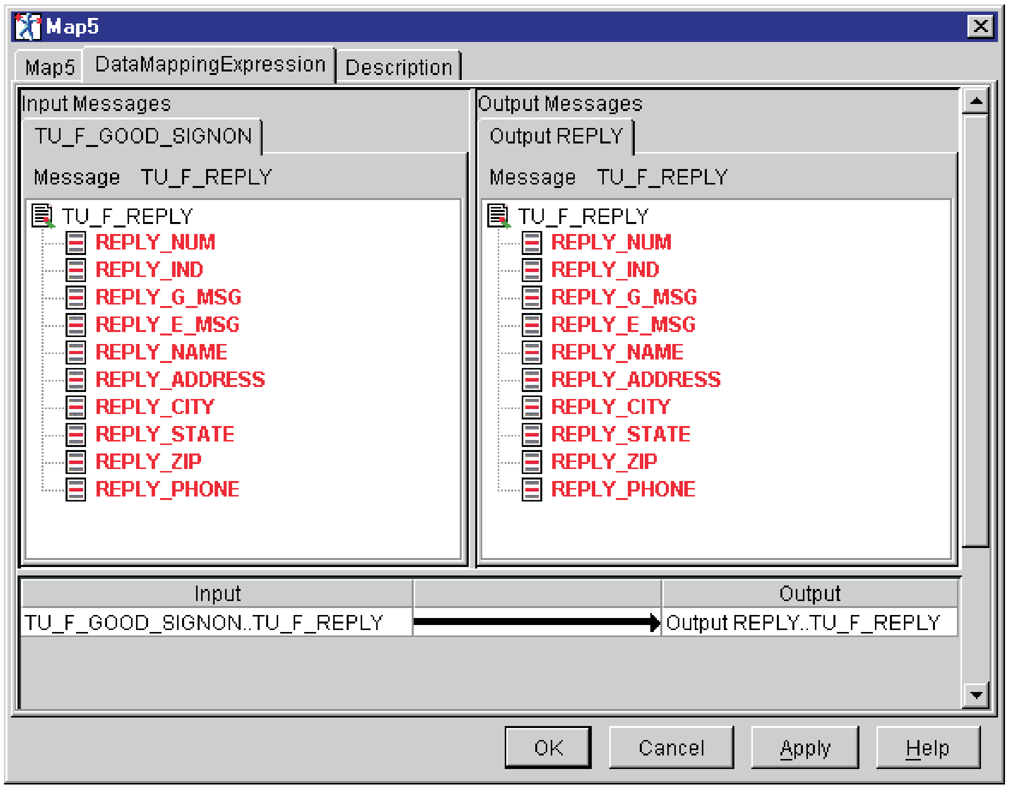


Figure 173. Mapping for Map5 node

6) Perform the mapping for the Map6 node as listed in Table 71 and shown in Figure 174 on page 223.

   This mapping occurs in the case where an Unknown screen is received from the TU_F_SCR_ID decision. The mapping sets the REPLY_IND to ’E’ and creates an error message.

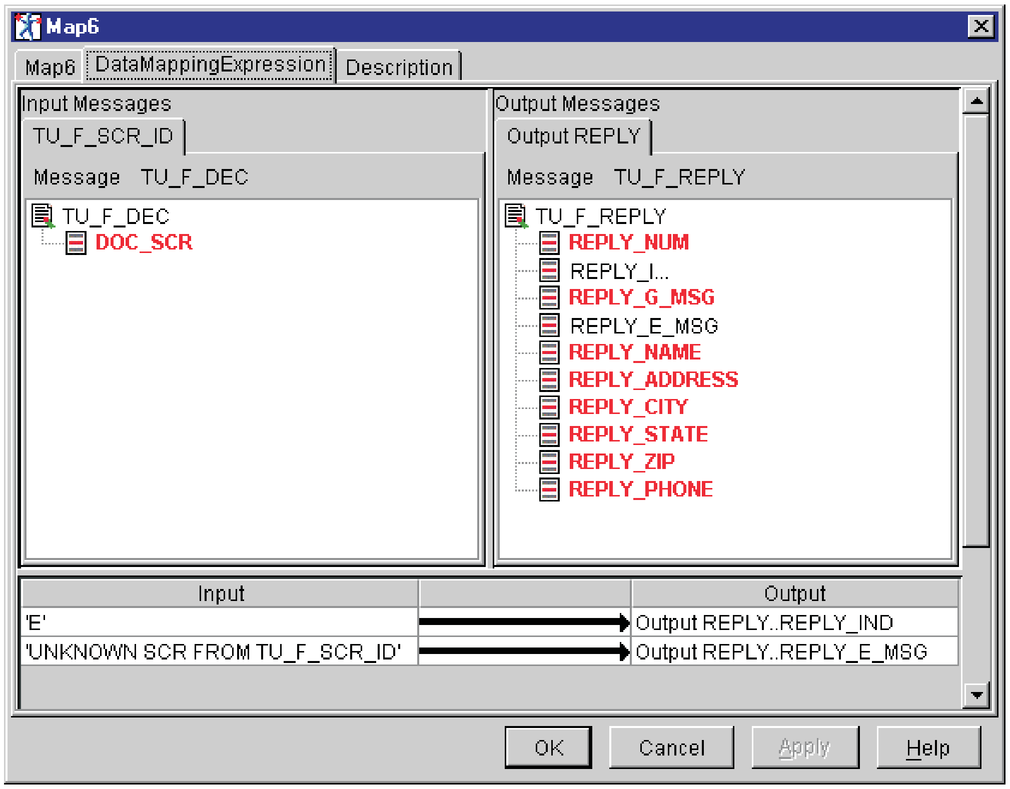
   a) Perform a literal mapping. Right click on the destination field for the literal (the REPLY_IND field in the TU_F_REPLY Message) and select **Add element**. This will create a mapping that is labeled LITERAL on the input field. Double click on LITERAL field and rename it to ’E’ (quotes must be used).

   b) Perform a second literal mapping. Right click on the destination field the REPLY_E_MSG field in the TU_F_REPLY Message and select **Add element**. This will create a mapping that is labeled LITERAL on the input field.

   c) Double click on LITERAL field and rename it to ’UNKNOWN SCR FROM TU_F_SCR_ID’ (quotes must be used). Click **OK**.

Table 71. Mapping fields for Map6 node (Unknown message to TU_F_REPLY message)

Input Field	Output Field	Description
’E’	REPLY_IND	Sets REPLY_IND to ’E’ to indicate an error
’UNKNOWN SCR FROM TU_F_SCR_ID’	REPLY_E_MSG	Error message



Input		Output
'E'	→	Output REPLY..REPLY_IND
'UNKNOWN SCR FROM TU_F_SCR_ID'	→	Output REPLY..REPLY_E_MSG

Figure 174. Mapping for Map6 node

You just modelled the Navigator microflow.

In your model, you have coded the instructions on how the Navigator is supposed to behave at run time. You are now ready to assign this subflow to an CICS MQAdapter.

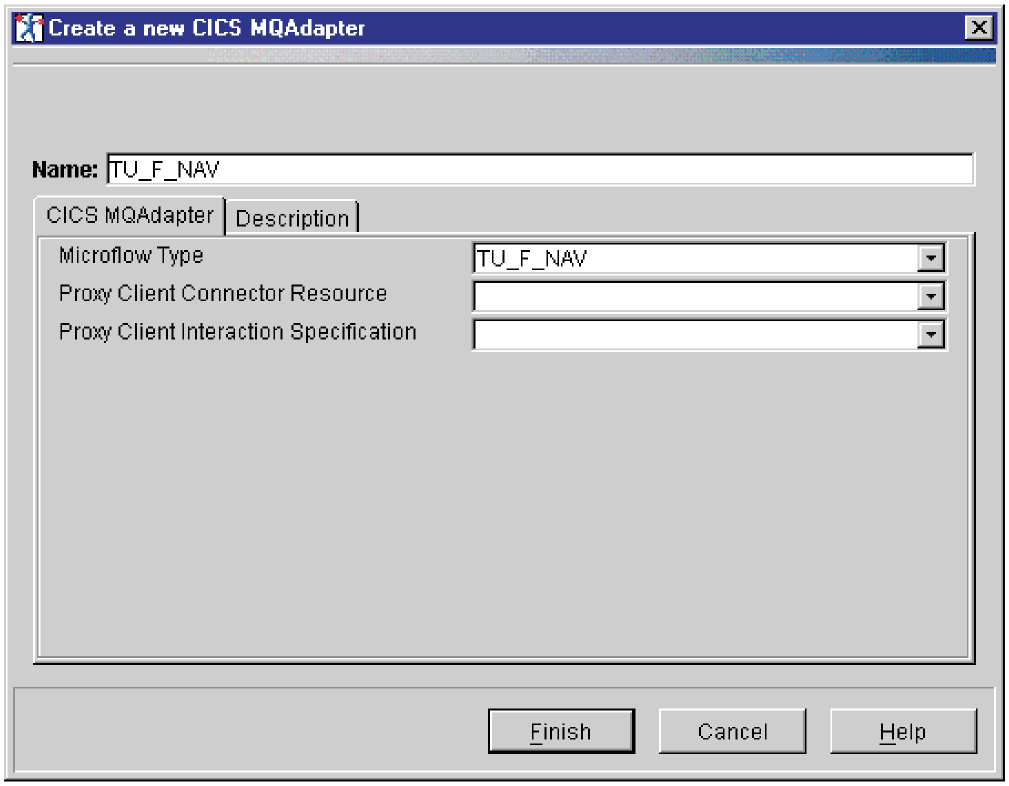
3. **Assign the model of the Navigator microflow to a CICS MQAdapter**.

   In this step you will associate the Navigator microflow (the model that you just completed), with a CICS MQAdapter.

   A CICS MQAdapter provides the actual implementation of the adapter request processing.

   a. Right click on the CICS MQAdapter Collection folder and select **Create > CICS MQAdapter**

   b. On the Create a new CICS MQAdapter dialog, enter TU_F_NAV for the Name and use the drop down menu to select TU_F_NAV for the Microflow Type. Leave the Proxy Client Connector Resource and Proxy Client Interaction Specification fields blank. Click **Finish**.



Name: TU_F_NAV

Microflow Type: TU_F_NAV
Proxy Client Connector Resource:
Proxy Client Interaction Specification:

Figure 175. Creating an CICS MQAdapter

You have completed the microflow and setup your adapter.

Save your workspace by selecting **File > Save Workspace** from the menubar.

Now you are ready to generate your adapter.

Note: The copybooks were previously generated during the parser subflow (TU_F_PARSER) adapter modeling section (see 2a on page 148).

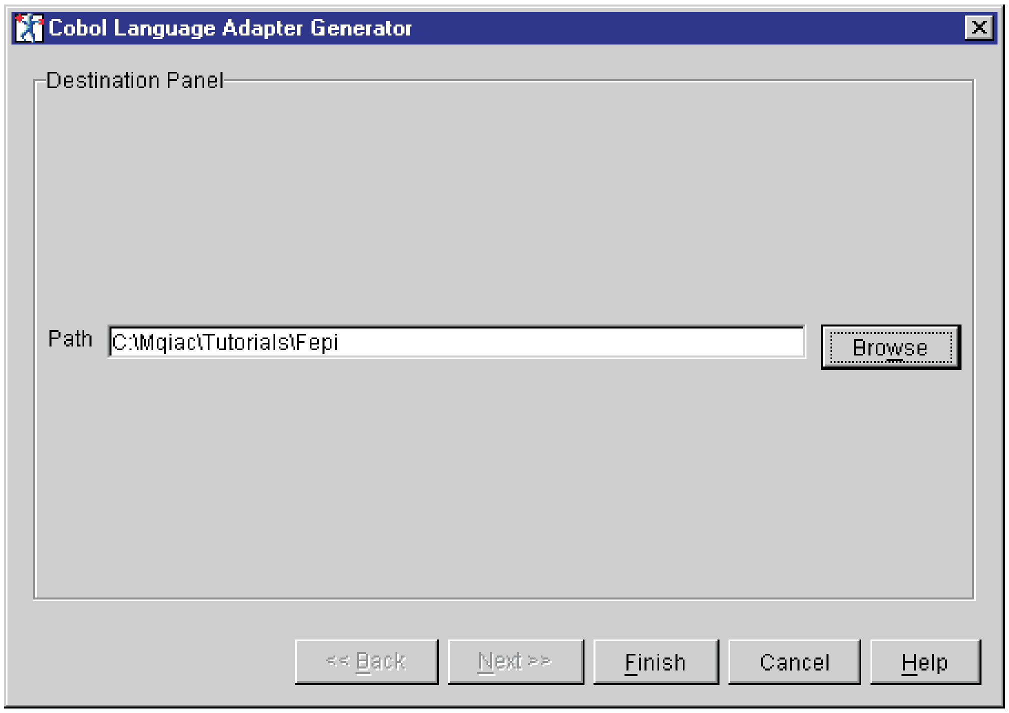
4. **Generate adapter Code**.

   To generate adapter code, make sure that you are in the Adapters view and then, follow this procedure:

   Note: You must generate the adapter code in the same directory where you generated the copybooks.

   a. Right click on the TU_F_NAV adapter (listed under the CICS MQAdapters folder) and select **Generate > Generate COBOL Adapter**. Enter the output destination **<mqiac_tutorials>\fepi** in the PATH field (the example uses C:\Mqiac\Tutorials\FEPI). Click **Finish**.

      The generated adapter code will be output to the destination path directory.



Path C:\Mqiac\Tutorials\Fepi

Figure 176. Specifying pathname for adapter code generation output

You have created all of the microflows that are required for the FEPI adapter.

Deploying an adapter

GOAL

In the following section you will learn how to deploy the adapter that you created. The deploy operation sends the copybooks, source code, JCL and the configuration parameters for each microflow that you generated, to the host system, for source code configuration, object code build and parameter update operations.

You must deploy all the subflows prior to deploying the Navigator flow.

You now need to deploy the adapters that you have generated. The following adapters need to be deployed:

- TU_F_PARSER (subflow)
- TU_F_SIGNON (subflow)
- TU_F_INQ (subflow)
- TU_F_SGNOFF (subflow)
- TU_F_RESET (subflow)
- TU_F_NAV (Navigator microflow)

Note: You will notice that all subflows will be compiled during the deployment of the Navigator microflow (TU_F_NAV). This does not mean that you can deploy only the Navigator as a shortcut. Each microflow must be deployed individually.

Build an adapter that supports a FEPI interface

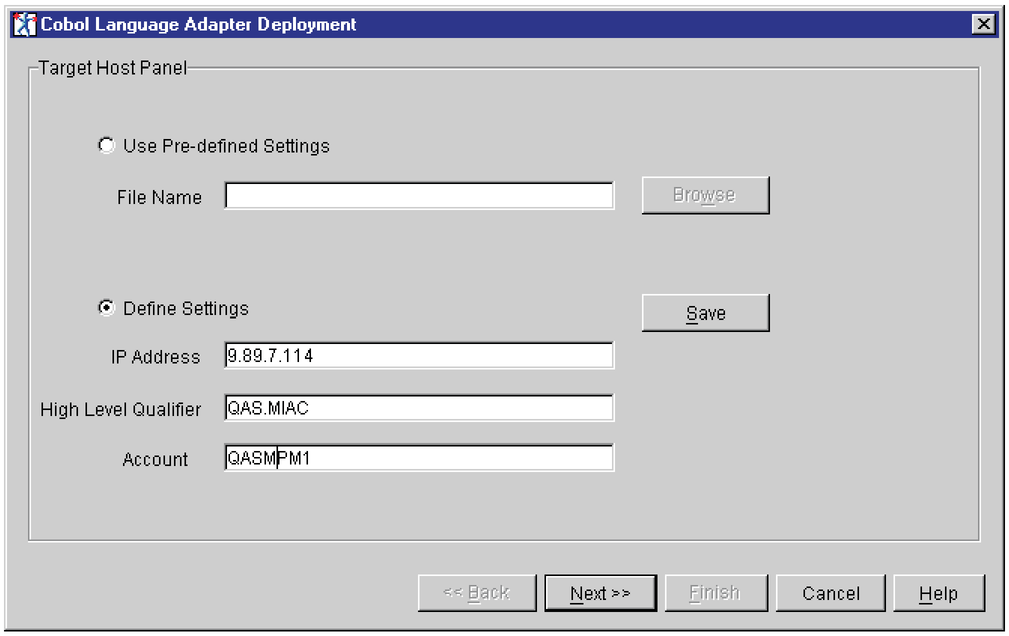
The Navigator (TU_F_NAV) has no knowledge of the copybooks and file structures required by each subflow. Therefore, deploying only the Navigator will result in compile errors for each subflow.

To deploy an adapter, make sure that you are in the Adapters view and then, follow this procedure:

1. Right click on the adapter (listed under the CICS MQAdapters folder) and select **Generate > Deploy COBOL Adapter**. Click the Define Settings radio button and enter the following information:
   - IP Address — IP Address - The host system IP address (for example, 9.89.7.114)
   - High Level Qualifier — The high level qualifier for the partition data set (PDS)
   - Account — The account under which JCL submits a job for compilation.

   Note: If you wish to save these settings for reuse, then click **Save**. You will be prompted to specify an output location and filename to store the setting information. The next time you deploy adapter code you can click the Use Pre-defined Settings radio button and enter the saved filename.

   Click **Next**.



Use Pre-defined Settings

File Name

[✓ Define Settings]

IP Address 9.89.7.114

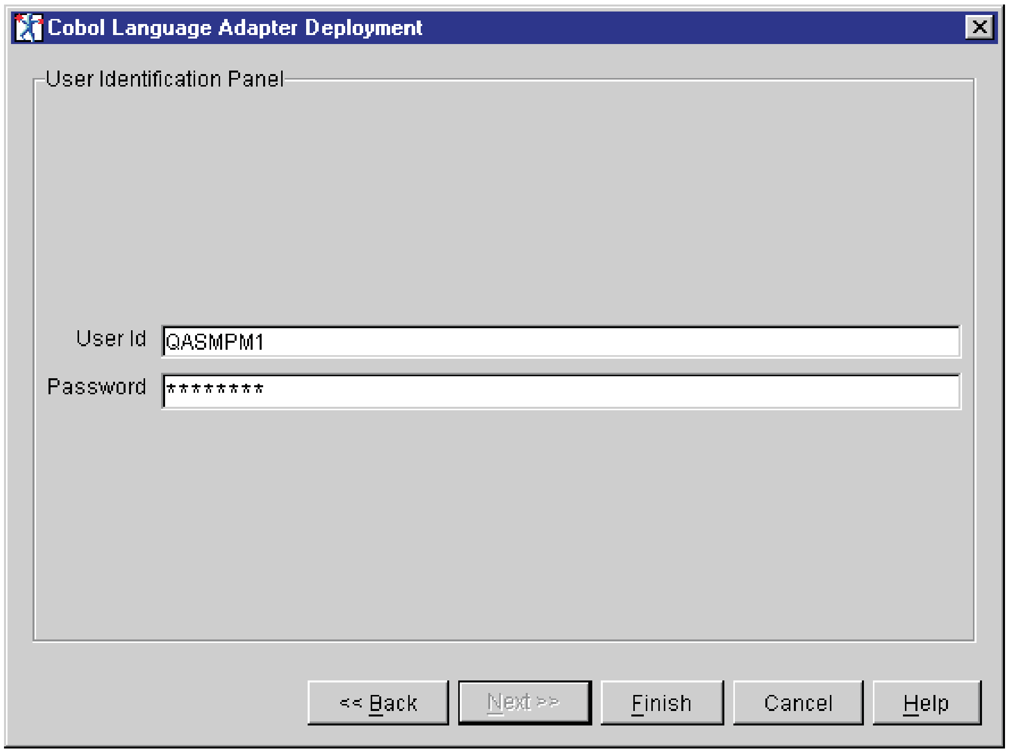
High Level Qualifier QAS.MIAC

Account QASMPM1

Figure 177. Specifying the target host

2. On the User Identification panel enter your user ID and password. Click **Finish**.

   The generated adapter code, copybooks, and JCL (Compile / Properties File Update) files will be moved to the OS/390 server



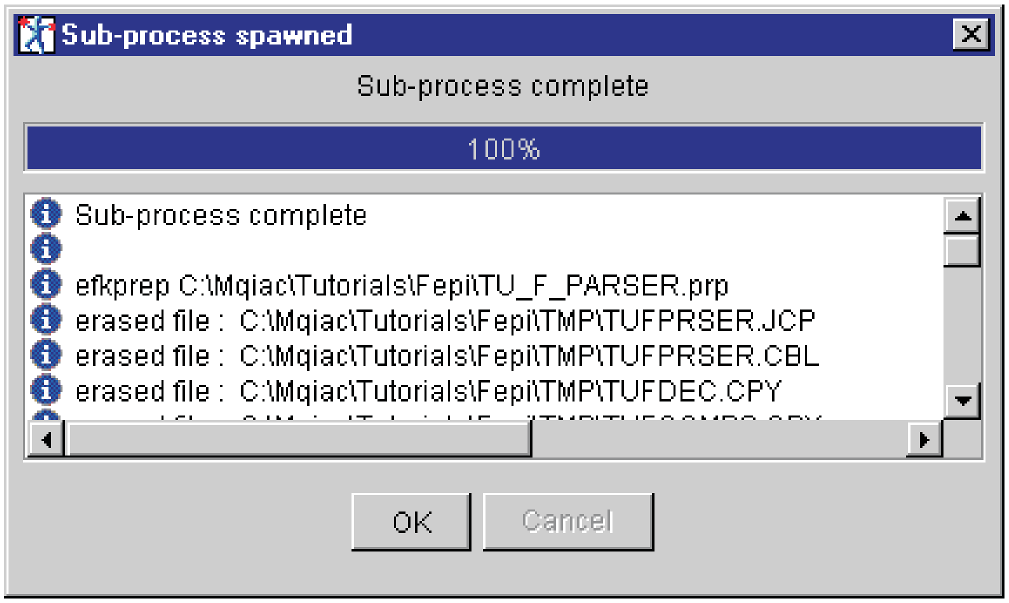
User Id QASMPM1

Password ********

Figure 178. Logon to the host

3. The *Sub-process dialog* appears and provides a status of the deploy process as it happens. When the deploy is complete the generated adapter code, copybooks, and JCL (Compile / Properties File Update) files will be moved to the OS/390 server.

   Note: You should scroll through the output listing in the Sub-process dialog window to see if any errors occurred.



```
Sub-process complete

efkprep C:\Mqiac\Tutorials\Fepi\TU_F_PARSER.prp
erased file :  C:\Mqiac\Tutorials\Fepi\TMP\TUFPRSER.JCP
erased file :  C:\Mqiac\Tutorials\Fepi\TMP\TUFPRSER.CBL
erased file :  C:\Mqiac\Tutorials\Fepi\TMP\TUFDEC.CPY
```

Figure 179. Sub-process dialog indicating status of the deploy process

4. Select **OK** to close the dialog.

You have completed the deploy steps and the adapter now resides on the OS/390 server and is ready to be tested. See Chapter 6, "Validating the adapters" on page 231 for instructions on how to test the adapter.

Check to see that the adapter compiled in CICS

GOAL

In the following sections you will you will perform a series of tasks designed to test and validate that the adapters that you created were successfully deployed to the host.

After you have deployed the adapter to the OS/390 server, you need to make sure that it compiled with no errors. Consult with your CICS systems administrator for assistance with this procedure.

Defining the adapter resources to CICS

If you do not have access to CICS at your site, you will need to ask your CICS administrator to perform the necessary CEDA and CEMT functions. You will need to provide the CICS administrator with the following information as it relates to the adapter that you deployed:

- Program names
- Group name
- Transaction Identifiers

For the FEPI adapter, the following values apply:

Table 72. Values for the Define Transactions screen

Program	Group	Transid
TUFNAV	MIACUSER	TUF1
TUFSGON	MIACUSER	TUF2
TUFSGOFF	MIACUSER	TUF3
TUFPRSER	MIACUSER	TUF4
TUFRESET	MIACUSER	TUF5
TUFINQ	MIACUSER	TUF6

To define resources to CICS, the CICS administrator must:

- Run the CEDA transaction to define programs and any files to CICS.
- Submit JCL to run the Properties File Update job.

  This is necessary only if you did not automatically submit JCL using the generator facility in the builder.

  If you were not allowed to submit JCL automatically, you can manually submit JCL (DFHMAMPU) to run the Properties File Update job (DFHMAMUP). See the MQSI Agent for CICS Run Time User's Guide for information on the Properties file update JCL (DFHMAMPU).

The CICS administrator must NEWCOPY any server adapter programs that were modified.

For an example of defining CICS resources to CICS, See “Example procedure for defining adapter resources to CICS” on page 239.

Chapter 6. Validating the adapters

The adapter validation process employs the same Simulator program that was used in the MQSI Agent for CICS run time installation verification procedure. This program formats and submits a request message, in order to simulate a controlling application that is requesting services of the MQSI Agent for CICS run time software.

The Simulator program, along with the same sample back-end programs used in the IVP are incorporated as part of this tutorial to simulate run time processing, including processing the back-end transactions associated with each type of adapter you modeled.

How the Simulator works

Upon receipt of the request message from the Simulator, the MQSI Agent for CICS invokes the server adapter program (DPL, MQ or FEPI) identified in the request.

The server adapter performs adapter request processing, as modeled, which results in a reply message. A successful reply (GOOD RESPONSE RECEIVED) means that the server adapter was deployed successfully.

See Figure 180 on page 232 for an illustration of the processing that occurs when you issue a request using the simulator.

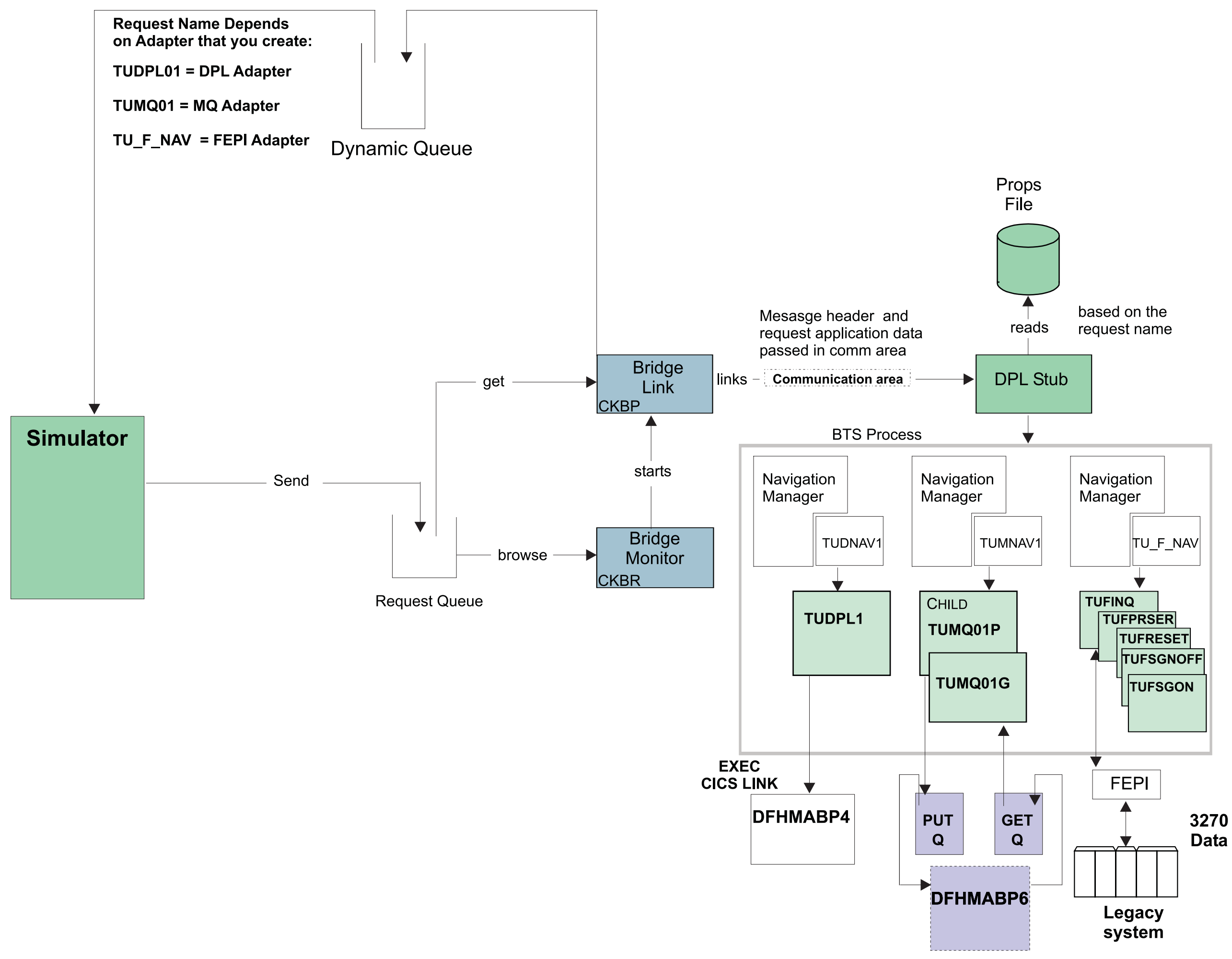


Figure 180. Request processing using the simulator

Preparing to use the Simulator

When you first invoke the Simulator, some of the screens will be populated with data from the Installation Verification Procedure (IVP). Although the adapters you modeled will access the same back-end sample transactions that the IVP accessed, you will need to modify some fields to reflect information about the adapters that you modeled in the tutorials.

Running the Simulator to validate the adapters

See Figure 181 on page 233 for a flow of the Simulator invoking an adapter that you modeled in the tutorial.

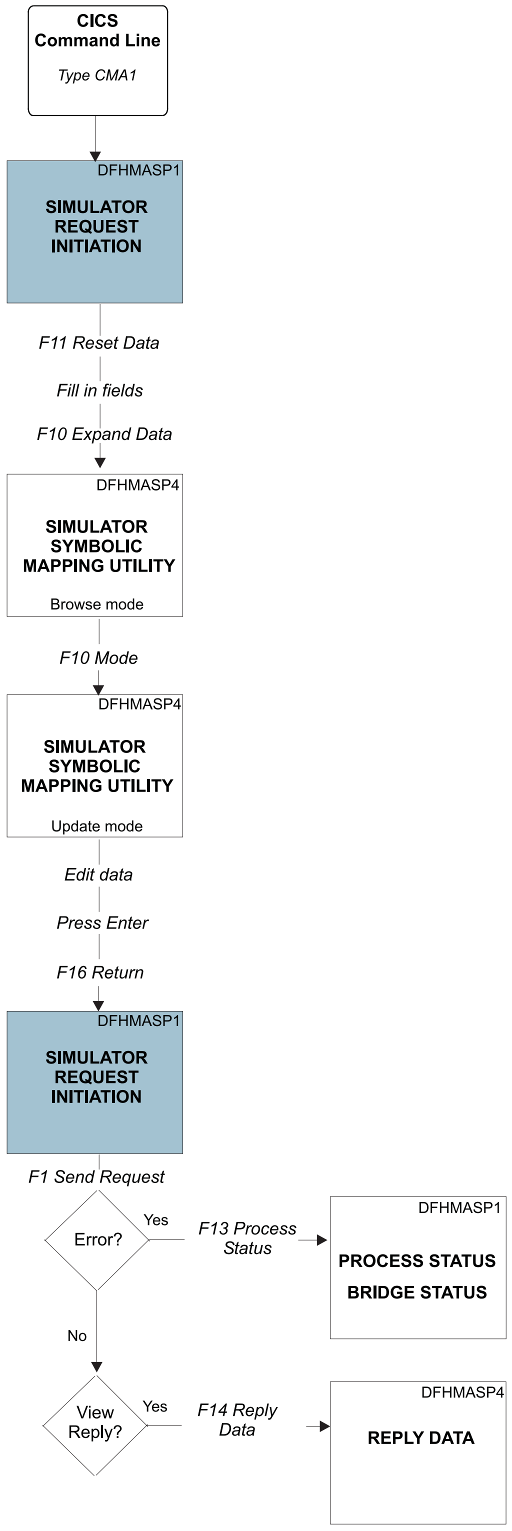


Figure 181. Flow of an adapter being run from the Simulator

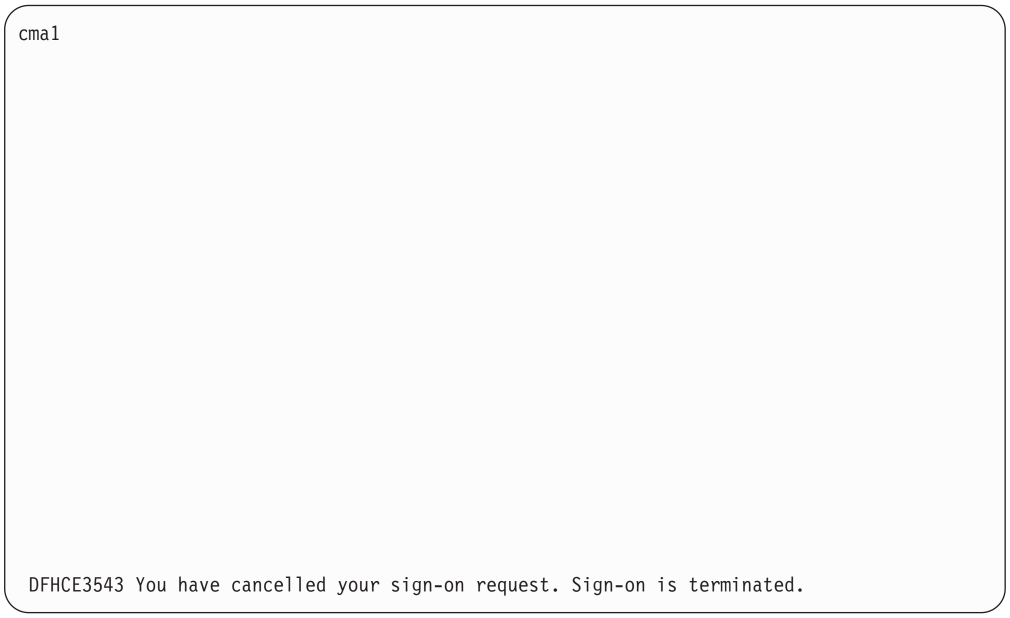
Perform the following steps to validate the adapters that you have deployed:

__ 1. Log on to the OS/390 host system

__ 2. Start the Simulator.

Validating the adapters

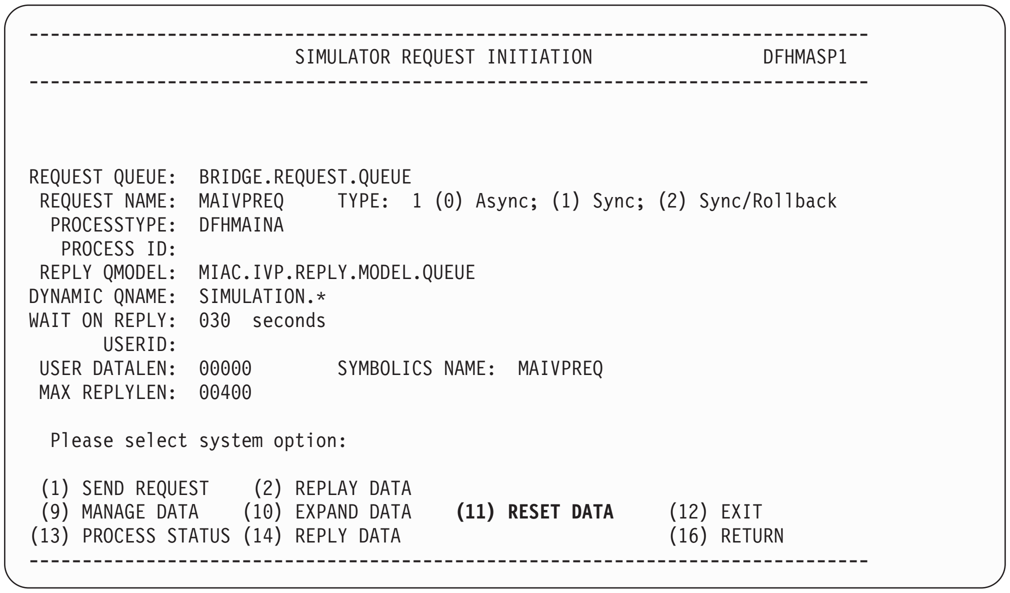
From the CICS command line type **CMA1**.
CMA1 is the transaction identifier for the Simulator.



```
cma1

 DFHCE3543 You have cancelled your sign-on request. Sign-on is terminated.
```

Figure 182. Simulator transaction

__ 3. Press **Enter**. The *Simulator Request Initiation* screen displays.



```
 ------------------------------------------------------------------------------
                          SIMULATOR REQUEST INITIATION                DFHMASP1
 ------------------------------------------------------------------------------



 REQUEST QUEUE:  BRIDGE.REQUEST.QUEUE
  REQUEST NAME:  MAIVPREQ     TYPE:  1 (0) Async; (1) Sync; (2) Sync/Rollback
   PROCESSTYPE:  DFHMAINA
    PROCESS ID:
  REPLY QMODEL:  MIAC.IVP.REPLY.MODEL.QUEUE
 DYNAMIC QNAME:  SIMULATION.*
 WAIT ON REPLY:  030  seconds
        USERID:
  USER DATALEN:  00000        SYMBOLICS NAME:  MAIVPREQ
  MAX REPLYLEN:  00400

   Please select system option:

  (1) SEND REQUEST    (2) REPLAY DATA
  (9) MANAGE DATA    (10) EXPAND DATA    (11) RESET DATA     (12) EXIT
 (13) PROCESS STATUS (14) REPLY DATA                         (16) RETURN
 ------------------------------------------------------------------------------
```

Figure 183. Simulator request initiation screen — Initial appearance

__ 4. Press **F11 Reset Data**.

Type the following values.

REQUEST QUEUE:
X(48). Accept the default value. This is the CICS Bridge Request Queue.

REQUEST NAME:
X(8). The request name indicates the name of the request to process. The value specified is used to read the MQSI Agent for CICS Properties file to determine processing flow and parameters. Valid values are as follows:

- For MQ Adapter, enter **TUMQ01**
- For DPL Adapter, enter **TUDPL01**
- For FEPI Adapter, enter **TU_F_NAV**

TYPE: X(1). Indicates the processing mode. Enter **0** for asynchronous processing.

- 0 = request processing will be run asynchronously. For requests that will be updating information when adapter request processing runs in asynchronous mode, syncpoints are taken while the Navigation Manager, Navigator and server adapter programs complete their processing. These synchronization points provide the necessary state, status and journaling information in the event of subsequent failure. This information can in turn be used in a compensation flow.
- 1 = request processing will be run synchronously. For requests that will be making inquiries only (i.e., the request will not result in information being updated).
- 2 = request processing will include rollback processing. Synchronous Rollback is a processing mode where the MQSI Agent for CICS BTS process and all activities run within the process are initiated and run in synchronous mode (i.e., BTS RUN ACQPROCESS SYNCHRONOUS and RUN ACTIVITY( ) SYNCHRONOUS commands), however, any failure within any activity within the process results in an abend of the process. This has the effect of returning the state of any and all recoverable resources updated during adapter request processing to its original state, that is, the state prior to the execution of the failed adapter request or process.

PROCESSTYPE:
X(8). Accept the default value. This field indicates the type of the new MQSI Agent for CICS run time process instance.

PROCESS ID:
Leave blank.

REPLY QMODEL:
X(48). Accept the default value. Name of the model reply queue.

DYNAMIC QNAME:
X(48). Accept the default value. Prefix used for the name of the Reply Queue when dynamically built.

WAIT ON REPLY:
9(4). Identifies how long the Simulator will wait for a response before timing out. Accept the default value of 30 seconds.

USERID:
X(8). For MQ and DPL adapters you can leave this blank. For FEPI, enter your valid CICS user ID. This will be used for authentication in MQSeries-CICS bridge and to allocate a PassTicket in the FEPI Navigator. For the FEPI Navigator, this field corresponds to the MAT_USELUPASS field that is set to Y in the Resource files.

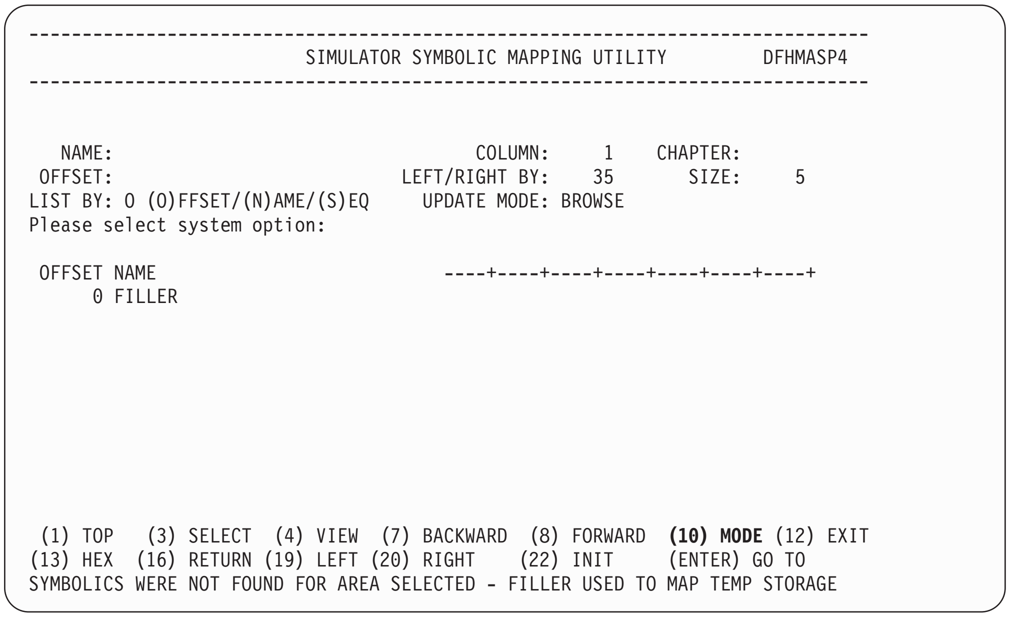
USER DATALEN:
: 9(5). For DPL and MQ type **00005**. For FEPI type **00006**. This indicates the length of the user data in the request message.

SYMBOLICS NAME:
: Leave blank.

MAX REPLYLEN:
: 9(5). For DPL and MQ type **00401**. For FEPI type **00400**. This indicates the length of reply data in the reply message.

__ 5. Select **PF 10 EXPAND DATA**

The *Simulator symbolic mapping utility* screen appears



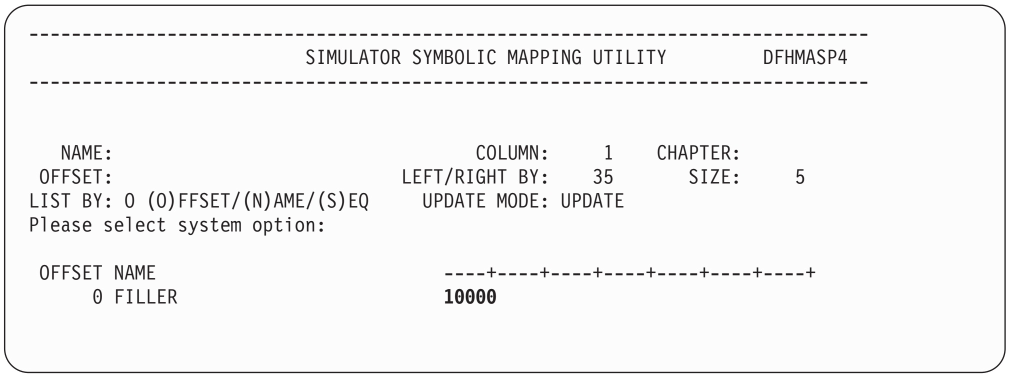
```
-------------------------------------------------------------------------------
                          SIMULATOR SYMBOLIC MAPPING UTILITY         DFHMASP4
-------------------------------------------------------------------------------


   NAME:                                  COLUMN:     1    CHAPTER:
 OFFSET:                           LEFT/RIGHT BY:    35       SIZE:     5
LIST BY: O (O)FFSET/(N)AME/(S)EQ     UPDATE MODE: BROWSE
Please select system option:

 OFFSET NAME                           ----+----+----+----+----+----+----+
      0 FILLER









 (1) TOP   (3) SELECT  (4) VIEW  (7) BACKWARD  (8) FORWARD  (10) MODE (12) EXIT
(13) HEX  (16) RETURN (19) LEFT (20) RIGHT    (22) INIT     (ENTER) GO TO
SYMBOLICS WERE NOT FOUND FOR AREA SELECTED - FILLER USED TO MAP TEMP STORAGE
```

Figure 184. Simulator Symbolic Mapping Utility

Select **F10 MODE** to change from BROWSE mode to **UPDATE** mode.

For MQ and DPL, tab to data entry area on screen and type **10000** .

For FEPI you must account for the SIGNOFF field for releasing the LU. So tab to the data entry area on screen and type **10000Y**. The Y will release the LU. If you enter any other value but Y, the LU will not be released.



```
--------------------------------------------------------------------------------
                          SIMULATOR SYMBOLIC MAPPING UTILITY        DFHMASP4
--------------------------------------------------------------------------------



   NAME:                                  COLUMN:     1    CHAPTER:
 OFFSET:                           LEFT/RIGHT BY:    35       SIZE:     5
LIST BY: O (O)FFSET/(N)AME/(S)EQ     UPDATE MODE: UPDATE
Please select system option:

 OFFSET NAME                           ----+----+----+----+----+----+----+
      0 FILLER                         10000
```

Figure 185. Updated symbolic mapping utility screen

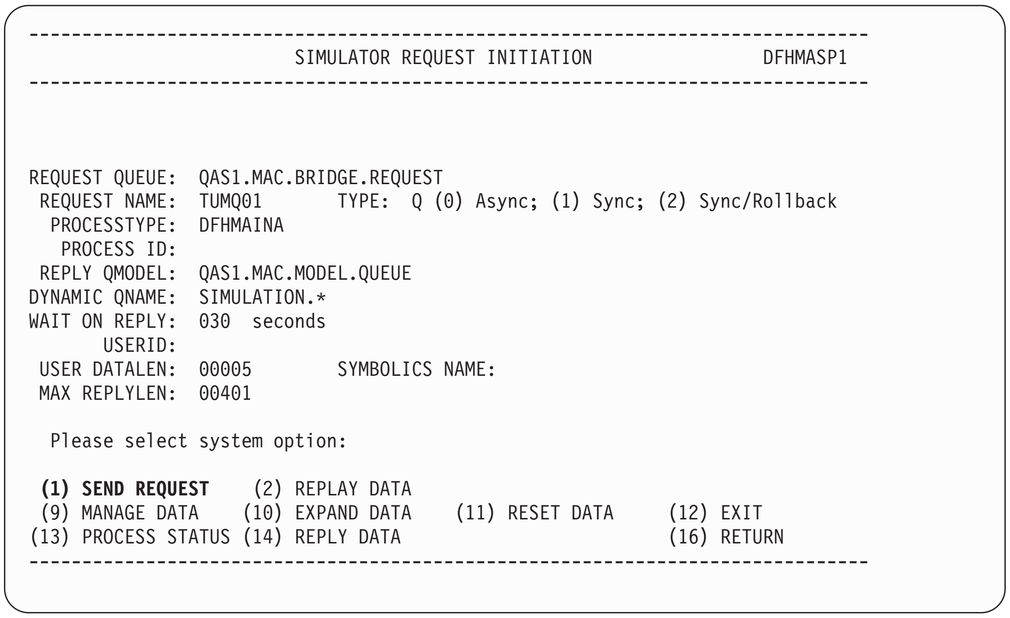
__ 6. Select **(ENTER).**

__ 7. Select **F16 RETURN** to go back to the Simulator request initiation screen.

__ 8. For **FEPI Adapter only**

Before you send the request message, make sure that FEPI is running by ensuring that the Target, Nodes, Property Set and Pool are in service. You may need to check with your CICS administrator.

__ 9. Select **F1 SEND REQUEST** to send the request.



```
--------------------------------------------------------------------------------
                         SIMULATOR REQUEST INITIATION               DFHMASP1
--------------------------------------------------------------------------------




REQUEST QUEUE:  QAS1.MAC.BRIDGE.REQUEST
 REQUEST NAME:  TUMQ01       TYPE:  Q (0) Async; (1) Sync; (2) Sync/Rollback
  PROCESSTYPE:  DFHMAINA
   PROCESS ID:
 REPLY QMODEL:  QAS1.MAC.MODEL.QUEUE
DYNAMIC QNAME:  SIMULATION.*
WAIT ON REPLY:  030  seconds
       USERID:
 USER DATALEN:  00005       SYMBOLICS NAME:
 MAX REPLYLEN:  00401

  Please select system option:

 (1) SEND REQUEST    (2) REPLAY DATA
 (9) MANAGE DATA    (10) EXPAND DATA    (11) RESET DATA     (12) EXIT
(13) PROCESS STATUS (14) REPLY DATA                         (16) RETURN
--------------------------------------------------------------------------------
```

Figure 186. Simulator request initiation screen — Sending the request

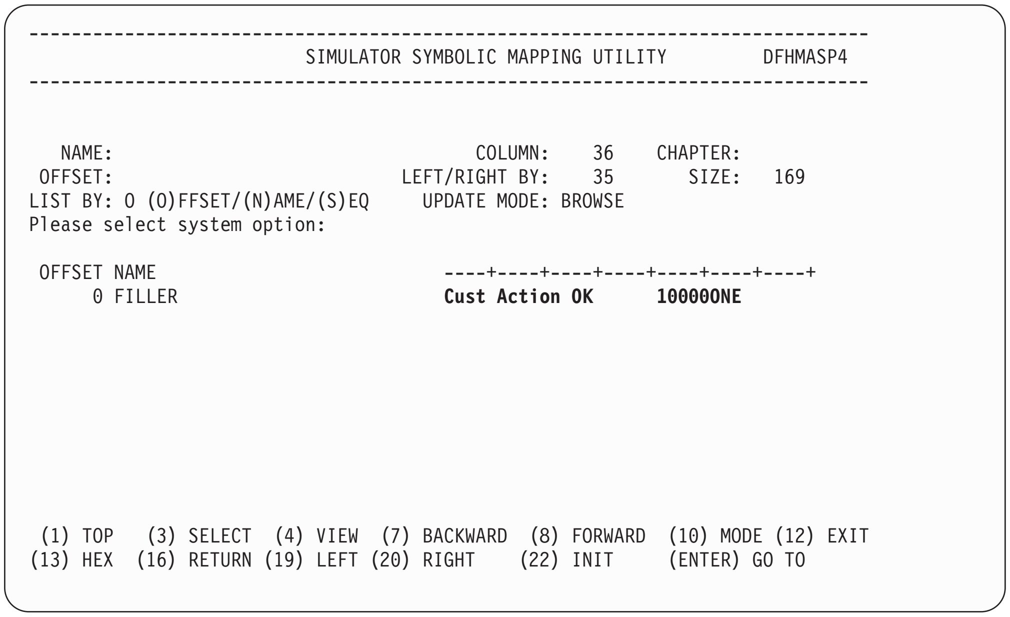
- If the request is processed, you receive a GOOD RESPONSE RECEIVED in the message area of the Simulator Request Initiation screen.
- If the request is not processed, you receive an ERROR RESPONSE RECEIVED in the message area of the Simulator Request Initiation screen. See the section on troubleshooting the simulator in the *MQSeries Integrator Agent for CICS Run Tim User's Guide* for information on how to respond to an ERROR RESPONSE RECEIVED message.

__ 10. **Check the reply to the adapter request**

After you receive a response to the F1 SEND REQUEST, select F14 REPLY DATA from the Simulator Request Initiation screen.

The Simulator Symbolic Mapping Utility screen appears.

Select F20 RIGHT (shift F8) to scroll the screen image to the right. This will allow you to view the reply data:



```
--------------------------------------------------------------------------------
                          SIMULATOR SYMBOLIC MAPPING UTILITY         DFHMASP4
--------------------------------------------------------------------------------



   NAME:                                 COLUMN:    36    CHAPTER:
 OFFSET:                          LEFT/RIGHT BY:    35       SIZE:   169
LIST BY: O (O)FFSET/(N)AME/(S)EQ     UPDATE MODE: BROWSE
Please select system option:

 OFFSET NAME                           ----+----+----+----+----+----+----+
      0 FILLER                         Cust Action OK      100000NE










 (1) TOP   (3) SELECT  (4) VIEW  (7) BACKWARD  (8) FORWARD  (10) MODE (12) EXIT
(13) HEX  (16) RETURN (19) LEFT (20) RIGHT    (22) INIT     (ENTER) GO TO
```

Figure 187. Simulator symbolic mapping utility screen

__ 11. Select **F16 RETURN** to return to the *Simulator Request Initiation Screen*.

From this screen you can reset the data and type in the values required to test the next adapter. See 4 on page 234.

Appendix. Example procedure for defining adapter resources to CICS

After you build and deploy your adapters to CICS, a CICS systems administrator will need to define the adapter resources to CICS.

As CICS environments will vary from site to site, **do not use the information in this example to define your adapter resources to your CICS environment**. The procedures documented in the following sections are examples and they should be used only for reference purposes.

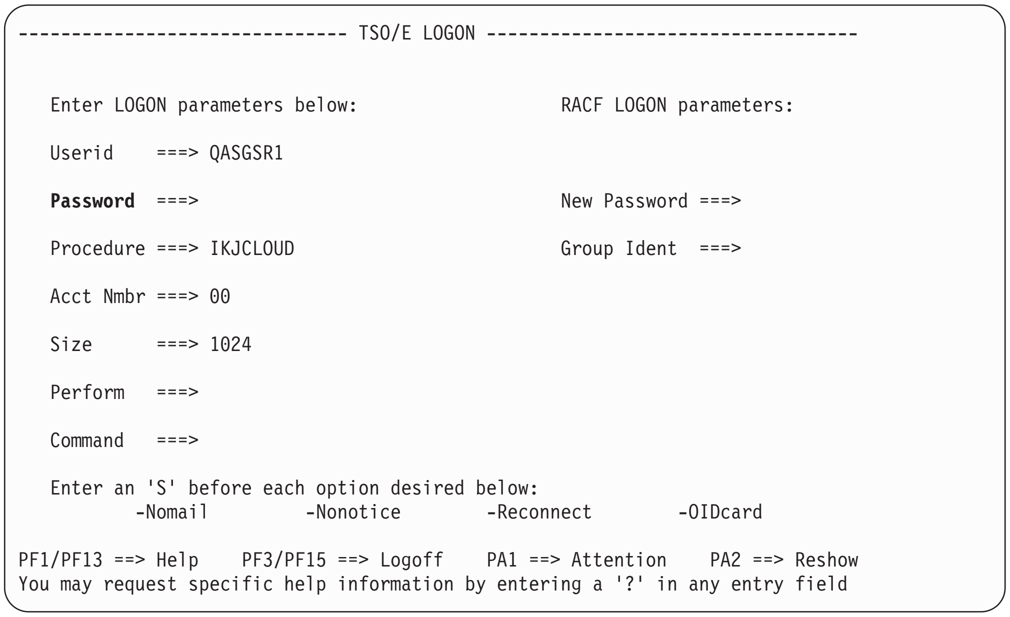
Defining DPL adapter resources to CICS

Check to see that the adapter compiled in CICS

After you have deployed the adapter to the OS/390 server, you need to make sure that it compiled with no errors.

Perform the following steps to check the compile:

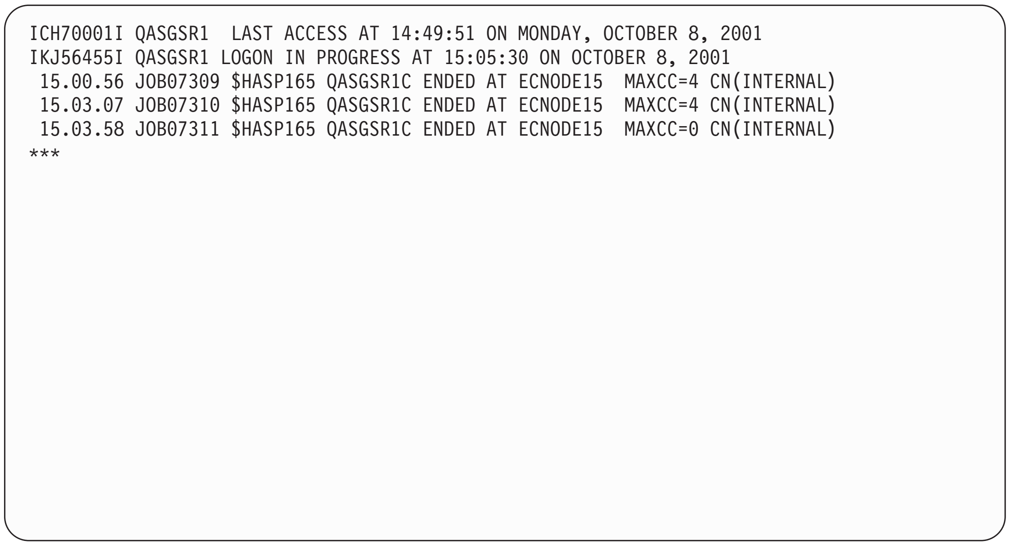
1. Sign on to the OS/390 server and select the option to access TSO.
2. Enter your TSO logon password and select **Enter**.



```
------------------------------- TSO/E LOGON -----------------------------------


   Enter LOGON parameters below:                   RACF LOGON parameters:

   Userid    ===> QASGSR1

   Password  ===>                                  New Password ===>

   Procedure ===> IKJCLOUD                         Group Ident  ===>

   Acct Nmbr ===> 00

   Size      ===> 1024

   Perform   ===>

   Command   ===>

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Figure 188. TSO/E logon screen

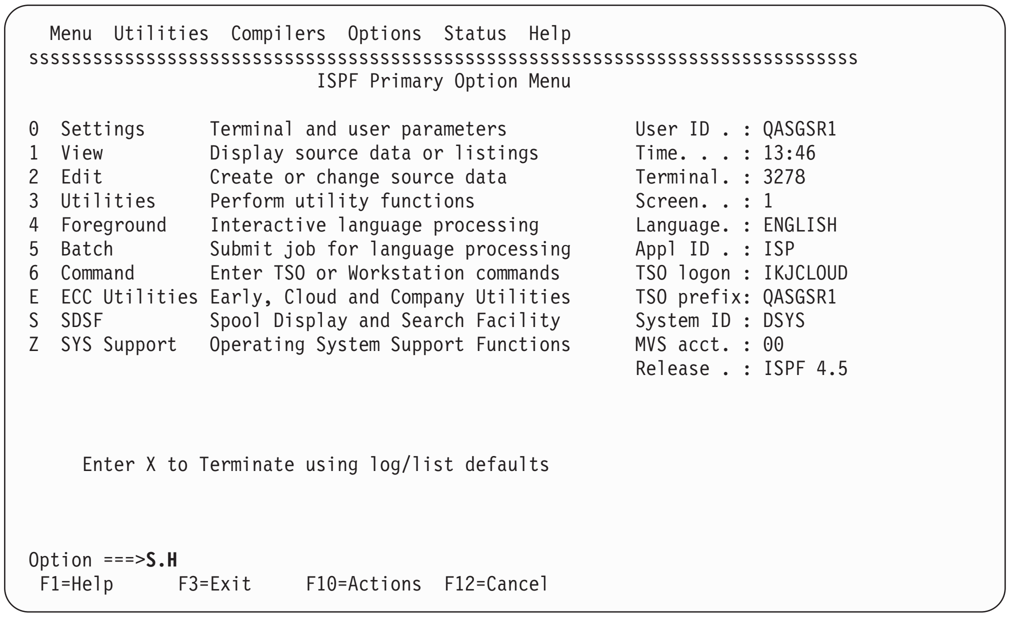
3. Press **Enter** to complete signon.



```
ICH70001I QASGSR1  LAST ACCESS AT 14:49:51 ON MONDAY, OCTOBER 8, 2001
IKJ56455I QASGSR1 LOGON IN PROGRESS AT 15:05:30 ON OCTOBER 8, 2001
 15.00.56 JOB07309 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 15.03.07 JOB07310 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 15.03.58 JOB07311 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=0 CN(INTERNAL)
***
```

Figure 189. List of completed job notifications sent to OS/390 server (for active user id) via the deploy process

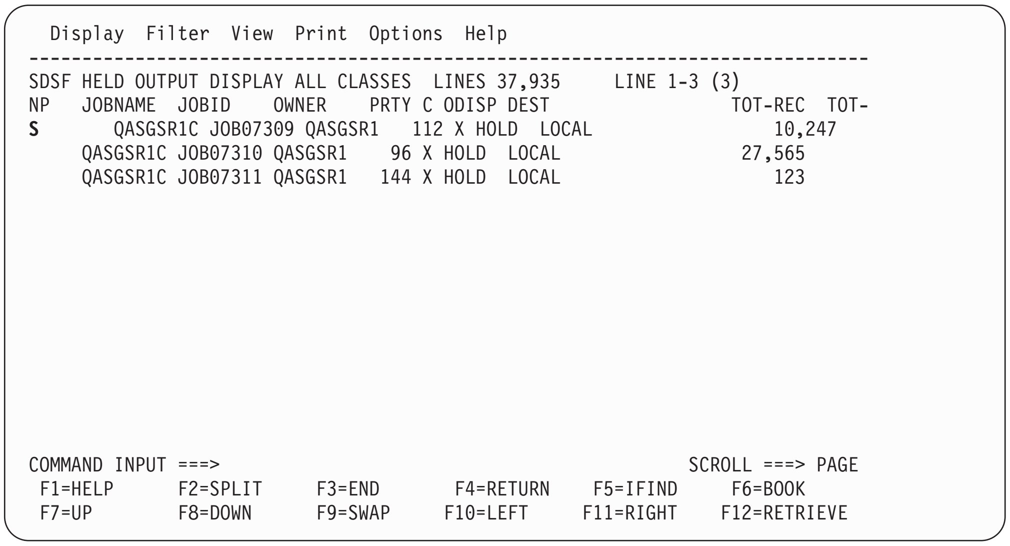
4. Press **Pause** to go to the *ISPF Primary Option Menu*.



```
  Menu  Utilities  Compilers  Options  Status  Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                           ISPF Primary Option Menu

0  Settings      Terminal and user parameters           User ID . : QASGSR1
1  View          Display source data or listings        Time. . . : 13:46
2  Edit          Create or change source data           Terminal. : 3278
3  Utilities     Perform utility functions              Screen. . : 1
4  Foreground    Interactive language processing        Language. : ENGLISH
5  Batch         Submit job for language processing     Appl ID . : ISP
6  Command       Enter TSO or Workstation commands      TSO logon : IKJCLOUD
E  ECC Utilities Early, Cloud and Company Utilities     TSO prefix: QASGSR1
S  SDSF          Spool Display and Search Facility      System ID : DSYS
Z  SYS Support   Operating System Support Functions     MVS acct. : 00
                                                        Release . : ISPF 4.5



     Enter X to Terminate using log/list defaults



Option ===>S.H
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Figure 190. ISPF Primary Option Menu

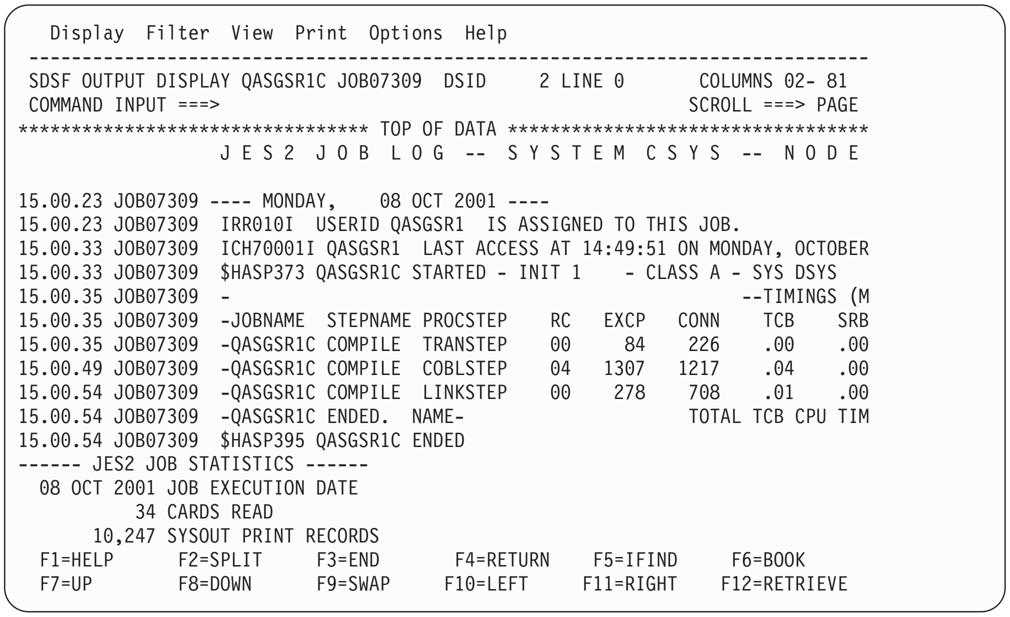
5. Type **S.H** on the Option line and press **Enter** to see go to the compile job listing:



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF HELD OUTPUT DISPLAY ALL CLASSES  LINES 37,935     LINE 1-3 (3)
 NP   JOBNAME  JOBID    OWNER    PRTY C ODISP DEST                TOT-REC  TOT-
 S       QASGSR1C JOB07309 QASGSR1   112 X HOLD  LOCAL                10,247
      QASGSR1C JOB07310 QASGSR1    96 X HOLD  LOCAL               27,565
      QASGSR1C JOB07311 QASGSR1   144 X HOLD  LOCAL                  123















 COMMAND INPUT ===>                                     SCROLL ===> PAGE
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 191. Spool Display and Search Facility Held Output Display screen

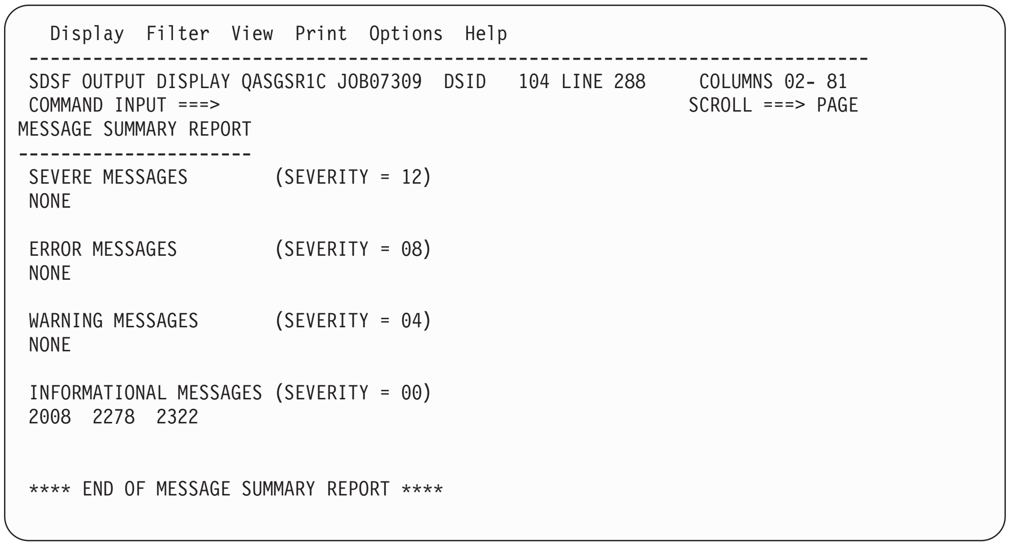
6. To view an output listing, type an **S** to the left of its job name and press **Enter**. This will open (for view purposes) the JES2 JOB LOG:



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB07309  DSID     2 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                     SCROLL ===> PAGE
********************************* TOP OF DATA **********************************
                  J E S 2  J O B  L O G  --  S Y S T E M  C S Y S  --  N O D E

15.00.23 JOB07309 ---- MONDAY,    08 OCT 2001 ----
15.00.23 JOB07309  IRR010I  USERID QASGSR1  IS ASSIGNED TO THIS JOB.
15.00.33 JOB07309  ICH70001I QASGSR1  LAST ACCESS AT 14:49:51 ON MONDAY, OCTOBER
15.00.33 JOB07309  $HASP373 QASGSR1C STARTED - INIT 1    - CLASS A - SYS DSYS
15.00.35 JOB07309  -                                               --TIMINGS (M
15.00.35 JOB07309  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB
15.00.35 JOB07309  -QASGSR1C COMPILE  TRANSTEP    00     84    226    .00    .00
15.00.49 JOB07309  -QASGSR1C COMPILE  COBLSTEP    04   1307   1217    .04    .00
15.00.54 JOB07309  -QASGSR1C COMPILE  LINKSTEP    00    278    708    .01    .00
15.00.54 JOB07309  -QASGSR1C ENDED.  NAME-                      TOTAL TCB CPU TIM
15.00.54 JOB07309  $HASP395 QASGSR1C ENDED
------ JES2 JOB STATISTICS ------
  08 OCT 2001 JOB EXECUTION DATE
           34 CARDS READ
       10,247 SYSOUT PRINT RECORDS
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 192. JES2 Job Log

Scroll to the bottom of the JES2 JOB LOG to see the *Error Message Summary* report. To advance directly to the bottom of the JES2 JOB LOG type **M** on the Command Input line and press **F8**.



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB07309  DSID   104 LINE 288     COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> PAGE
MESSAGE SUMMARY REPORT
----------------------
 SEVERE MESSAGES        (SEVERITY = 12)
 NONE

 ERROR MESSAGES         (SEVERITY = 08)
 NONE

 WARNING MESSAGES       (SEVERITY = 04)
 NONE

 INFORMATIONAL MESSAGES (SEVERITY = 00)
 2008  2278  2322


 **** END OF MESSAGE SUMMARY REPORT ****
```

Figure 193. Error Message Summary report section of the JES2 JOB LOG

7. Review the summary report. If errors are found, correct them.
8. Press **F3 END** to return to the output listings and select another to view.

Defining the adapter resources to CICS

After deploying an adapter to the OS/390 server, you need to define the adapter resources (programs and transaction ids) to CICS. You will need to do this each time a new adapter is deployed.

You also must NEWCOPY any server adapter programs that were modified.

Using the CEDA transaction, perform the following tasks:
- Define the adapter programs to CICS
- Define the adapter transactions to CICS
- Install the adapter programs to CICS
- Install the adapter transactions to CICS

Using the CEMT transaction, perform the following tasks:
- Validate the association of installed programs to installed transactions
- NEWCOPY the adapter programs

Running the CEDA transaction

The following sections provide the instructions on running the CEDA transaction to define and install adapter resources to CICS.

Define the adapter programs to CICS: You should have the names of the adapter programs and transactions available before running the CEDA transaction. The names of the programs and their associated transaction identifiers are listed in the ispec file(s). The ispec file(s) that you used for your adapter is located in the following directory:

```
C:\<mqiac_base>\cics
```

Perform the following steps to define adapter programs to CICS.

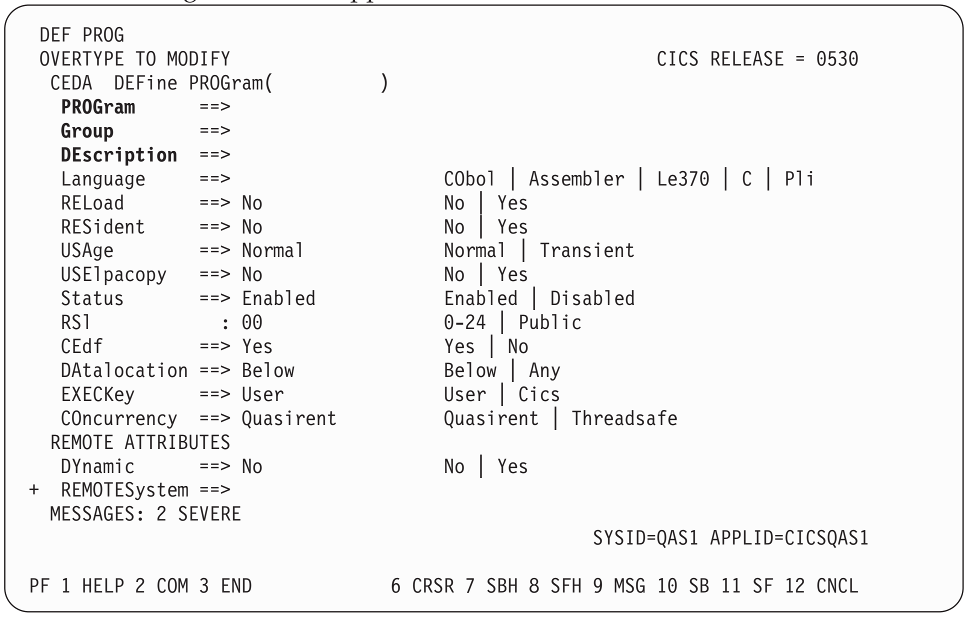
1. Access a command line in CICS.

2. Type the following command:

   `ceda def prog`

3. Press **Enter**.

   The Define Program screen appears:



```
 DEF PROG
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
  CEDA  DEFine PROGram(          )
   PROGram      ==>
   Group        ==>
   DEscription  ==>
   Language     ==>                    CObol | Assembler | Le370 | C | Pli
   RELoad       ==> No                 No | Yes
   RESident     ==> No                 No | Yes
   USAge        ==> Normal             Normal | Transient
   USElpacopy   ==> No                 No | Yes
   Status       ==> Enabled            Enabled | Disabled
   RSl            : 00                 0-24 | Public
   CEdf         ==> Yes                Yes | No
   DAtalocation ==> Below              Below | Any
   EXECKey      ==> User               User | Cics
   COncurrency  ==> Quasirent          Quasirent | Threadsafe
  REMOTE ATTRIBUTES
   DYnamic      ==> No                 No | Yes
+  REMOTESystem ==>
  MESSAGES: 2 SEVERE
                                                    SYSID=QAS1 APPLID=CICSQAS1

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Type values for the **PROGram**, **Group** and **Transid** (Use **F8 — Scroll Forward** to access the transaction id field).

Table 73. Values for the Define Transactions screen

Program	Group	Transid
TUNAV1	MIACUSER	TUDN
TUDPL1	MIACUSER	TUD1

5. Press **Enter** to define the program to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.

6. Press **F3 END** to return to the define program command line.

   Repeat the CEDA define program function until each adapter program is defined to CICS.

Define the adapter program transactions to CICS: Perform the following steps to define adapter programs to CICS:

1. Access a command line in CICS.
2. Type the following command:

   `ceda def trans`

3. Press **Enter**.

   The Define Transaction screen appears:

```
 DEF TRANS
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
  CEDA  DEFine TRANSaction(      )
   TRANSaction  ==>
   Group        ==> DEV1
   DEscription  ==>
   PROGram      ==>
   TWasize      ==> 00000            0-32767
   PROFile      ==> DFHCICST
   PArtitionset ==>
   STAtus       ==> Enabled          Enabled | Disabled
   PRIMedsize     : 00000            0-65520
   TASKDATALoc  ==> Below            Below | Any
   TASKDATAKey  ==> User             User | Cics
   STOrageclear ==> No               No | Yes
   RUnaway      ==> System           System | 0 | 500-2700000
   SHutdown     ==> Disabled         Disabled | Enabled
   ISolate      ==> Yes              Yes | No
   Brexit       ==>
+ REMOTE ATTRIBUTES
  S An object must be specified.
                                                    SYSID=QAS1 APPLID=CICSQAS1
```

4. Type values for the **TRANSaction**, **Group**, **DEscription** and **PROGram**.
5. Press **Enter** to define the transaction to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.
6. Press **F3 END** to return to the define transaction command line.

   Repeat the CEDA define transaction function until each adapter program transaction is defined to CICS.

Install the adapter programs to CICS: Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) prog(y)
   ```

   Where x = the group name assigned and y = the program being installed.

   Note: You can choose to run the install command on the entire group by typing **ceda inst gr(x)** on the command line (where x = the name of the group). This command automatically installs all of the adapter programs and their transactions to CICS. When you install programs to CICS in this manner, **you will receive an error even though the programs and transactions were successfully installed**. To verify that the programs and their associated transactions were installed, go to a CICS command line and type the following command:

   ```
   CEMT I TRANS
   ```

   The screen shows the programs and the transactions installed for the group specified in the command.
3. Press **Enter** to install the program.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen:

```
 INST GR(MIACUSER) PROG(TUMQ01G)
 OVERTYPE TO MODIFY
  CEDA  Install
   Connection   ==>
   DB2Conn      ==>
   DB2Entry     ==>
   DB2Tran      ==>
   DOctemplate  ==>
   Enqmodel     ==>
   File         ==>
   Journalmodel ==>
   LSrpool      ==>
   Mapset       ==>
   PARTItionset ==>
   PARTNer      ==>
   PROCesstype  ==>
   PROFile      ==>
   PROGram      ==> TUMQ01G
   Requestmodel ==>
+  Sessions     ==>

                                                    SYSID=QAS1 APPLID=CICSQAS1
  INSTALL SUCCESSFUL                          TIME:  16.24.37  DATE: 01.275
PF 1 HELP      3 END             6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Select **F3 END** to return to the CEDA INST command line.
5. Edit the command line to indicate the next program to install and select **Enter** to install that program.

   Repeat the process until you have installed all the programs to CICS.

Install the adapter program transactions to CICS: Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) trans(y)
   ```

   Where x = the group name assigned and y = the program transaction being installed.
3. Press **Enter** to install the transaction.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen.
4. Select **F3 END** to return to the CEDA INST command line.
5. Edit the command line to indicate the next program transaction to install and select **Enter** to install that transaction.

   Repeat the process until you have installed all the program transactions to CICS.

If you want to see that the programs and transactions are associated with each other, go to a CICS command line and type the following command:

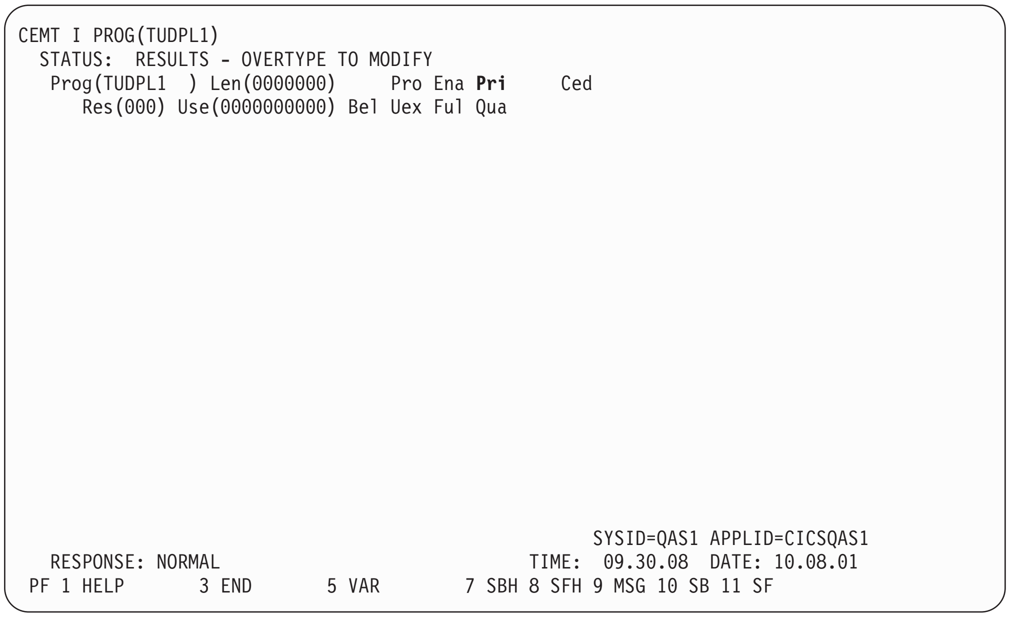
```
CEMT I TRANS
```

NEWCOPY the adapter programs to CICS: From the CICS command line type the following:

```
CEMT I PROG(x)
```

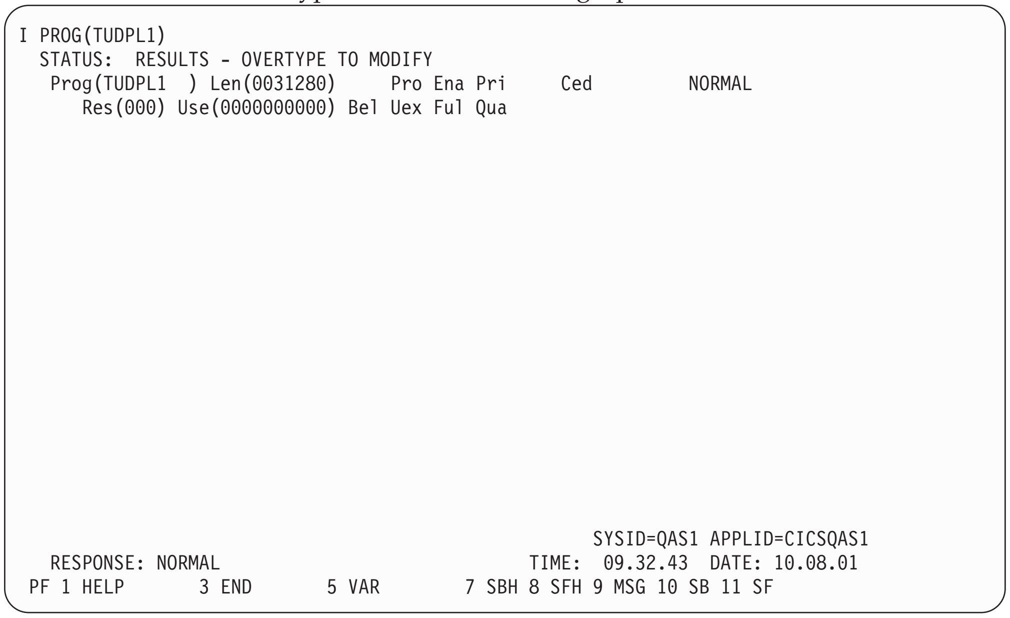
Where x = the installed adapter program.

The following screen appears:



```
CEMT I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0000000)     Pro Ena Pri     Ced
      Res(000) Use(0000000000) Bel Uex Ful Qua





















                                                      SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.30.08  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Tab to the Pri field and type **New**. The following update occurs:



```
I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0031280)     Pro Ena Pri     Ced         NORMAL
      Res(000) Use(0000000000) Bel Uex Ful Qua





















                                                      SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.32.43  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

See Chapter 6, “Validating the adapters” on page 231 for how to run and validate the adapter.

Defining MQ adapter resources to CICS

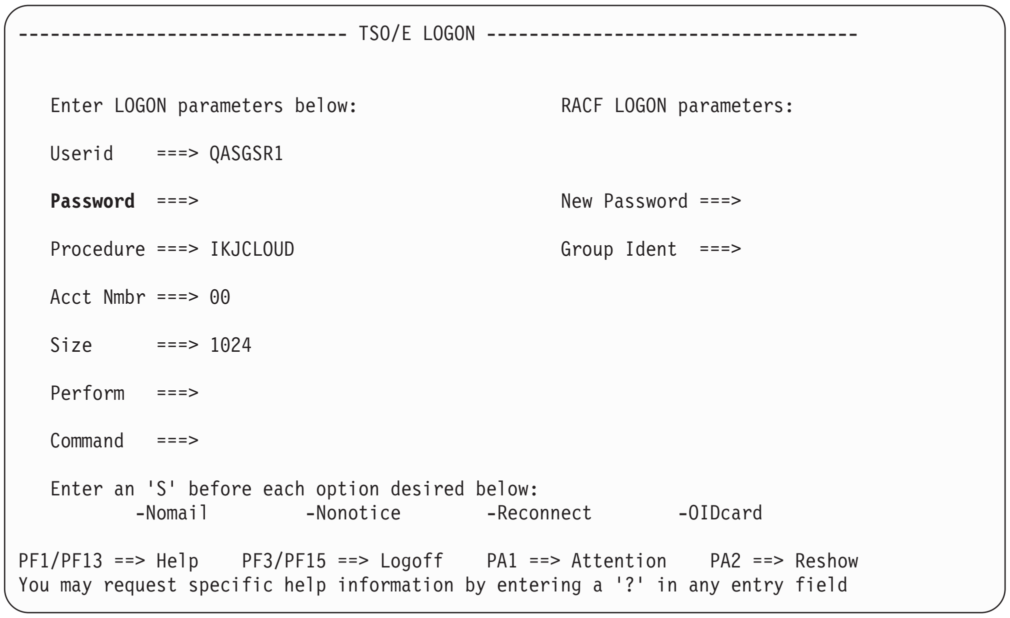
Check to see that the adapter compiled in CICS

After you have deployed the adapter to the OS/390 server, you need to make sure that it compiled with no errors.

Perform the following steps to check the compile:

1. Sign on to the OS/390 server and select the option to access TSO.

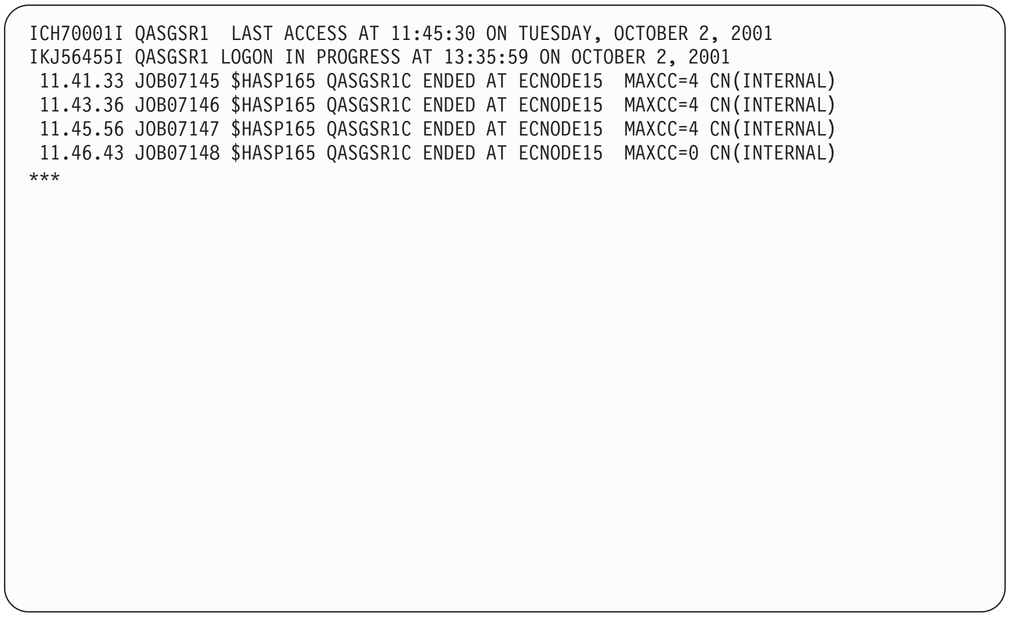
2. Enter your TSO logon password and select **Enter**.



```
------------------------------- TSO/E LOGON -----------------------------------


   Enter LOGON parameters below:                 RACF LOGON parameters:

   Userid    ===> QASGSR1

   Password  ===>                                New Password ===>

   Procedure ===> IKJCLOUD                       Group Ident  ===>

   Acct Nmbr ===> 00

   Size      ===> 1024

   Perform   ===>

   Command   ===>

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Figure 194. TSO/E logon screen

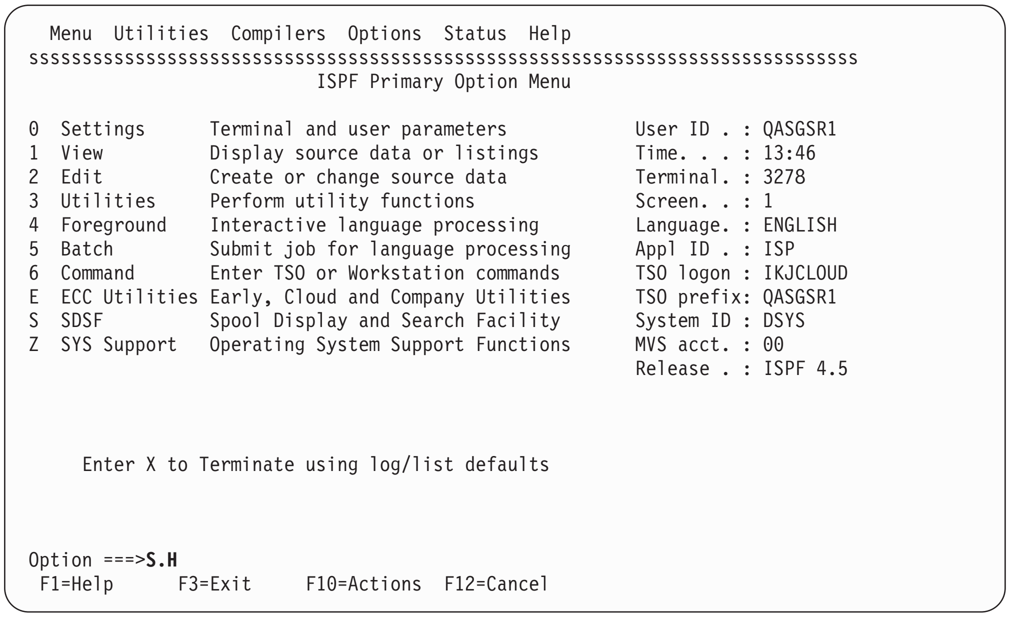
3. Press **Enter** to complete signon.



```
ICH70001I QASGSR1  LAST ACCESS AT 11:45:30 ON TUESDAY, OCTOBER 2, 2001
IKJ56455I QASGSR1 LOGON IN PROGRESS AT 13:35:59 ON OCTOBER 2, 2001
 11.41.33 JOB07145 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 11.43.36 JOB07146 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 11.45.56 JOB07147 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 11.46.43 JOB07148 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=0 CN(INTERNAL)
***
```

Figure 195. List of completed job notifications sent to OS/390 server (for active user id) via the deploy process

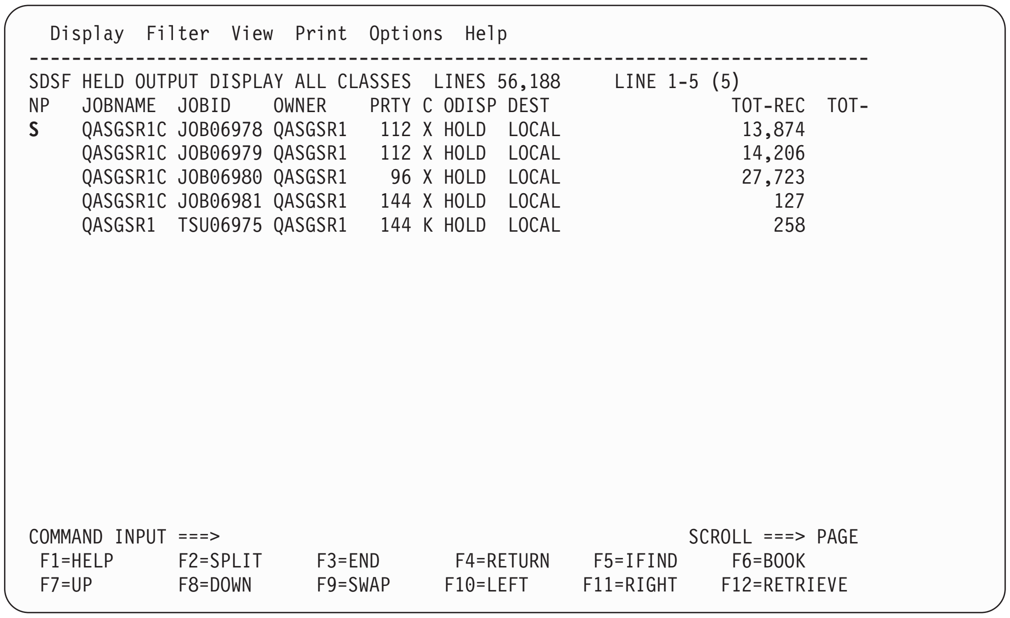
4. Press **Pause** to go to the *ISPF Primary Option Menu*



```
  Menu  Utilities  Compilers  Options  Status  Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                           ISPF Primary Option Menu

0  Settings      Terminal and user parameters            User ID . : QASGSR1
1  View          Display source data or listings         Time. . . : 13:46
2  Edit          Create or change source data            Terminal. : 3278
3  Utilities     Perform utility functions               Screen. . : 1
4  Foreground    Interactive language processing         Language. : ENGLISH
5  Batch         Submit job for language processing      Appl ID . : ISP
6  Command       Enter TSO or Workstation commands       TSO logon : IKJCLOUD
E  ECC Utilities Early, Cloud and Company Utilities      TSO prefix: QASGSR1
S  SDSF          Spool Display and Search Facility       System ID : DSYS
Z  SYS Support   Operating System Support Functions      MVS acct. : 00
                                                         Release . : ISPF 4.5




     Enter X to Terminate using log/list defaults




Option ===>S.H
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Figure 196. ISPF Primary Option Menu

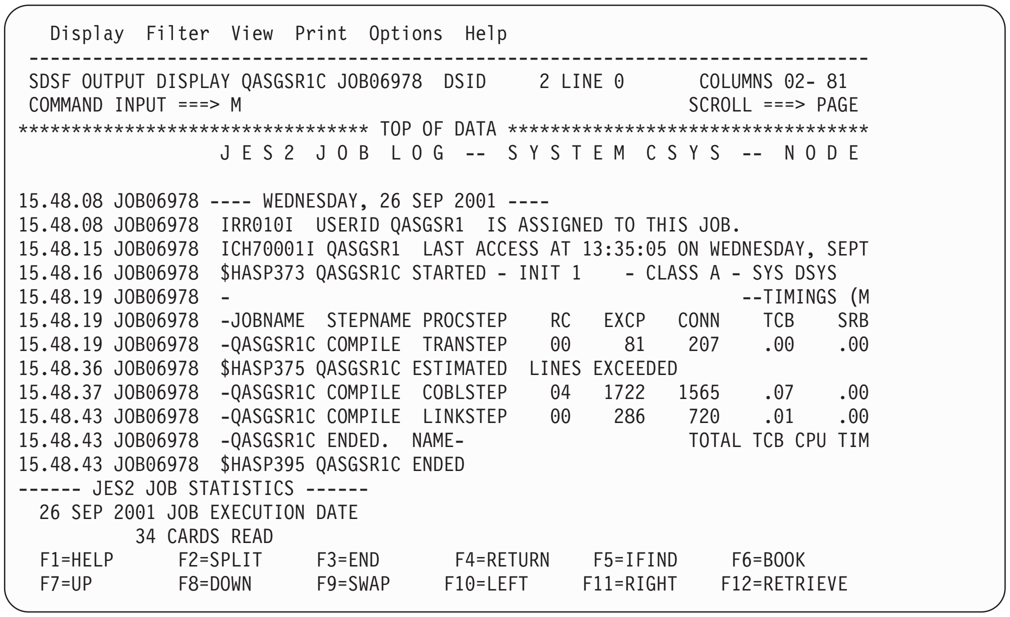
5. Type **S.H** on the Option line and press **Enter** to see go to the compile job listing:



```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF HELD OUTPUT DISPLAY ALL CLASSES  LINES 56,188     LINE 1-5 (5)
NP   JOBNAME  JOBID    OWNER    PRTY C ODISP DEST               TOT-REC  TOT-
S    QASGSR1C JOB06978 QASGSR1   112 X HOLD  LOCAL               13,874
     QASGSR1C JOB06979 QASGSR1   112 X HOLD  LOCAL               14,206
     QASGSR1C JOB06980 QASGSR1    96 X HOLD  LOCAL               27,723
     QASGSR1C JOB06981 QASGSR1   144 X HOLD  LOCAL                  127
     QASGSR1  TSU06975 QASGSR1   144 K HOLD  LOCAL                  258














COMMAND INPUT ===>                                            SCROLL ===> PAGE
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 197. Spool Display and Search Facility Held Output Display screen

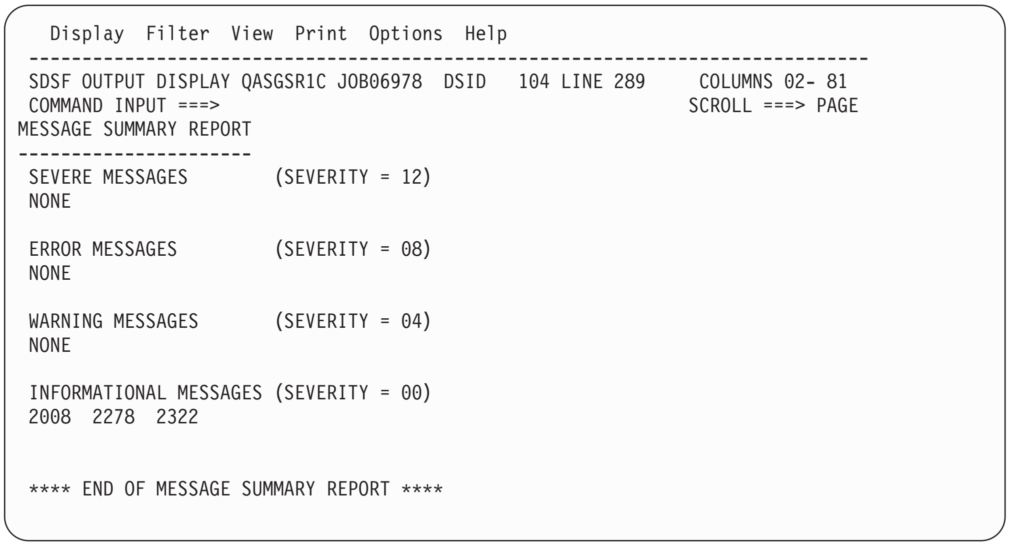
6. To view a job type an **S** to the left of its job name and press **Enter**. This will open (for view purposes) the JES2 JOB LOG:



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB06978  DSID     2 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===> M                                          SCROLL ===> PAGE
********************************* TOP OF DATA **********************************
                   J E S 2  J O B  L O G -- S Y S T E M  C S Y S -- N O D E

15.48.08 JOB06978 ---- WEDNESDAY, 26 SEP 2001 ----
15.48.08 JOB06978  IRR010I  USERID QASGSR1  IS ASSIGNED TO THIS JOB.
15.48.15 JOB06978  ICH70001I QASGSR1  LAST ACCESS AT 13:35:05 ON WEDNESDAY, SEPT
15.48.16 JOB06978  $HASP373 QASGSR1C STARTED - INIT 1    - CLASS A - SYS DSYS
15.48.19 JOB06978  -                                                --TIMINGS (M
15.48.19 JOB06978  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB
15.48.19 JOB06978  -QASGSR1C COMPILE  TRANSTEP    00     81    207    .00    .00
15.48.36 JOB06978  $HASP375 QASGSR1C ESTIMATED  LINES EXCEEDED
15.48.37 JOB06978  -QASGSR1C COMPILE  COBLSTEP    04   1722   1565    .07    .00
15.48.43 JOB06978  -QASGSR1C COMPILE  LINKSTEP    00    286    720    .01    .00
15.48.43 JOB06978  -QASGSR1C ENDED.  NAME-                     TOTAL TCB CPU TIM
15.48.43 JOB06978  $HASP395 QASGSR1C ENDED
------ JES2 JOB STATISTICS ------
  26 SEP 2001 JOB EXECUTION DATE
           34 CARDS READ
  F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
  F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 198. JES2 Job Log

Scroll to the bottom of the JES2 JOB LOG to see the *Error Message Summary* report. To advance directly to the bottom of the JES2 JOB LOG type **M** on the Command Input line and press **F8**.



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB06978  DSID   104 LINE 289     COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> PAGE
MESSAGE SUMMARY REPORT
----------------------
 SEVERE MESSAGES        (SEVERITY = 12)
 NONE

 ERROR MESSAGES         (SEVERITY = 08)
 NONE

 WARNING MESSAGES       (SEVERITY = 04)
 NONE

 INFORMATIONAL MESSAGES (SEVERITY = 00)
 2008  2278  2322


 **** END OF MESSAGE SUMMARY REPORT ****
```

Figure 199. Error Message Summary report section of the JES2 JOB LOG

7. Review the summary report. If errors are found, correct them.
8. Press **F3 END** to return to the list of output listings and select another to view.

Defining the adapter resources to CICS

After deploying an adapter to the OS/390 server, you need to define the adapter resources (programs and transaction ids) to CICS. You will need to do this each time a new adapter is deployed.

You also must NEWCOPY any server adapter programs that were modified.

Using the CEDA transaction, perform the following tasks:
- Define the adapter programs to CICS.
- Define the adapter transactions to CICS.
- Install the adapter programs to CICS.
- Install the adapter transactions to CICS.

Using the CEMT transaction, perform the following tasks:
- Validate the association of installed programs to installed transactions
- NEWCOPY the adapter programs

Running the CEDA transaction

The following sections provide the instructions on running the CEDA transaction to define and install adapter resources to CICS.

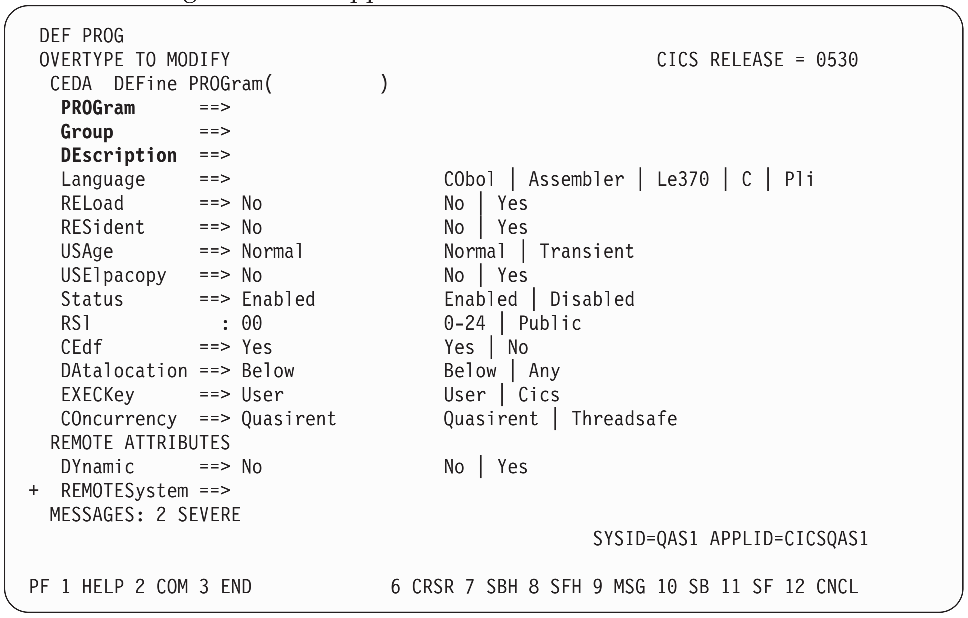
Define the adapter programs to CICS: You should have the names of the adapter programs and transactions available before running the CEDA transaction. The names of the programs and their associated transaction identifiers are listed in the ispec file(s). The ispec file(s) that you used for your adapter is located in the following directory:

```
C:\<mqiac_base>\cics
```

Perform the following steps to define adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:
   ```
   ceda def prog
   ```
3. Press **Enter**.

   The Define Program screen appears:



```
  DEF PROG
  OVERTYPE TO MODIFY                                       CICS RELEASE = 0530
   CEDA  DEFine PROGram(         )
    PROGram      ==>
    Group        ==>
    DEscription  ==>
    Language     ==>                    COboI | Assembler | Le370 | C | Pli
    RELoad       ==> No                 No | Yes
    RESident     ==> No                 No | Yes
    USAge        ==> Normal             Normal | Transient
    USElpacopy   ==> No                 No | Yes
    Status       ==> Enabled            Enabled | Disabled
    RSl            : 00                 0-24 | Public
    CEdf         ==> Yes                Yes | No
    DAtalocation ==> Below              Below | Any
    EXECKey      ==> User               User | Cics
    COncurrency  ==> Quasirent          Quasirent | Threadsafe
   REMOTE ATTRIBUTES
    DYnamic      ==> No                 No | Yes
 +  REMOTESystem ==>
   MESSAGES: 2 SEVERE
                                                    SYSID=QAS1 APPLID=CICSQAS1

 PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Type values for the **PROGram**, **Group** and **Transid** (Use F8 — scroll forward to access the transaction id field).
5. Press **Enter** to define the program to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.

6. Press **F3 END** to return to the define program command line.

   Repeat the CEDA define program function until each adapter program is defined to CICS.

Define the adapter program transactions to CICS: Perform the following steps to define adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda def trans
   ```
3. Press **Enter**.

   The Define Transaction screen appears:

```
 DEF TRANS
 OVERTYPE TO MODIFY                                         CICS RELEASE = 0530
  CEDA  DEFine TRANSaction(      )
   TRANSaction  ==>
   Group        ==> DEV1
   DEscription  ==>
   PROGram      ==>
   TWasize      ==> 00000             0-32767
   PROFile      ==> DFHCICST
   PArtitionset ==>
   STAtus       ==> Enabled           Enabled | Disabled
   PRIMedsize     : 00000             0-65520
   TASKDATALoc  ==> Below             Below | Any
   TASKDATAKey  ==> User              User | Cics
   STOrageclear ==> No                No | Yes
   RUnaway      ==> System            System | 0 | 500-2700000
   SHutdown     ==> Disabled          Disabled | Enabled
   ISolate      ==> Yes               Yes | No
   Brexit       ==>
+ REMOTE ATTRIBUTES
  S An object must be specified.
                                                      SYSID=QAS1 APPLID=CICSQAS1
```

4. Type values for the **TRANSaction**, **Group**, **DEscription** and **PROGram**

Table 74. Values for the Define Transactions screen

Program	Group	Transid
TUMNAV1	MIACUSER	TUM1
TUMQ01P	MIACUSER	TUMP
TUMQ01G	MIACUSER	TUMG

5. Press **Enter** to define the transaction to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.
6. Press **F3 END** to return to the define transaction command line.

   Repeat the CEDA define transaction function until each adapter program transaction is defined to CICS.

Install the adapter programs to CICS: Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) prog(y)
   ```

   Where x = the group name assigned and y = the program being installed.

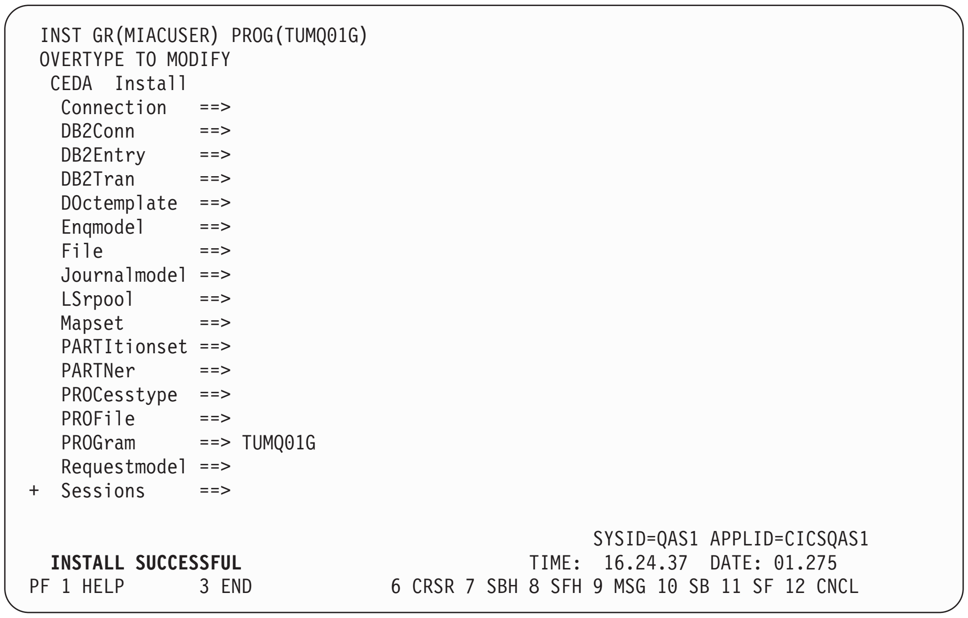
Note: You can choose to run the install command on the entire group by typing **ceda inst gr(x)** on the command line (where x = the name of the group). This command automatically installs all of the adapter programs and their transactions to CICS. When you install programs to CICS in this manner, **you will receive an error even though the programs and transactions were successfully installed**. To verify that the programs and their associated transactions were installed, go to a CICS command line and type the following command:

```
CEMT I TRANS
```

The screen shows the programs and the transactions installed for the group specified in the command.

3. Press **Enter** to install the program.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen:



```
 INST GR(MIACUSER) PROG(TUMQ01G)
 OVERTYPE TO MODIFY
  CEDA  Install
   Connection   ==>
   DB2Conn      ==>
   DB2Entry     ==>
   DB2Tran      ==>
   DOctemplate  ==>
   Enqmodel     ==>
   File         ==>
   Journalmodel ==>
   LSrpool      ==>
   Mapset       ==>
   PARTItionset ==>
   PARTNer      ==>
   PROCesstype  ==>
   PROFile      ==>
   PROGram      ==> TUMQ01G
   Requestmodel ==>
+  Sessions     ==>

                                                     SYSID=QAS1 APPLID=CICSQAS1
  INSTALL SUCCESSFUL                           TIME:  16.24.37  DATE: 01.275
PF 1 HELP      3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Select **F3 END** to return to the CEDA INST command line.
5. Edit the command line to indicate the next program to install and select **Enter** to install that program.

   Repeat the process until you have installed all the programs to CICS.

Install the adapter program transactions to CICS: Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) trans(y)
   ```

   Where x = the group name assigned and y = the program transaction being installed.
3. Press **Enter** to install the transaction.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen.
4. Select **F3 END** to return to the CEDA INST command line.

5. Edit the command line to indicate the next program transaction to install and select **Enter** to install that transaction.

   Repeat the process until you have installed all the program transactions to CICS.

If you want to see that the programs and transactions are associated with each other, go to a CICS command line and type the following command:

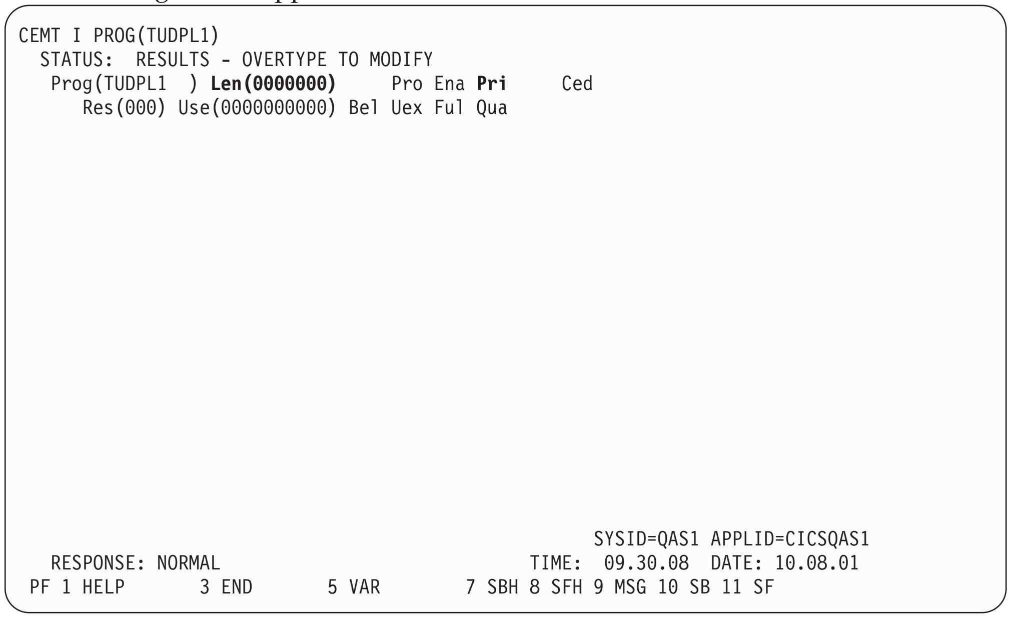
```
CEMT I TRANS
```

NEWCOPY the adapter programs to CICS: From the CICS command line type the following:

```
CEMT I PROG(x,y,z)
```

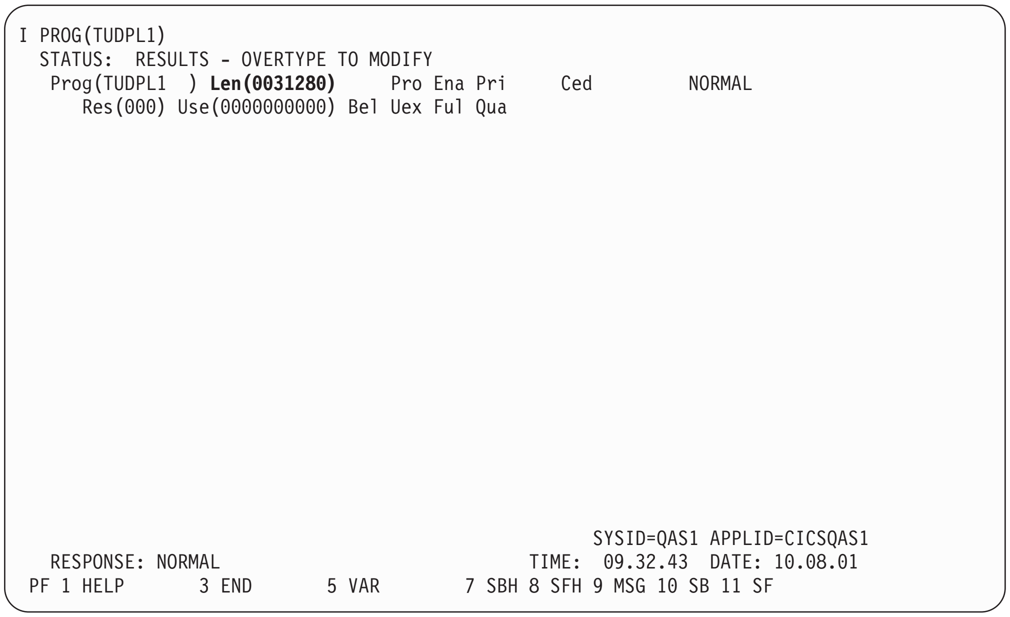
Where x,y, z = the installed adapter programs.

The following screen appears:



```
CEMT I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0000000)     Pro Ena Pri     Ced
      Res(000) Use(0000000000) Bel Uex Ful Qua






















                                                      SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.30.08  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR       7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Tab so that the cursor follows the **Pri** field and type **New**. The following update occurs:



```
I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0031280)     Pro Ena Pri     Ced         NORMAL
      Res(000) Use(0000000000) Bel Uex Ful Qua







                                                       SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.32.43  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

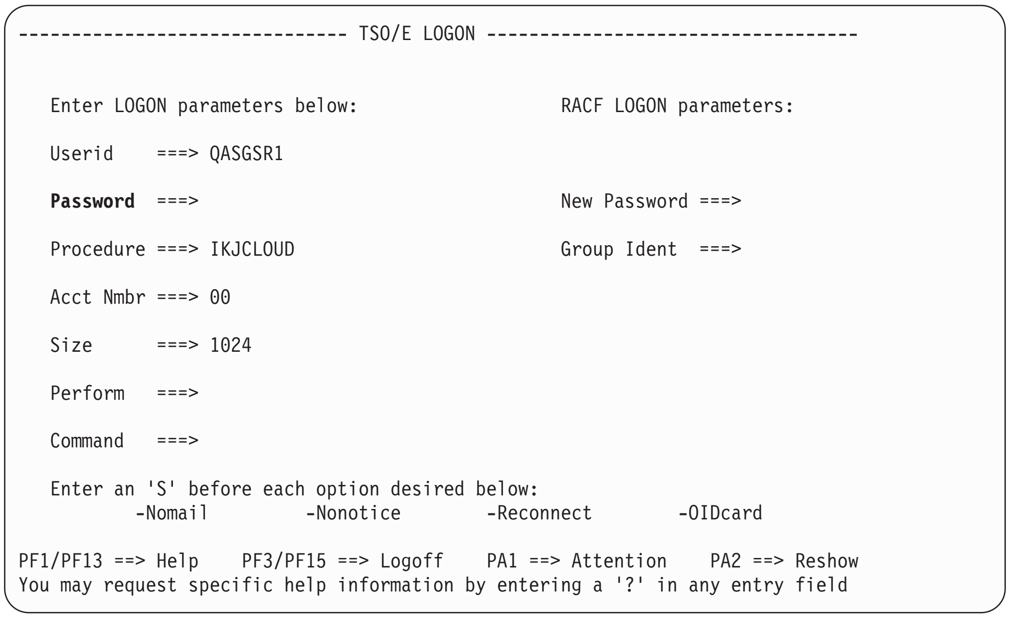
See Chapter 6, “Validating the adapters” on page 231 for how to run and validate the adapter.

Defining FEPI adapter resources to CICS

Check to see that the adapter compiled in CICS

Perform the following steps to check the compile:

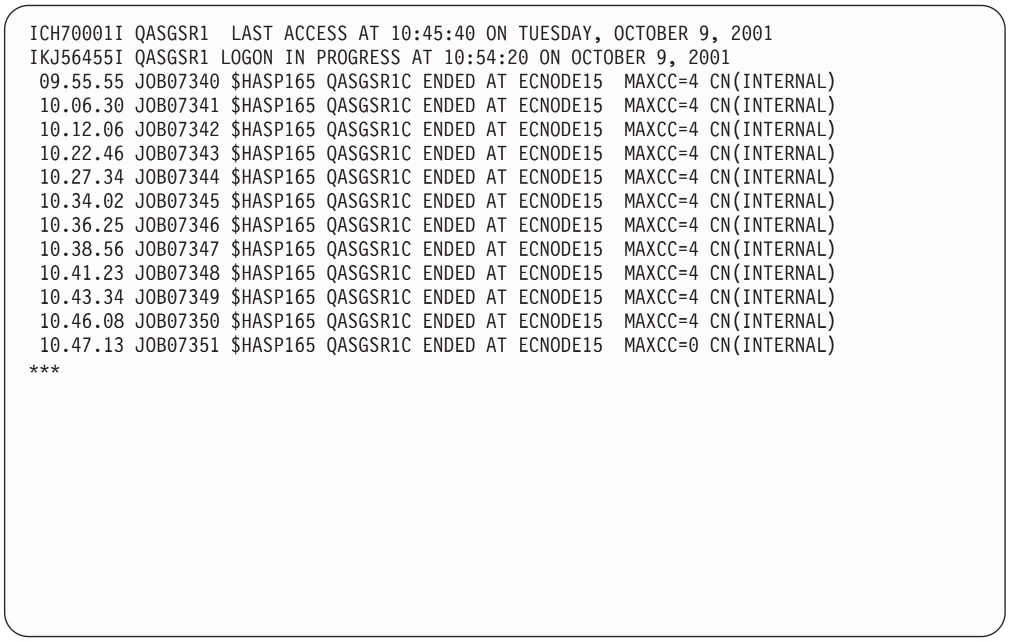
1. Sign on to the OS/390 server and select the option to access TSO.
2. Enter your TSO logon password and select **Enter**.



```
------------------------------- TSO/E LOGON -----------------------------------


   Enter LOGON parameters below:                   RACF LOGON parameters:

   Userid    ===> QASGSR1

   Password  ===>                                  New Password ===>

   Procedure ===> IKJCLOUD                         Group Ident  ===>

   Acct Nmbr ===> 00

   Size      ===> 1024

   Perform   ===>

   Command   ===>

   Enter an 'S' before each option desired below:
           -Nomail         -Nonotice        -Reconnect        -OIDcard

PF1/PF13 ==> Help    PF3/PF15 ==> Logoff    PA1 ==> Attention    PA2 ==> Reshow
You may request specific help information by entering a '?' in any entry field
```

Figure 200. TSO/E logon screen

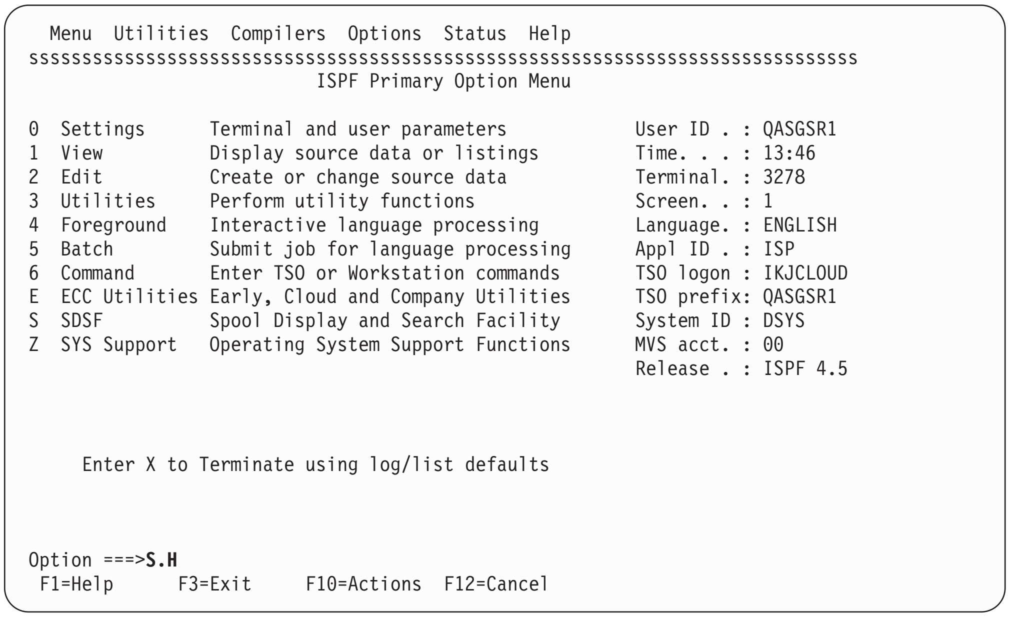
3. Press **Enter** to complete signon.



```
ICH70001I QASGSR1  LAST ACCESS AT 10:45:40 ON TUESDAY, OCTOBER 9, 2001
IKJ56455I QASGSR1 LOGON IN PROGRESS AT 10:54:20 ON OCTOBER 9, 2001
 09.55.55 JOB07340 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.06.30 JOB07341 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.12.06 JOB07342 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.22.46 JOB07343 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.27.34 JOB07344 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.34.02 JOB07345 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.36.25 JOB07346 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.38.56 JOB07347 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.41.23 JOB07348 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.43.34 JOB07349 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.46.08 JOB07350 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=4 CN(INTERNAL)
 10.47.13 JOB07351 $HASP165 QASGSR1C ENDED AT ECNODE15  MAXCC=0 CN(INTERNAL)
***
```

Figure 201. List of completed job notifications sent to the OS/390 server (for active user id) via the deploy process

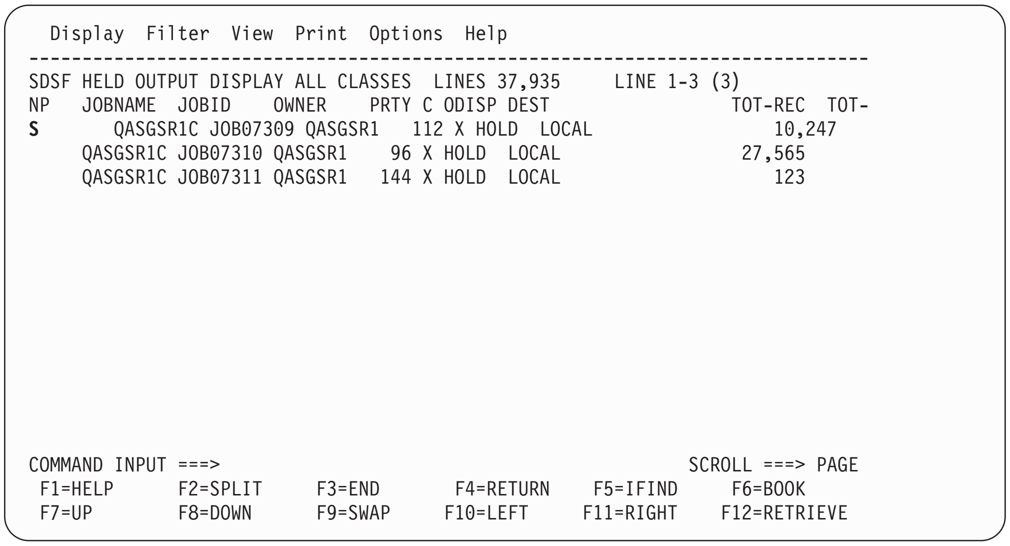
4. Press **Pause** to go to the *ISPF Primary Option Menu*.



```
  Menu  Utilities  Compilers  Options  Status  Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
                           ISPF Primary Option Menu

0  Settings      Terminal and user parameters            User ID . : QASGSR1
1  View          Display source data or listings         Time. . . : 13:46
2  Edit          Create or change source data            Terminal. : 3278
3  Utilities     Perform utility functions               Screen. . : 1
4  Foreground    Interactive language processing         Language. : ENGLISH
5  Batch         Submit job for language processing      Appl ID . : ISP
6  Command       Enter TSO or Workstation commands       TSO logon : IKJCLOUD
E  ECC Utilities Early, Cloud and Company Utilities      TSO prefix: QASGSR1
S  SDSF          Spool Display and Search Facility       System ID : DSYS
Z  SYS Support   Operating System Support Functions      MVS acct. : 00
                                                         Release . : ISPF 4.5



     Enter X to Terminate using log/list defaults



Option ===>S.H
 F1=Help      F3=Exit     F10=Actions  F12=Cancel
```

Figure 202. ISPF Primary Option Menu

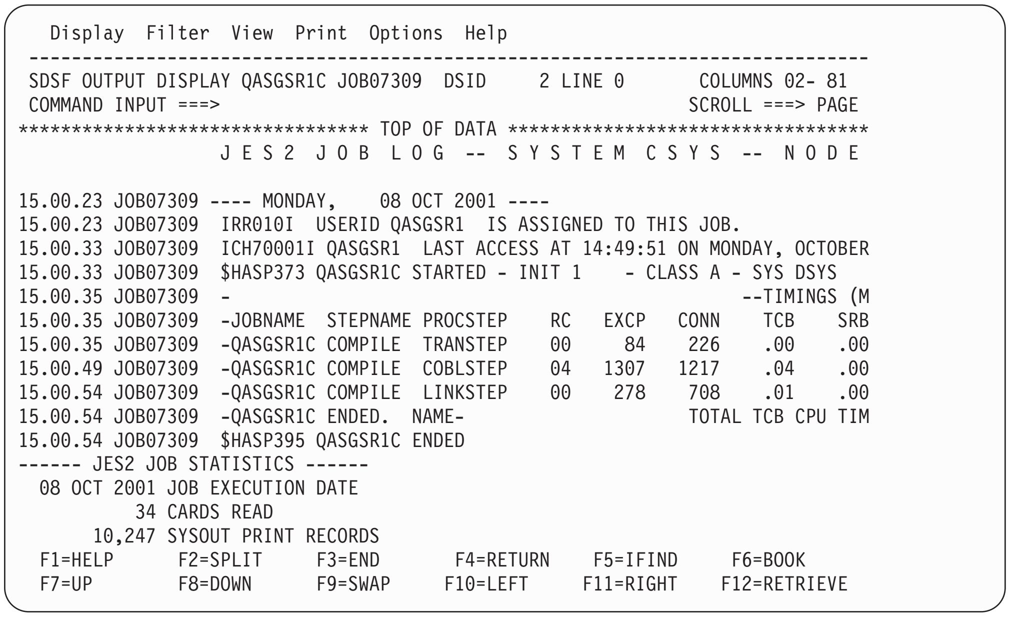
5. Type **S.H** on the Option line and press **Enter** to see go to the compile job listing:



```
  Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF HELD OUTPUT DISPLAY ALL CLASSES  LINES 37,935     LINE 1-3 (3)
NP   JOBNAME  JOBID    OWNER    PRTY C ODISP DEST               TOT-REC  TOT-
S       QASGSR1C JOB07309 QASGSR1   112 X HOLD  LOCAL               10,247
     QASGSR1C JOB07310 QASGSR1    96 X HOLD  LOCAL               27,565
     QASGSR1C JOB07311 QASGSR1   144 X HOLD  LOCAL                  123















COMMAND INPUT ===>                                              SCROLL ===> PAGE
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 203. Spool Display and Search Facility Held Output Display screen

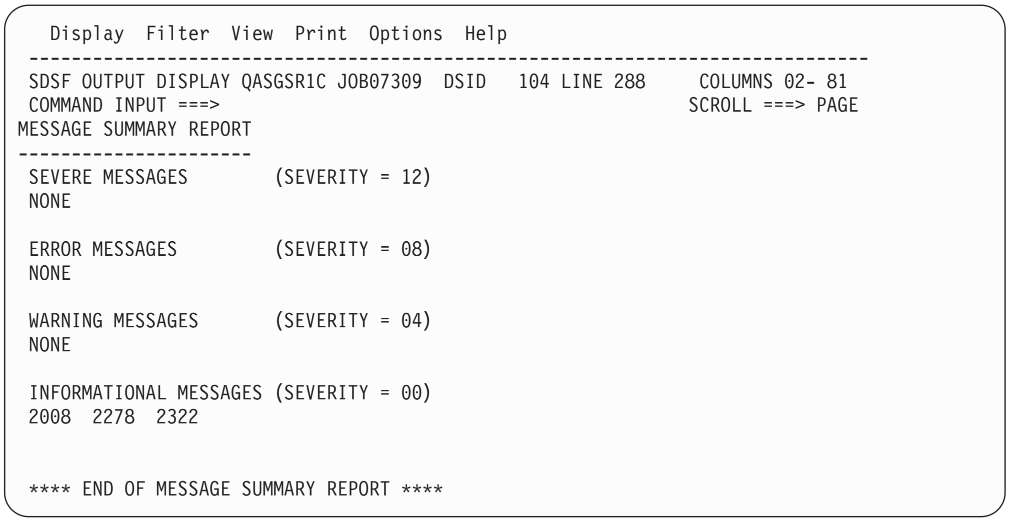
6. To view an output listing an **S** to the left of its name and press **Enter**. This will open (for view purposes) the JES2 JOB LOG:



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB07309  DSID     2 LINE 0       COLUMNS 02- 81
 COMMAND INPUT ===>                                             SCROLL ===> PAGE
********************************* TOP OF DATA **********************************
                   J E S 2  J O B  L O G  --  S Y S T E M  C S Y S  --  N O D E

15.00.23 JOB07309 ---- MONDAY,    08 OCT 2001 ----
15.00.23 JOB07309  IRR010I  USERID QASGSR1  IS ASSIGNED TO THIS JOB.
15.00.33 JOB07309  ICH70001I QASGSR1  LAST ACCESS AT 14:49:51 ON MONDAY, OCTOBER
15.00.33 JOB07309  $HASP373 QASGSR1C STARTED - INIT 1    - CLASS A - SYS DSYS
15.00.35 JOB07309  -                                                --TIMINGS (M
15.00.35 JOB07309  -JOBNAME  STEPNAME PROCSTEP    RC   EXCP   CONN    TCB    SRB
15.00.35 JOB07309  -QASGSR1C COMPILE  TRANSTEP    00     84    226    .00    .00
15.00.49 JOB07309  -QASGSR1C COMPILE  COBLSTEP    04   1307   1217    .04    .00
15.00.54 JOB07309  -QASGSR1C COMPILE  LINKSTEP    00    278    708    .01    .00
15.00.54 JOB07309  -QASGSR1C ENDED.  NAME-                     TOTAL TCB CPU TIM
15.00.54 JOB07309  $HASP395 QASGSR1C ENDED
------ JES2 JOB STATISTICS ------
  08 OCT 2001 JOB EXECUTION DATE
           34 CARDS READ
       10,247 SYSOUT PRINT RECORDS
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=IFIND     F6=BOOK
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 204. JES2 Job Log

Scroll to the bottom of the JES2 JOB LOG to see the *Error Message Summary* report. To advance directly to the bottom of the JES2 JOB LOG type **M** on the Command Input line and press **F8**.



```
  Display  Filter  View  Print  Options  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY QASGSR1C JOB07309  DSID   104 LINE 288     COLUMNS 02- 81
 COMMAND INPUT ===>                                            SCROLL ===> PAGE
MESSAGE SUMMARY REPORT
----------------------
 SEVERE MESSAGES        (SEVERITY = 12)
 NONE

 ERROR MESSAGES         (SEVERITY = 08)
 NONE

 WARNING MESSAGES       (SEVERITY = 04)
 NONE

 INFORMATIONAL MESSAGES (SEVERITY = 00)
 2008  2278  2322


 **** END OF MESSAGE SUMMARY REPORT ****
```

Figure 205. Error Message Summary report section of the JES2 JOB LOG

7. Review the summary report. If errors are found, correct them.
8. Press **F3 END** to return to the list of output listings and select another to view.

Defining the adapter resources to CICS

After deploying an adapter to the OS/390 server, you need to define the adapter resources (programs and transaction ids) to CICS. You will need to do this each time a new adapter is deployed.

You also must NEWCOPY any server adapter programs that were modified.

Using the CEDA transaction, perform the following tasks:
- Define the adapter programs to CICS.
- Define the adapter transactions to CICS.
- Install the adapter programs to CICS.
- Install the adapter transactions to CICS.

Using the CEMT transaction, perform the following tasks:
- Validate the association of installed programs to installed transactions
- NEWCOPY the adapter programs

Running the CEDA transaction

The following sections provide the instructions on running the CEDA transaction to define and install adapter resources to CICS.

You should have the names of the adapter programs and transactions available before running the CEDA transaction. The names of the programs and their associated transaction identifiers are listed in the ispec file(s). The ispec file(s) that you used for your adapter is located in the following directory:

```
C:\program files\ibm mqseries integrator agent for cics\cics
```

Define the adapter programs to CICS

Perform the following steps to define adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:
   `ceda def prog`
3. Press **Enter**

   The Define Program screen appears:

```
 DEF PROG
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
  CEDA  DEFine PROGram(          )
   PROGram      ==>
   Group        ==>
   DEscription  ==>
   Language     ==>                  CObol | Assembler | Le370 | C | Pli
   RELoad       ==> No               No | Yes
   RESident     ==> No               No | Yes
   USAge        ==> Normal           Normal | Transient
   USElpacopy   ==> No               No | Yes
   Status       ==> Enabled          Enabled | Disabled
   RSl            : 00               0-24 | Public
   CEdf         ==> Yes              Yes | No
   DAtalocation ==> Below            Below | Any
   EXECKey      ==> User             User | Cics
   COncurrency  ==> Quasirent        Quasirent | Threadsafe
  REMOTE ATTRIBUTES
   DYnamic      ==> No               No | Yes
+  REMOTESystem ==>
  MESSAGES: 2 SEVERE
                                                     SYSID=QAS1 APPLID=CICSQAS1

PF 1 HELP 2 COM 3 END             6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Type values for the **PROGram**, **Group**, and **Transid** (you will need to scroll to the next page to enter a value for the transaction id).

Table 75. Values for the Define Transactions screen

Program	Group	Transid
TUFNAV	MIACUSER	TUF1
TUFSGON	MIACUSER	TUF2
TUFSGOFF	MIACUSER	TUF3
TUFPRSER	MIACUSER	TUF4
TUFRESET	MIACUSER	TUF5
TUFINQ	MIACUSER	TUF6

   Note: If you have modified the Specification files supplied with the FEPI tutorial, then you will need to modify the values used to define the values for the program, group and transid accordingly.

5. Press **Enter** to define the program to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.
6. Press **F3 END** to return to the define program command line.

   Repeat the CEDA define program function until each adapter program is defined to CICS.

Define the adapter program transactions to CICS

Perform the following steps to define adapter programs to CICS.

1. Access a command line in CICS.

2. Type the following command:

   ```
   ceda def trans
   ```
3. Press **Enter**

   The Define Transaction screen appears:

   ```
    DEF TRANS
    OVERTYPE TO MODIFY                                        CICS RELEASE = 0530
     CEDA  DEFine TRANSaction(      )
      TRANSaction  ==>
      Group        ==> DEV1
      DEscription  ==>
      PROGram      ==>
      TWasize      ==> 00000             0-32767
      PROFile      ==> DFHCICST
      PArtitionset ==>
      STAtus       ==> Enabled           Enabled | Disabled
      PRIMedsize     : 00000             0-65520
      TASKDATALoc  ==> Below             Below | Any
      TASKDATAKey  ==> User              User | Cics
      STOrageclear ==> No                No | Yes
      RUnaway      ==> System            System | 0 | 500-2700000
      SHutdown     ==> Disabled          Disabled | Enabled
      ISolate      ==> Yes               Yes | No
      Brexit       ==>
   + REMOTE ATTRIBUTES
     S An object must be specified.
                                                        SYSID=QAS1 APPLID=CICSQAS1
   ```
4. Type values for the **TRANSaction**, **Group**, **DEscription** and **PROGram**.
5. Press **Enter** to define the transaction to CICS.

   You should see the message DEFINE SUCCESSFUL at the bottom of the screen.
6. Press **F3 END** to return to the define transaction command line.

   Repeat the CEDA define transaction function until each adapter program transaction is defined to CICS.

Install the adapter programs to CICS

Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) prog(y)
   ```

   Where x = the group name assigned and y = the program being installed.

   Note: You can choose to run the install command on the entire group by typing **ceda inst gr(x)** on the command line (where x = the name of the group). This command automatically installs all of the adapter programs and their transactions to CICS. When you install programs to CICS in this manner, **you will receive an error even though the programs and transactions were successfully installed**. To verify that the programs and their associated transactions were installed, go to a CICS command line and type the following command:

   ```
   CEMT I TRANS
   ```

   The screen shows the programs and the transactions installed for the group specified in the command.
3. Press **Enter** to install the program.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen:

```
 INST GR(MIACUSER) PROG(TUMQ01G)
 OVERTYPE TO MODIFY
  CEDA  Install
   Connection   ==>
   DB2Conn      ==>
   DB2Entry     ==>
   DB2Tran      ==>
   DOctemplate  ==>
   Enqmodel     ==>
   File         ==>
   Journalmodel ==>
   LSrpool      ==>
   Mapset       ==>
   PARTItionset ==>
   PARTNer      ==>
   PROCesstype  ==>
   PROFile      ==>
   PROGram      ==> TUMQ01G
   Requestmodel ==>
+  Sessions     ==>

                                                   SYSID=QAS1 APPLID=CICSQAS1
  INSTALL SUCCESSFUL                         TIME:  16.24.37  DATE: 01.275
PF 1 HELP      3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

4. Select **F3 END** to return to the CEDA INST command line.
5. Edit the command line to indicate the next program to install and select **Enter** to install that program.

   Repeat the process until you have installed all the programs to CICS.

Install the adapter program transactions to CICS

Perform the following steps to install the adapter programs to CICS.

1. Access a command line in CICS.
2. Type the following command:

   ```
   ceda inst gr(x) trans(y)
   ```

   Where x = the group name assigned and y = the program transaction being installed.
3. Press **Enter** to install the transaction.

   You should see the message INSTALL SUCCESSFUL at the bottom of the screen.
4. Select **F3 END** to return to the CEDA INST command line.
5. Edit the command line to indicate the next program transaction to install and select **Enter** to install that transaction.

   Repeat the process until you have installed all the program transactions to CICS.

If you want to see that the programs and transactions are associated with each other, go to a CICS command line and type the following command:

```
CEMT I TRANS(TUF*)
```

This lists all the transactions that start with TUF.

NEWCOPY the adapter programs to CICS

From the CICS command line type the following:

```
CEMT I PROG(x)
```

Where x = the installed adapter program.

The following screen appears:

```
CEMT I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0000000)     Pro Ena Pri     Ced
      Res(000) Use(0000000000) Bel Uex Ful Qua









                                                       SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.30.08  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Tab to the Pri field and type **New**. The following update occurs:

```
I PROG(TUDPL1)
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Prog(TUDPL1  ) Len(0031280)     Pro Ena Pri     Ced         NORMAL
      Res(000) Use(0000000000) Bel Uex Ful Qua









                                                       SYSID=QAS1 APPLID=CICSQAS1
   RESPONSE: NORMAL                             TIME:  09.32.43  DATE: 10.08.01
 PF 1 HELP       3 END       5 VAR        7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

See Chapter 6, “Validating the adapters” on page 231 for how to run and validate the adapter.

Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Trademarks

CICS	MQSeries	SupportPac
CICSPlex	MVS	VTAM
e-business	OS/390	
IBM	RACF	

Glossary

The glossary contains *key* terms and their meanings as used in the information.

If a particular concept or term appears in one section only, it might not be contained in the glossary. It might, however, be found via the “Index” on page 273.

The glossary does not contain terms of other IBM products such as MQSeries.

A

activity. In BTS, one part of the process managed by CICS business transaction services. Typically, an activity is part of a business transaction.

Activities can be hierarchically organized, in a tree structure. An activity that starts another activity is known as a parent activity. An activity that is started by another is known as a child activity.

A program that implements an activity differs from a traditional CICS application program only in its being designed to respond to BTS events.

adapter . The output of the MQSeries Integrator Agent for CICS Adapter Builder. It consists of COBOL source code that is compiled and run in a CICS environment on an OS/390 server. The adapter implements a business transaction.

Depending on how one models the adapter, it can contain a wide variety of functionalities, such as control flow, data flow, sequential navigation, conditional branching including decision and iteration, data typing, storing data context, transformation of data elements, logical operations and custom code.

The adapter can enable any MQSeries-enabled application or application that can initiate a CICS program by invoking one or more server adapter programs, to access:

- Existing CICS transactions (including custom programs) via a distributed program link (DPL).
- Legacy CICS and IMS applications via a 3270 data stream.
- MQSeries-enabled applications via MQSeries.

In the MQSeries Integrator Agent for CICS Adapter Builder, the adapter is built in two complementary ways:

- The structures of messages are imported or created and maintained in the form of message sets, in the Message Set view. See message set.
- The processing of messages is modeled in the form of a microflow in the Adapters view. See microflow.

adapter reply message. A message sent out of the MQSI Agent for CICS run time in response to an adapter request message sent from the controlling application. An adapter reply message contains the result of processing the business transaction that was defined in the request message. Not every adapter request message merits a reply. At build time the request message is formatted to indicate whether or not a reply is required

The adapter reply message is an application-level reply. It is different from a response that is required by a communications protocol. For example, EXCI requires that all requests be responded to at the protocol level. Therefore, if the controlling application used EXCI for the adapter request message and if no adapter reply message was required, a protocol-level response would still be sent. However, this protocol-level response would not be performed by the MQSeries Integrator Agent for CICS. This protocol-level response would not have to be addressed during build time, diagnostics and tracing.

adapter request. The means by which the controlling application invokes the MQSI Agent for CICS run time. An adapter request is sent in the form of an adapter request message.

adapter request message. A message sent by the controlling application to the MQSI Agent for CICS run time to invoke an adapter to process a business transaction. If the controlling application is MQSeries-enabled, the adapter request message is of the form of an MQSeries message. If the controlling application is using a CICS-supplied interface the request message is of the form of a communication area (COMMAREA).

adapter request processing. The programmatic functions (modeled at build time) that an adapter performs in order to manage and fulfill a business transaction on the server run time. To handle the work required by adapter request processing, MQSI Agent for CICS can invoke one or multiple server adapter programs without requiring action by the controlling application. Each adapter request results in a different instance of the Navigation Manager, Navigators and only those server adapter programs that are needed to support that adapter request.

Glossary

asynchronous. An event that occurs at a time that is unrelated to the time at which another event occurs. The two events are mutually asynchronous. The relationship between the times at which they occur is unpredictable.

asynchronous mode. A type of MQSI Agent for CICS run time processing in which the BTS process implements an instance of the MQSI Agent for CICS run time is run asynchronously from the initiating unit-of-work. All BTS activities within that BTS process will be run asynchronously from their parent activities. This has the effect of running the BTS process and all activities as separate units-of-work each with a distinct commit scope.

You would typically want to process a request in asynchronous mode if as a result of the processing, data will be updated.

asynchronous processing. A means of distributing the processing of an application between systems in an intercommunication environment. The processing in each system is independent of the session on which requests are sent and replies are received. No direct correlation can be made between requests and replies and no assumptions can be made about the timing of replies.

auditing. Collecting and recording information about the state of MQSI Agent for CICS run time for the purpose of diagnostics and tracing. MQSI Agent for CICS run time uses BTS facilities for auditing.

audit trail utility. A CICS-supplied utility program, DFHATUP, that enables you to print selected BTS audit records from a specified logstream.

authentication. In computer security, verification of the identity of a user or the user's eligibility to access an object. In MQSI Agent for CICS, the authentication process is established within the MQSeries-CICS bridge via an AUTH parameter passed to the bridge monitor at startup.

B

build time. The time period when business transaction processing is defined, modeled or modified electronically. At build time, a programmer that is familiar with the enterprises business processes uses the MQSeries Integrator Adapter Builder to:

- Extract (and store as structured data types) information from COBOL records and 3270 screens.
- Model and define the Navigators and server adapter programs to be used by MQSI Agent for CICS run time.
- Generate the source code used by MQSI Agent for CICS run time.

build time environment. A modeling environment. The adapter modeling environment for MQSI Agent for CICS runs under Windows NT .

builder. See *MQSeries Integrator Agent for CICS Adapter Builder* on *270*.

business transaction. A self-contained business function. An account transfer for example. Traditionally, in CICS a business transaction might be implemented as multiple user transactions. Using BTS, a business transaction might be implemented as multiple activities. In MQSI Agent for CICS run time, the adapter enables the processing that will manage and complete the business transaction.

C

CICS transaction. A unit of application data processing (consisting of one or more application programs) initiated by a single request, often from a terminal. A transaction may require the initiation of one or more tasks for its execution.

CICS Business Transaction Services (CICS/BTS). A CICS domain that supports an application programming interface (API) and services that simplify the development of business transactions. Using BTS, each action that comprises the business transaction is implemented as one or more CICS transactions. In order to use MQSeries Integrator Agent for CICS Transaction Server you must install CICS/BTS.

communication area (COMMAREA). A CICS area that is used to pass data between tasks that communicate with a given terminal. The area can also be used to pass data between programs within a task. At run time, the DPL Stub program requires that information from the controlling application be passed in the form of a communication area.

compensation. The act of modifying the effects of a child activity. Typically, compensation undoes the actions taken by an activity. For example, compensation for an order activity might be to cancel the order. In MQSeries Integrator Agent for CICS Transaction Server compensation is taken under consideration at build time. If a business transaction is to include compensation, the adapter model needs to reflect it. It can modify the effects of many activities within a given process. Although the Adapter Builder does not support creating compensation microflows explicitly, programmer's can set values in a microflow and use logic in the controlling application to associate one flow to another for the purpose of performing compensation.

controlling application. Any MQSeries-enabled application or any application that is capable of initiating a CICS program. In MQSI Agent for CICS processing, the controlling application is responsible for the overall business flow and that also invokes MQSI Agent for CICS run time. The controlling application manages business context, complex state, multiple request and reply interactions, asynchronous processing, overall business flow compensation and the continuation of one logical request through multiple requests, if required. Examples of controlling applications include MQSeries Integrator, MQSeries Workflow, WebSphere, or any local or remote application can initiate a CICS program.

microflow. The model of the functionality that is realized in the compiled adapter. A user creates microflows using the MQSeries Integrator Agent for CICS Adapter Builder. At build time the microflows are models of all, or part, of the processing of a business transaction during adapter request processing. At run time, the modeled microflows are implemented in adapter request processing.

custom program. A program that augments adapter request processing. A custom program can contain complex rules such as logic and complex IO that could not be modeled using the MQSeries Integrator Agent for CICS Adapter Builder. To invoke the custom program is exactly the same as the mechanism to invoke a microflow processing with CICS transactions, (that is, DPL.)

D

data-container. A named area of storage, maintained by BTS and used to pass data between activities or between different invocations of the same activity. Each data-container is associated with an activity; it is identified by its name and by the activity for which it is a container. An activity can have any number of containers as long as they all have different names. A data-container can be read by all the activities that comprise a process.

Distributed Program Link (DPL). A function of CICS intersystem communication that enables CICS to ship LINK requests between CICS regions. MQSI Agent for CICS run time can initiate programs, including custom programs using one or a sequence of DPLs, via CICS LINK.

DPL Stub program. During MQSI Agent for CICS run time processing, a DPL Stub program defines and runs the BTS process (synchronously or asynchronously) and creates process data-containers.

E

error logging. The process of writing error information to a file. During MQSI Agent for CICS run time processing, the error logging will occur via an MQ queue. An error listener program allows the queue to be drained and hardened to a VSAM resource. Error listener program could be replaced and the queue drained and hardened to resource such as a database, a Tivoli interface or a third party system management package.

F

FEPI. Front End Programming Interface. A terminal emulator that permits CICS programs to interact with other 3270-based applications through virtual terminal sessions. InMQSI Agent for CICS run time, a server adapter program can interface with IBM's FEPI as part of processing a business transaction. The server adapter program interaction with FEPI must be modeled and defined at build time. Using IBM's FEPI product, the server adapter program can send requests to and receive replies from any CICS and IMS application whose 3270 datastream is intended for a SLU2 3278 Model 2 terminal (24 rows by 80 columns), that is, the returned buffer in the a single send and receive is not greater than 3600 bytes.

J

journal. A set of one or more data sets to which records are written during a CICS run:

- By CICS to implement user-defined resource protection (logging to the system log).
- By CICS to implement user-defined automatic journaling (to any journal, including the system log) .
- Explicitly by the JOURNAL command from any application program (user journaling to any journal, including the system log).

journaling. The recording of information onto any journal (including the system log), for possible subsequent processing by the user. The primary purpose of journaling is to enable forward recovery of data sets. In MQSI Agent for CICS run time, journaling refers to the collecting and maintaining information about the state of MQSI Agent for CICS run time and application data to enable the compensation and recovery of the processing of an adapter request message.

The MQSI Agent for CICS run time journaling facility uses CICS/BTS container services to support compensation. Journal information is maintained only during the processing of each adapter request message, except in the case of failure. In the case of failure, MQSI Agent for CICS retains state information and application data for subsequent use in a compensation flow.

The Navigation Manager, Navigators and server adapter programs participate in capturing two types of data that are used for compensation:

- State information is stored in the process and status data-containers as part of the BTS process.
- Journal data is stored in the journal data-container as part of the BTS process.

L

legacy application. An application to which data is sent and from which data is received by MQSeries Integrator Agent for CICS Transaction Server via the FEPI server adapter program.

M

MQSeries-CICS bridge. An IBM product that provides the interface between MQSeries enabled applications and CICS. MQSeries-CICS bridge enables an application, not running in a CICS environment, to run a program or transaction on CICS and get a response back.

If the controlling application invokes the adapter via MQSeries, then the MQSeries-CICS bridge will provide the interface between MQSeries and the run time adapter. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for MVS/ESA. The MQSI Agent for CICS run time does not require users to signon before issuing requests for processing. However, the run time permits customers to check authentication levels based on the user ID and or password in request messages for CICS programs that are run as part of MQSI Agent for CICS run time. The MQSeries-CICS bridge is the control point for establishing the authentication level required. The MQSeries-CICS bridge will link to a DPL Stub program that in turn defines and starts the BTS process that implements adapter request processing.

MQSI Agent for CICS message header. The required portion of the adapter request message that provides the meta-information used by the MQSI Agent for CICS run time for the processing of a message in CICS.

MQSeries Integrator Agent for CICS Transaction Server. A member of IBM MQSeries product family that facilitates development of adapters for business integration solutions. MQSeries Integrator Agent for CICS Transaction Server will enable any MQSeries-enabled application or any application capable of initiating a CICS program to access:

- Existing CICS transactions (including custom programs) via a distributed program link (DPL).
- Legacy CICS and IMS applications via a 3270 data stream.
- MQSeries-enabled applications via MQSeries.

MQSeries Integrator Agent for CICS Transaction Server consists of the following components:

- MQSeries Integrator Agent for CICS Adapter Builder
- MQSeries Integrator Agent for CICS server run time

MQSeries Integrator Agent for CICS Adapter Builder. The part of MQSeries Integrator Agent for CICS Transaction Server that is used to model, build and output adapters to MQSI Agent for CICS run time. The builder provides a graphical environment for modeling adapters. The models generate COBOL source code for deployment of the adapters on an OS/390 server. The builder software consists of the following facilities:

- Importers, for extracting information from COBOL records and 3270 screens and storing the information as structured data types.
- Control Center component, for a GUI that supports the modeling and definition of the Navigators and of the three types of server adapter programs (3270 Dialog Adapter, DPL Adapter and MQSeries Adapter). The definitions are stored as Extensible Markup Language (XML) documents in MQSeries Integrator's repository.
- Generator facility, for reading Navigator and server adapter program definitions from the repository. The generator also reads static templates that contain the portion of server run time that is never affected by modeling. The generator then transforms the definitions and generates source code (COBOL and JCL). The generator sends the source code to the OS/390 server for compilation.

MQSeries Integrator Agent for CICS server run time. The part of the MQSeries Integrator Agent for CICS that runs and executes on an OS/390 server as a CICS application using CICS/BTS facilities. The server run time is capable of operating in a SYSplex environment.

The server run time consists of:

- MQSeries-CICS bridge monitor and link tasks
- A DPL Stub program that links to the Navigation Manager
- A Navigation Manager program that invokes the Navigator programs (The type of Navigator program and server adapter programs generated at run time depends on what was modeled in the builder).
- Three types of server adapter programs. These programs perform processing as modeled in the builder:
  - The FEPI server adapter program that interfaces with IBM's FEPI product to access CICS and IMS applications.
  - The CICS server adapter program that interfaces to the existing CICS transactions, including custom programs that can be developed to augment the adapter, via DPL.
  - The MQSeries server adapter program that interfaces to the MQSeries-enabled applications.
- Error listener program.
- Support of compensation flows and journaling.
- Support of audit levels using CICS/BTS facilities.
- Utility programs that support the server run time.

N

Navigator. MQSI Agent for CICS run time programs that perform adapter request processing, manage states during the microflow processing and invoke server adapter programs. The Navigator and the server adapter programs are generated as the result of modeling via the MQSeries Integrator Agent for CICS Adapter Builder.

Navigation Manager. MQSI Agent for CICS run time program that invokes the Navigator programs. Runs as DFHROOT in BTS process.

P

process. In BTS, a collection of one or more activities. A process is the largest unit that CICS BTS can work with and has a unique name by which it can be referenced and invoked. In MQSI Agent for CICS, the process is uniquely identified by the 36 byte process name value in the message adapter message header (DFHMAHV).

process container. A data-container associated with a process. Process containers can be read by all the activities that make up the process. Note that they are not the same as the root activity's containers.

R

Resource Access Control Facility (RACF). An IBM licensed program that provides access control by identifying users to the system; verifying users of the system; authorizing access to protected resources; logging detected, unauthorized attempts to enter the system; and logging detected accesses to protected resources. RACF is included in OS/390 Security Server and is also available as a separate program for the MVS and VM environments. In MQSI Agent for CICS RACF is used to make sure that a user has the authority to run a particular CICS DPL bridge task.

run time. The time period during which the adapter is operational, with business transactions being managed and completed.

S

screen navigation. A form of data transfer between two application programs in which the first program accesses the second program through a terminal emulator or other communications program, and obtains data that would appear at known screen locations if the second program was being accessed by a human operator. The FEPI server adapter program performs screen navigation to capture 3270 screen images from legacy CICS and IMS applications.

server adapter programs. Any one of three types of programs in the server run time that are invoked by the Navigator program to perform the business transaction activity defined within a microflow at build time.

Server adapter programs include the following:

- The FEPI server adapter program that interfaces to the legacy CICS and IMS applications. It performs screen navigation.
- The CICS server adapter program that interfaces to the existing CICS transactions, including custom programs that can be developed to augment the Message Adapter, via DPL.
- The MQSeries server adapter program that interfaces to the MQSeries-enabled applications.

synch point. A logical point in execution of an application program where the changes made to the recoverable resources by the program are consistent and complete and can be committed. The output, which has been stalled to that point, is sent to its destination(s), the input is removed from the message queues, and any database updates are made available to other applications. When a program terminates abnormally, CICS recovery and restart facilities do not backout updates *prior* to the last completed syncpoint.

synchronous. 1) Pertaining to an event that happens, exists, or arises at precisely the same time as another event. (2) Pertaining to an operation that occurs regularly or predictably with regard to the occurrence of a specified event in another process; for example, the calling of an input output routine that receives control at a pre-coded location in a program. Contrast with *asynchronous*.

synchronous mode. A type of MQSI Agent for CICS run time processing in which the BTS process that implements an instance of the MQSI Agent for CICS run time is run in the same unit-of-work with the same commit scope as the MQSeries-CICS bridge link task. The DPL Stub program (DFHMADPL) and the BTS process initiated by the Stub program are run synchronously as part of this single unit-of-work.

You would typically want to process a request in synchronous mode if the request is merely inquiring on status (an account inquire for example).

synchronous rollback. A type of MQSI Agent for CICS run time processing where, as in synchronous mode processing, the MQSI Agent for CICS BTS process and all activities run within the process are initiated and run in synchronous mode (i.e., BTS RUN ACQPROCESS SYNCHRONOUS and RUN ACTIVITY ( ) SYNCHRONOUS commands) however, any failure within any activity within the process results in an abend of the process. This has the effect of returning the state of any and all recoverable resources updated during adapter request processing to its original state (the state prior to the execution of the failed adapter request or process).

U

user. The persons that interact with both the MQSI Agent for CICS run time and the MQSI Agent for CICS Adapter Builder.

W

workload management. In CICS, a method of optimizing the use of system resources by spreading workload as evenly as possible between different regions.

Index

Printed in U.S.A.

SC34-6087-00