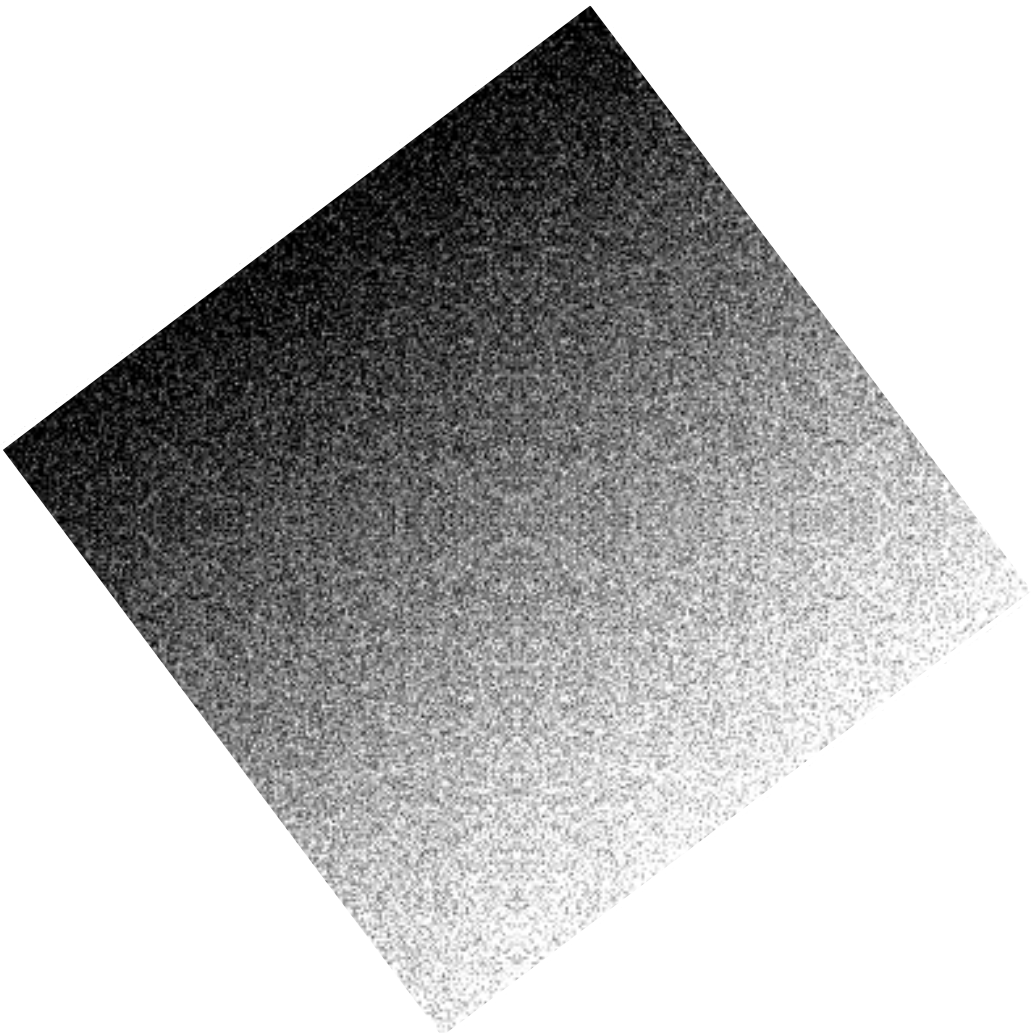
Intelligent Decision Server

IBM

Developing Applications with Cooperative Application Services

Version 1



GC26-8883-00

Intelligent Decision Server

IBM

Developing Applications with Cooperative Application Services

Version 1

GC26-8883-00

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

First Edition (October 1996)

This edition applies to Version 1 of Intelligent Decision Server, 5639–IDS, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

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.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices v
Trademarks v

About This Book vii
Who Should Read This Book vii
Prerequisite Reading viii
IDS Publications viii
IBM Publications viii
Other Publications viii
We'd Like Your Comments ix

Chapter 1. Introducing Cooperative Application Services 1
How CAS Works 1
Summarizing the CAS Transformer's Purpose 3
The Script 3
Typical Applications 3
Other Tools Used with the CAS Transformer 4
Transformers for Data Clean-up 4
Spreadsheet Icons 4
Text Icon for Host Run Log 4
Host Environment and Software 5

Chapter 2. Building and Running a Simple Application 6
Visualizing the End Product 6
Building a CAS Application: the Main Steps 7
Running a CAS Application 8
The Components of CAS 10
The Capsule Control Window 11
The Data Entry Icon 11
The Script 12
The CAS Transformer Icon 14
Directing Input 21
Directing Output 24
Using the Correct Data Types and Formats 24
Using Other Transformers Inside the Capsule Icon 24
Data Format Requirements for Input and Output Regions 25
Requirements for Header Rows 26
Taking Advantage of Run-Time Aids 27
Host Run Log 27
Capsule Run Log 29
Stopping a Capsule Process 34
Capsule Services 34
Capsule Summary 34

Chapter 3. Using the API 37

© Copyright IBM Corp. 1996

The General Purpose of Each API . . . 37
The API and REXX . . . 38
The Usual Order of API Functions . . . 38
API Return Codes . . . 38
API Syntax and Use . . . 39
METFCLOS—Close an Output Region . . . 39
METFGET—Get an Input Region . . . 39
METFOPEN—Open an Output Region . . . 40
METFPUT—Write a File to an Output Region . . . 40
METMDSP—Display Important Message . . . 41
METMGET — Get CAS Error Message . . . 42
Defining Data Format . . . 42
IXF Format Syntax . . . 42
Data Format Syntax . . . 43
Remarks . . . 43
Converting Data Types . . . 46
Specifying Header Rows . . . 47

Appendix A. Sample Capsule Applications . . . 48
Sample 1: A File Transfer Routine . . . 48
Sample 2: A Flat-File Filter Application . . . 53
Sample 3: A Cross-Tabular Analysis Using SAS . . . 62
Sample 4: A Regression Analysis Using DB2, DXT, and SAS . . . 73

Appendix B. Sample Supporting Routines . . . 98
Purpose of Supporting Routines . . . 98
METCMD . . . 98
METDXT . . . 100
METDXTRN . . . 103
METERROR . . . 106
METFILE . . . 106
METRGN . . . 109
METSUBMT . . . 113

Glossary . . . 115

Bibliography . . . 120
Referenced Publications . . . 120
IBM Publications . . . 120
Other Publications . . . 120
IDS Publications . . . 120

Index . . . 122

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling (1) the exchange of information between independently created programs and other programs (including this one) and (2) the mutual use of the information that has been exchanged, should contact:

IBM Corporation
555 Bailey Avenue, W92/H3
P.O. Box 49023
San Jose, CA 95161–9023

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of IBM Corporation in the United States or other countries or both:

ACF/VTAM	DXT	PROFS
AIX	Enterprise System/9000	PS/2
Application System/400	ES/9000	QMF
APPN	IBM	RACF
AS/400	MVS/ESA	RISC System/6000
Capsule	MVS/XA	RS/6000
DATABASE 2	OfficeVision	S/390

© Copyright IBM Corp. 1996

DataJoiner	OfficeVision/VM	SQL/DS
DB2	Operating System/2	System/360
DB2 for OS/2	Operating System/400	System/370
DB2/6000	OS/2	Virtual Machine/Enterprise Systems Architecture
DFSORT	OS/400	VM/ESA
Distributed Relational Database Architecture	Personal System/2	VTAM
DRDA	Presentation Manager	Win-OS/2

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This books contains information about the features of Cooperative Application Services (CAS) and the function of the CAS transformer tool.

The product offers application developers the ability to link their Intelligent Decision Server (IDS) client desktops to server resources available on host systems, thus adding power and flexibility to their desktop applications. CAS is designed for interface with MVS operating systems.

Other IDS tools (notably Capsule, Query, Spreadsheet, Text, Plot, and Forms Design, as well as other transformers) are also used in building applications around the CAS product. Details on those other tools are in the manual provided for each product; you will find references to them in appropriate places in this manual.

This manual includes several diverse samples of applications that use CAS. It lists critical code and points out important considerations in building such applications. The complete sample applications, along with the CAS transformer, are provided in the Cooperative Application Services file drawer.

Who Should Read This Book

This manual is written for application developers rather than for non-technical end users. The applications that are described in Appendix A, “Sample Capsule Applications” on page 48 are designed to be used in their finished form by anyone who uses the other tools and features of the IDS desktop. However, this manual consists mainly of instructions to the more technically trained people who would develop such applications.

To illustrate the flexibility and interoperability of the CAS product, several sample applications are shown invoking a variety of software. It is not assumed that every application developer who uses this manual has the entire range of skills listed below; typically, the expertise of several different people would contribute to building a single, complex application using CAS. It is assumed that among the audience, the following combination of skills and preparation are found:

Technical background, and system-level knowledge both of IDS and of the user's own host applications and system requirements. (Required of all users)

Facility with the IDS Capsule tool, and with other tools that commonly make up capsule applications, such as Forms Design and Data Entry, Spreadsheet, Text, Plot, and various other transformers. (Required of all users)

Knowledge and experience in using IBM's Restructured Extended Executor (REXX) language, as well as of DATABASE 2 (DB2), DFSORT, Data Extract (DXT), and Integration Exchange Format (IXF); and of the SAS Institute's SAS. (Only as required by respective applications)

© Copyright IBM Corp. 1996

Prerequisite Reading

This manual covers in detail CAS and the CAS transformer. It discusses in less detail several other IDS tools that are commonly used to build CAS applications. It also mentions IBM and other software used to complete those capsule applications. For more information on those other components, please refer to the following manuals. They are grouped as:

IDS publications

IBM publications

Other publications

IDS Publications

The following manuals provide detailed information on the tools that are most commonly used to build capsule applications.

Transformers User's Guide and Reference	SC26-8907
Capsule User's Guide	SC26-8906
Data Entry User's Guide	SC26-8897
Forms Design User's Guide	SC26-8899
Plot User's Guide	SC26-8905
Query User's Guide	SC26-8901
Spreadsheet User's Guide	SC26-8904
SQL Entry User's Guide	SC26-8898
Text User's Guide	SC26-8896

IBM Publications

Using QMF	SC26-8078
TSO Extensions Version 2 REXX Reference	SC28-1883
TSO Extensions Version 2 REXX User's Guide	SC28-1882

Other Publications

Practical Usage of REXX, A.S. Rudd (Ellis Horwood)	0-13-682790
REXX Language: A Practical Approach to Programming, M.F. Cowlishaw (Prentice Hall)	0-13-780651-5
REXX in the TSO Environment, Gabriel F. Gargiulo (QED Information Sciences)	0-89435-354-3
SAS Language and Procedures, SAS Institute, Inc.	ISBN 1-55544-371-0
SAS Procedure Guide, SAS Institute, Inc.	ISBN 1-55544-378-8
SAS Companion for the MVS Environment, SAS Institute, Inc.	ISBN 1-55544-398-2

SAS/STAT User's Guide, (two volumes) SAS Institute, Inc. ISBN 1-55544-376-1

We'd Like Your Comments

Your feedback is important to us in providing you with the most accurate and high-quality information. If you have any comments about this book or any other IDS documentation, please visit our Website at:

http://www.dss.ibm.com

There you'll find the Feedback Connection page where you can enter comments and send them to us.

Chapter 1. Introducing Cooperative Application Services

Cooperative Application Services (CAS) is a set of utilities that connect your IDS client desktop to a host server to give you access to new data sources and additional computing resources. Since CAS is implemented as a transformer, you can easily integrate it into an Intelligent Decision Server (IDS) capsule application.

CAS allows the application developer to include host data or processing steps in an IDS capsule application. Host processing can be invoked by an application that uses a CAS transformer, and data can be exchanged between desktop and host. Instructions for host processing are contained in a command script, to which the transformer refers, inside the capsule application.

Within a single application, you can now combine the friendly IDS user interface with the resources that your host system provides. You can use CAS to perform the most powerful and complex computations and analyses available on your host system, using all the data accessible to your network and/or host systems, without seeming to leave your desktop capsule application. You also receive information on processing status from both the desktop components and the host system as the application runs.

You can also use CAS to retrieve data from IBM's Query Management Facility (QMF) and bring the data into IDS for further processing. You do this by creating a command script that calls a QMF procedure. For more information about using QMF and creating QMF procedures, see *Using QMF*.

The CAS transformer is only one of 60 different transformers in IDS. Each type of preprogrammed transformer manipulates data in a particular way to automate common IDS functions. For example, the Sort transformer sorts data in alphabetical, numerical or chronological order, and the Merge transformer combines text with incoming data to produce a merged document, such as a report or series of letters. IDS also provides a blank transformer that you can name and program yourself. To find out more about these tools, refer to the IDS publications *Transformers User's Guide* and *Advanced Transformers User's Guide*. (See “Using Other Transformers Inside the Capsule Icon” on page 24 for a list of suggested transformers to use in CAS applications.)

How CAS Works

The following steps show in very simple terms how a CAS application looks and works. More complex examples are illustrated in Appendix A, “Sample Capsule Applications” on page 48 and Appendix B, “Sample Supporting Routines” on page 98.

1. A capsule application using the product can include some form of input to be sent to the host (input is represented in Figure 1 on page 2 by an In icon); a CAS transformer icon; a Text icon containing a command script; an output icon to receive the results of host processing (here, that is represented by an Out icon); and a Text icon to receive log data.

© Copyright IBM Corp. 1996

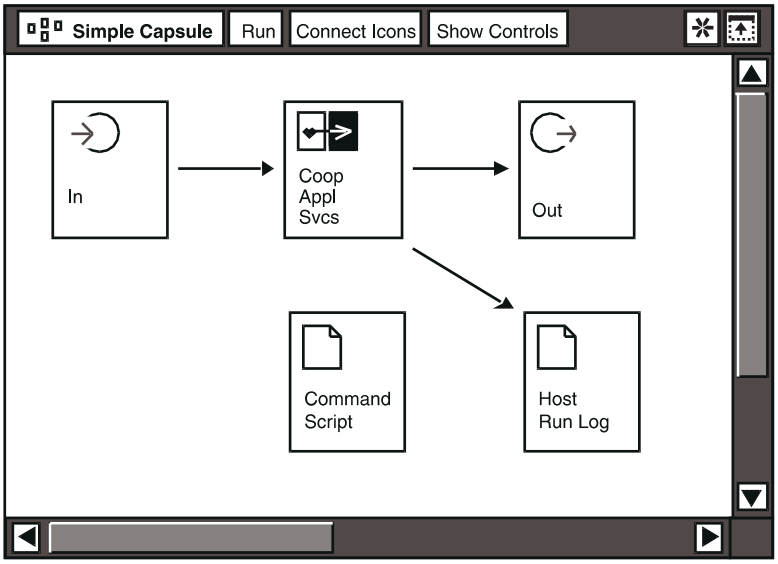


Figure 1. A simple capsule application using CAS

2. When the **Run** button is clicked on, input data is transferred into a specified input region of the transformer.
3. Upon receiving the input data, the transformer works with a database gateway service to open a session with the host, sends a copy of the script to the host, and transfers control to the script.

   The script now becomes the “traffic director.” It:

   Transfers data from the CAS transformer's input regions to the host system

   Directs host-resident software to perform pre-defined processes (for example, extract data from a host database and create a file)

   Returns output data to the desktop when all the required host-based steps are finished, and deposits the data in a specified output region in the CAS transformer

   In a typical CAS scenario, the system monitors the status of each process invoked on the host. When the required steps are done, the host service transfers system output data into the last output region of the transformer, the region designated as the host output log. This host run log is especially helpful to the user if a step has failed or has caused the whole sequence to end abnormally. In a capsule application such as the one shown in Figure 1, this system output would then be transferred to the Text icon named Host Run Log for easier access and readability.

   The host run log is different from the Capsule Run Log, which is described starting on “Capsule Run Log” on page 29. If you wish, you can have both types of these logs in a Capsule that uses CAS.
4. The transformer then ends its session with the host system.

5. The data from the transformer's output region is sent to the output icon, where it is manipulated into the desired form.

The CAS transformer is typically used in the Capsule environment. However, it can also be used directly from the desktop, which can be helpful while you are developing or testing host-based applications. In Chapter 2, “Building and Running a Simple Application” on page 6 you will find information on configuring a Capsule icon and the transformer itself. In Chapter 3 are instructions for using the CAS application program interface (API) in a command script.

Summarizing the CAS Transformer's Purpose

To summarize the purpose of the CAS transformer, it:

Accepts input data into a specified number of input regions

Establishes a connection, and starts a session, with the host

Reads a command script, transfers a copy of it to the host, and transfers control to the script

Transfers input data to the host when requested by the script

Accepts output data from the host into a specified number of output regions

Accepts host-generated log data into its last output region, thus creating a host run log

Sends output data to specified tools in a Capsule window

The Script

The script is a collection of commands, typically written in the Restructured Extended Executor (REXX) language, that are meant to be executed on the host system. A script usually contains commands to:

Create permanent or temporary data files

Bring data from transformer input regions to the host system

Invoke and run programs, passing parameters to them

Send resulting data back to transformer output regions

The sample command scripts shown in this manual are written in REXX code. A command script can be written in any command language that the host system supports on TSO/E under MVS. However, the API functions that effect data transfer and message retrieval must be invoked from a REXX environment.

Typical Applications

A good candidate for CAS is any application which needs data from a host-resident database, a file found on the host, or processing that can be done only (or more effi-

ciently) on the host. Your organization can also find CAS useful in bringing output from existing host systems into Capsule icons for further desktop processing.

The sample applications provided in Appendix A, “Sample Capsule Applications” on page 48 include:

- A simple file transfer between desktop and host
- A cross-tabular analysis using SAS under MVS
- A statistical regression analysis

Other Tools Used with the CAS Transformer

In a particular capsule application, you can use other IDS tools to further enhance your data:

- Transformers for data clean-up
- Spreadsheet icons
- Text icon for the host run log

A brief discussion of each follows.

Transformers for Data Clean-up

When data is transferred between an IDS desktop and a host, it undergoes conversion and might need to be formatted or cleaned up. This is because the CAS transformer accepts only tabular data, that is, tables whose rows and columns are filled with data.

For that kind of clean-up work, you can use some other IDS transformers, such as Clean, Compress, Select, Row Clean, or Replace. They can be placed inside the Capsule icon before (to the left of) the transformer to affect data going into the host, and/or after (to the right of) the transformer to affect data returning to the desktop.

More on this subject appears in Chapter 3, “Using the API” on page 37.

Spreadsheet Icons

Often, input to the CAS transformer resides in one or more Spreadsheet icons, within the same Capsule icon that contains the transformer. Likewise, output from the transformer often goes into a Spreadsheet icon for further manipulation and analysis.

Text Icon for Host Run Log

When you are building a capsule application, you can make the log of host run data more visible than it would be inside the CAS transformer. In that case, you can direct the output to a Text icon within the Capsule icon, as shown in Figure 1 on page 2. More on this subject appears in Chapter 2, “Building and Running a Simple Application” on page 6.

Host Environment and Software

The CAS transformer is designed to work with an MVS operating system under TSO/E. Sample scripts are provided in this manual for adaptation and use in MVS environments.

The range of host-resident software that a CAS transformer and its associated script can invoke is extensive. In this manual are sample applications that use a flat file, a DATABASE 2 (DB2) database, and SAS. However, a script can be written to call any accessible, non-interactive software that resides on a host system under MVS, and to return data to an IDS desktop.

For more information on the host environment as it relates to CAS, please refer to the following IDS publication:

Administering Host Services for MVS: DB2 and Cooperative Application Services

For more information on the LAN environment as it relates to CAS, please refer to the following IDS publications:

System Administration Guide and Reference

Installing IDS LAN Components

Configuring and Administering Database Gateway Services

Chapter 2. Building and Running a Simple Application

This chapter lists the steps involved in building a capsule application that uses Cooperative Application Services (CAS). It also covers issues you must consider when running a CAS application.

This is a model for a capsule application, and it is presented only for illustrative purposes. Although this application can be built according to the instructions in this chapter, the actual application is not furnished in software form. However, Appendix A, “Sample Capsule Applications” on page 48 contains actual sample applications that are furnished in the IDS file drawer Cooperative Application Services.

Visualizing the End Product

Suppose that you are going to build a capsule application that, when finished, will look like the one shown in Figure 2.

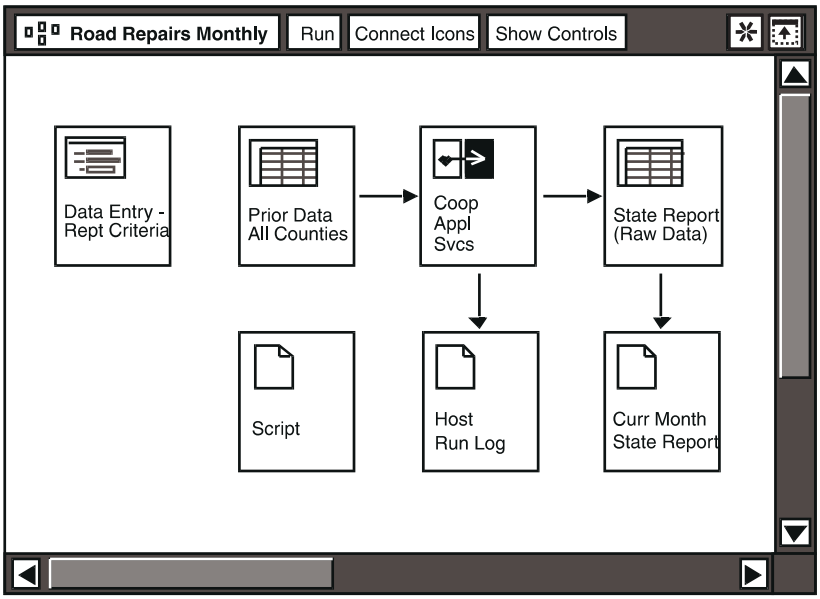


Figure 2. Simple capsule application

The purpose of the simple capsule application shown in Figure 2 is to produce a monthly record of county road repairs throughout the state. To produce the report, the application:

Combines

- Selection criteria that a user enters through the Data Entry icon named **Data Entry-Report Criteria**
- Data in the Spreadsheet icon named **Prior Data All Counties**

© Copyright IBM Corp. 1996

– Information from a database maintained on a host

Funnels the results into another spreadsheet called State Report (Raw Data).

Formats the spreadsheet data into the Curr Month State Report.

Instructions for building this simple application start on the next page.

Building a CAS Application: the Main Steps

To start building this model capsule application, you would do as follows:

1. Have the following icons ready on your Intelligent Decision Server (IDS) desktop. (Presumably, you would have already prepared such icons and the data in them.)

   A Spreadsheet icon containing Prior Data All Counties

   A Text icon containing the command script, or program that directs processing on the host system

   Another Spreadsheet icon formatted to contain State Report (Raw Data)

   Another Text icon formatted to receive the final Curr Month State Report

2. Go to the New Icons/*Blank Icons file drawer and copy a blank capsule icon onto your desktop. Open the capsule icon and copy the following icons into it:

   Forms Design (from the New Icons/*Blank Icons file drawer)

   CAS transformer (from the New Icons/*Transformer Icons file drawer)

   Text (for the command script and the host run log, from the New Icons/*Blank Icons file drawer).

   A script is needed in order for a CAS application to work. For details on writing a command script, see Chapter 3, “Using the API” on page 37, the sample applications in Appendix A, “Sample Capsule Applications” on page 48, and the sample routines in Appendix B, “Sample Supporting Routines” on page 98.

3. Use the Forms Design tool to create and name the Data Entry icon.

   This manual focuses on CAS, rather than on related operations such as building capsule applications and creating Data Entry icons. If you have not worked with the Capsule or Forms Design tools before, please refer to *Capsule User's Guide* and *Forms Design User's Guide*.

4. Configure the transformer icon as described in “The CAS Transformer Icon” on page 14.

5. Rename each icon appropriately.

6. Place arrows between appropriate icons, and direct the flow of data by completing an Arrow Options window for each arrow.

   No arrow is needed between the transformer and the Script icon, or between the Data Entry icon and the transformer. Those relationships are set up internally.

Figure 3 on page 8, which spans three pages, shows the 13 steps this model capsule application goes through to complete a successful run.

Running a CAS Application

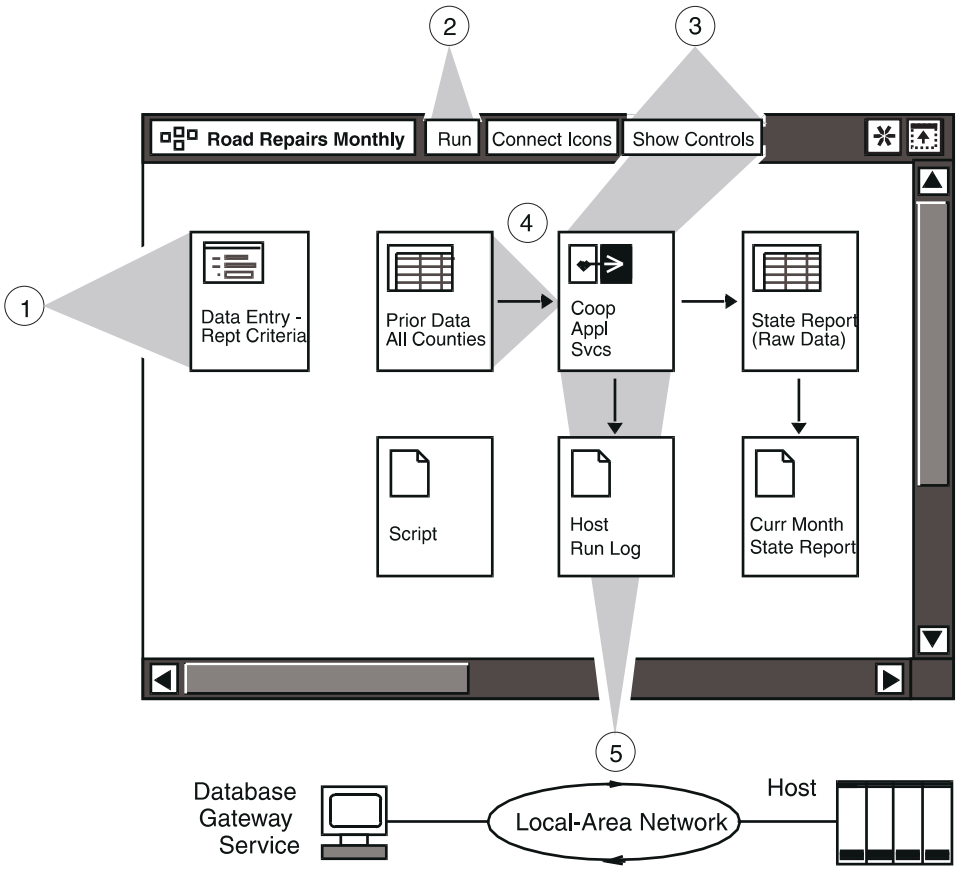


Figure 3. capsule application run, steps 1 through 5

1. The user enters report selection criteria in **Data Entry - Report Criteria**. This sets variables in the User Input Control window, from which the CAS transformer later requests those variables. (See Step 6 on page 9.)
2. The user clicks on the **Run** button.
3. The CAS transformer takes its report selection criteria from the Data Entry icon through the Capsule icon's User Input Control window.
4. The transformer takes its input data from the spreadsheet icon named **Prior Data All Counties** and places it in the transformer's last input region.

5. Use of the last input region prompts the transformer to start a session with the host system through a database gateway service, and proceed to the next step.

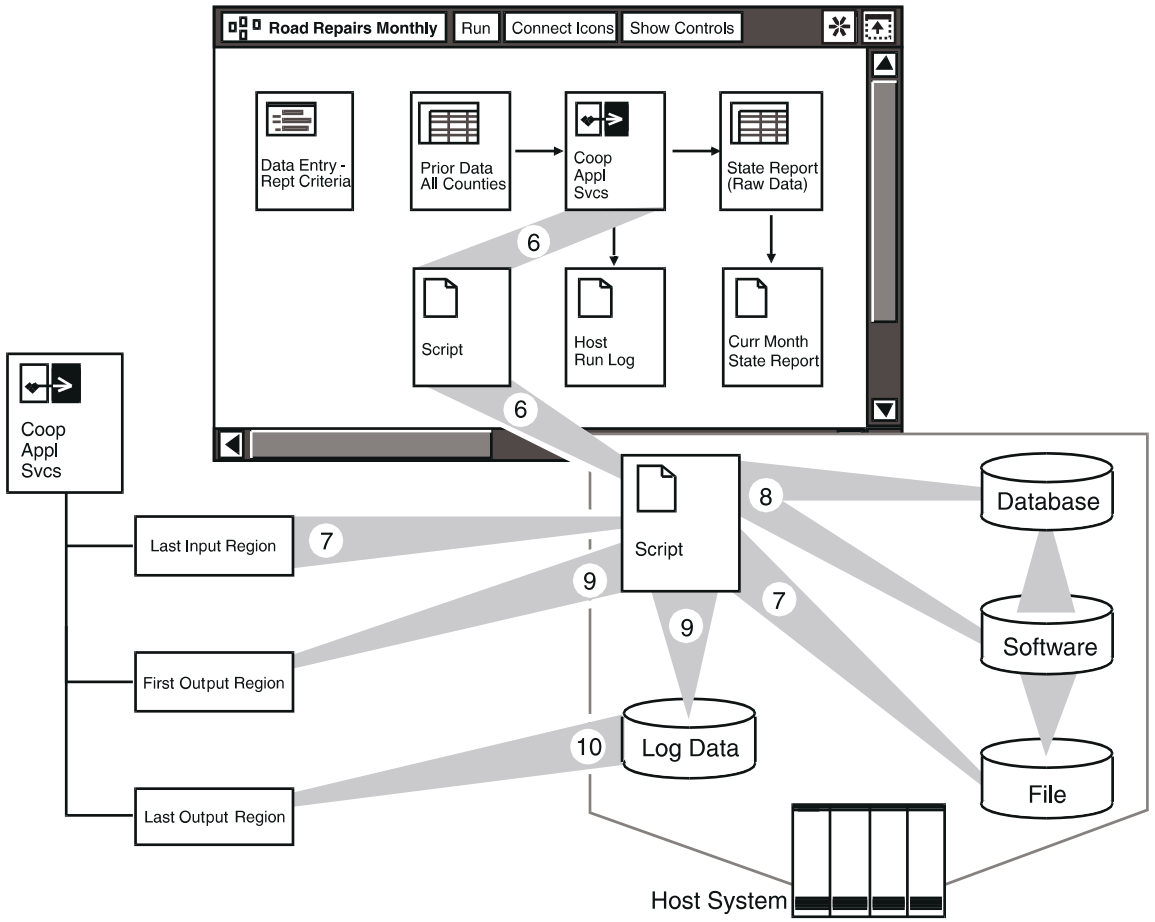


Figure 4. Capsule application run, steps 6 through 10

6. The transformer reads the Script icon, requests @-variable values from the Capsule Controls window, substitutes those @-variable values for @-variables found in the script, moves a copy of the script to the host for processing, and hands over control to the script.
7. The script opens the transformer's input region and copies its spreadsheet data (Prior Data All Counties) to a host file. (In this figure, the transformer is depicted outside the Capsule icon merely to illustrate the sequence of events. In reality, the transformer stays inside the Capsule icon.)
8. The script calls host-resident software to extract data from a database and combine it with the Prior Data All Counties data.

9. The script copies the resulting data into the transformer's first output region, and passes control back to the host service.
10. The host service copies the host run-log data to the transformer's last output region, which is reserved for the log.

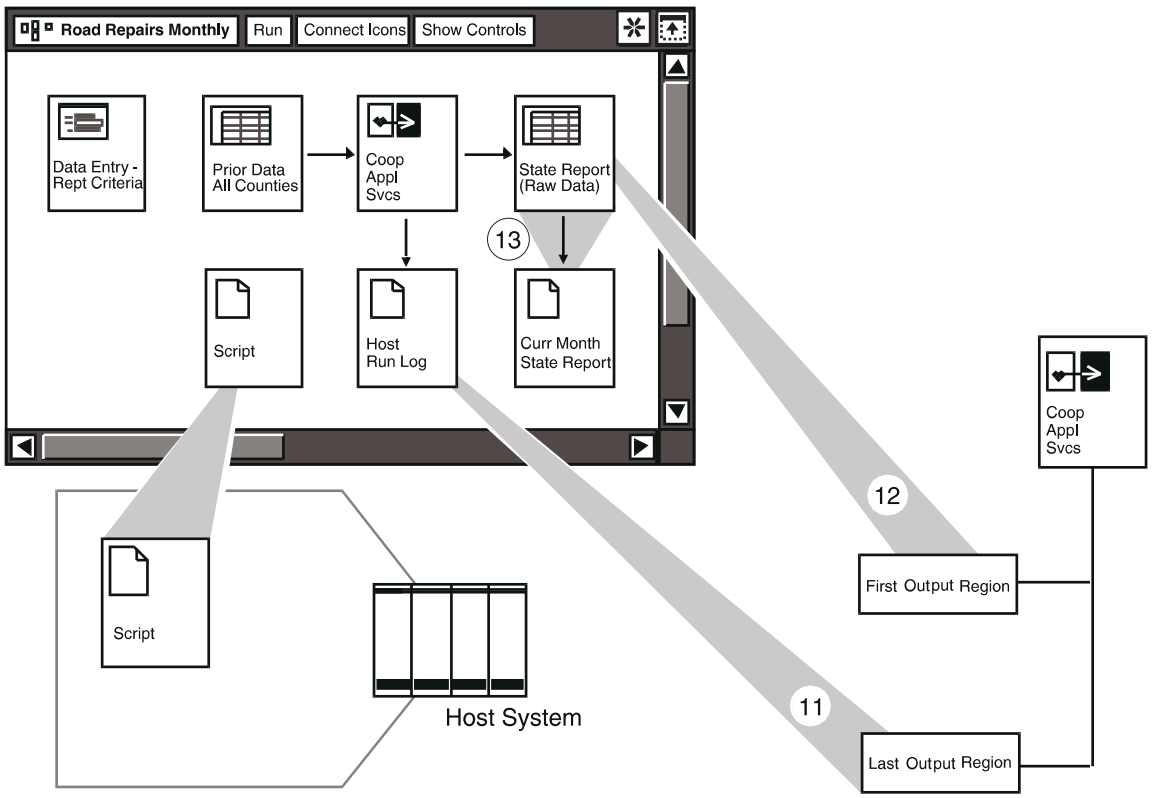


Figure 5. Capsule application run, steps 11 through 13

11. The capsule application copies the data from the transformer's last output region to the icon named Host Run Log.
12. The capsule application copies the data from the transformer's first output region to the icon named State Report (Raw Data).
13. The report is formatted into final form in the Text icon Curr Month State Report.

The Components of CAS

In the following sections, you will find a description of each icon and window that you would use to build this model capsule application.

The Capsule Control Window

In the capsule application described in this chapter, the user opens the Data Entry - Rept Criteria icon and enters variables for a particular report. Such variables might include a range of dates for the report to cover (in this case, June, 1994), or selected counties rather than the entire state (in this case, North specifies the counties in the northern half of the state). The Data Entry icon provides a convenient, visible user interface with which to enter @-variable values. However, the user can bypass the Data Entry icon by entering the values directly into the Capsule's User Input Control window. (This application could also have been built without a Data Entry icon.)

You can confirm that the @-variable values have been updated by opening the capsule's User Input Control window, which is shown updated in Figure 6. To open it, just select the **Show Controls** button in the capsule window header.

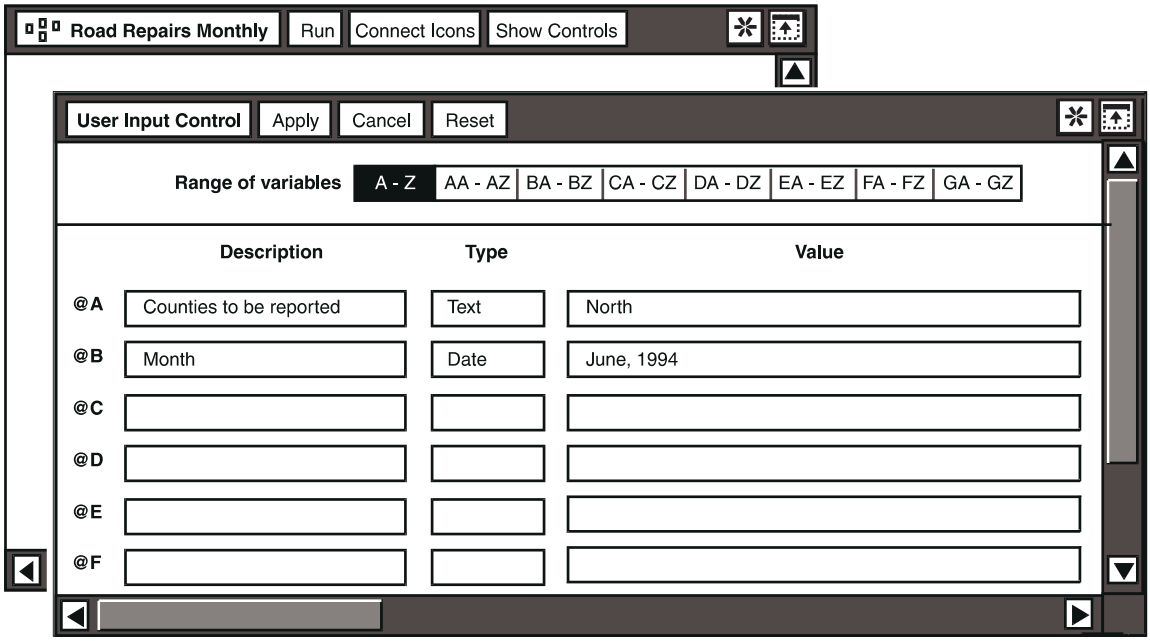


Figure 6. User Input Control window

You can design flexibility into any capsule application by using @-variables. The *Capsule User's Guide* explains in detail the many places and ways you can use them.

The Data Entry Icon

When you open the Capsule icon in this chapter's example, and then open the Data Entry - Rept Criteria icon, the following window appears.

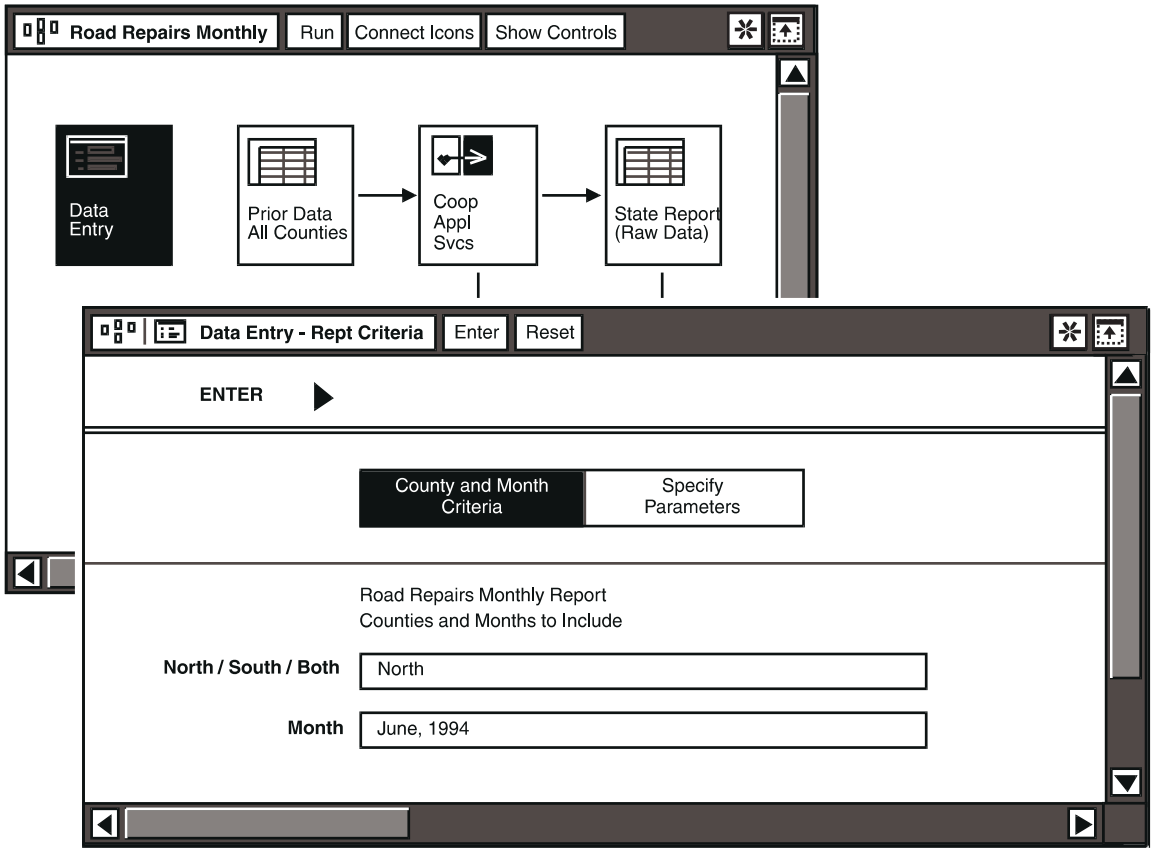


Figure 7. Data Entry icon window

In the fields shown in Figure 7, the user types the criteria for this run of the application. (In the case of this model application, the criteria are the area and period this report is to reflect.) When the user clicks on the **Enter** button, the Data Entry tool updates the values of the @-variables in the Capsule icon's User Input Control window. The script then incorporates the @-variable values into its instructions to the host.

As the application's developer, you would have used the IDS Forms Design tool to build the Data Entry icon. For more information on building forms, please refer to *Forms Design User's Guide*.

The Script

The command script in this model capsule application is entirely contained in a Text icon called Script. The icon resides on the desktop until the application is run and the CAS transformer transports a copy of the script to the host for processing.

Figure 8 on page 13 shows a typical command script contained in a Text icon. (To learn more about using the Text tool, refer to *Text User's Guide*.)

Figure 8. Text icon Script, and first page of the script it contains

A script is typically a REXX program consisting of commands to obtain data or invoke processing on a host system. A script can go on for many pages if necessary.

Other Locations for Scripts

To make processing more flexible, a script can also reside elsewhere:

In the Text icon to which the transformer refers, you could place a skeleton script that calls and processes a host-resident script. You can find it useful to maintain long scripts in the host environment; this way you avoid the overhead of sending a long script to the host each time the capsule application is run.

A script can reside somewhere else on the desktop, for example, in a folder or file drawer. To use such a script, you would specify its path name and file name in the Transformer Controls window.

Instead of referring to a particular Text icon, you could specify a script name using an @-variable. If you then made that @-variable refer to one of several scripts in a File Drawer icon, the user would have a choice of scripts at processing time. For

example, one script would result in a report that simply furnished a list of the roads repaired in each county that month, while another script would run a statistical analysis of road repair cost and frequency in the southern part of the state versus the same in the northern part.

Each script would still need to be in a separate Text icon. Instead of entering a script's name and path in **Parameters**, you would use an @-variable to specify the appropriate script. The @-variable (that is, the script's path name and file name) could be entered in the Capsule icon's User Input Control window or through a Data Entry icon at the time the capsule application is run.

Details are in "Specifying the Script to the Transformer" on page 16.

If you have had some experience using the IDS Capsule tool, you probably have noticed that in the model at the front of this chapter, there is no arrow connecting the script to the transformer. This is because the transformer calls the script from the entry in its **Parameters** field, as you will see in the following section.

For more information on command scripts, see the sample scripts in Appendix A, "Sample Capsule Applications" on page 48.

Writing Scripts with REXX

Actual sample scripts are included later in this manual. They are all written in the Restructured Extended Executor (REXX) language. If you wish to modify the samples to run on your own system, you must have REXX installed.

REXX is automatically included in the MVS TSO/E version required for CAS.

In your own applications, you can write the main script in any command language that the host supports on TSO/E. It does not have to be written in REXX. Conversely, a REXX script can call a procedure written in any supported command language. However, the application program interface (API) functions that effect data transfer and message retrieval must be called from a REXX environment. Therefore, if you wish to write your script in a command language other than REXX and use the API, you can do so by writing a simple REXX procedure that invokes the API function, and calling that REXX procedure from your script.

In Chapter 3, "Using the API" on page 37 you will find details on using the API.

The CAS Transformer Icon

The CAS transformer tool controls the execution of the cooperative application, and lets you specify which scripts, input and output regions, and log regions are to be used when the application runs.

When you copy a new CAS transformer icon into a Capsule icon (or to your desktop) and open it, you see windows like those shown in Figure 9 on page 15.

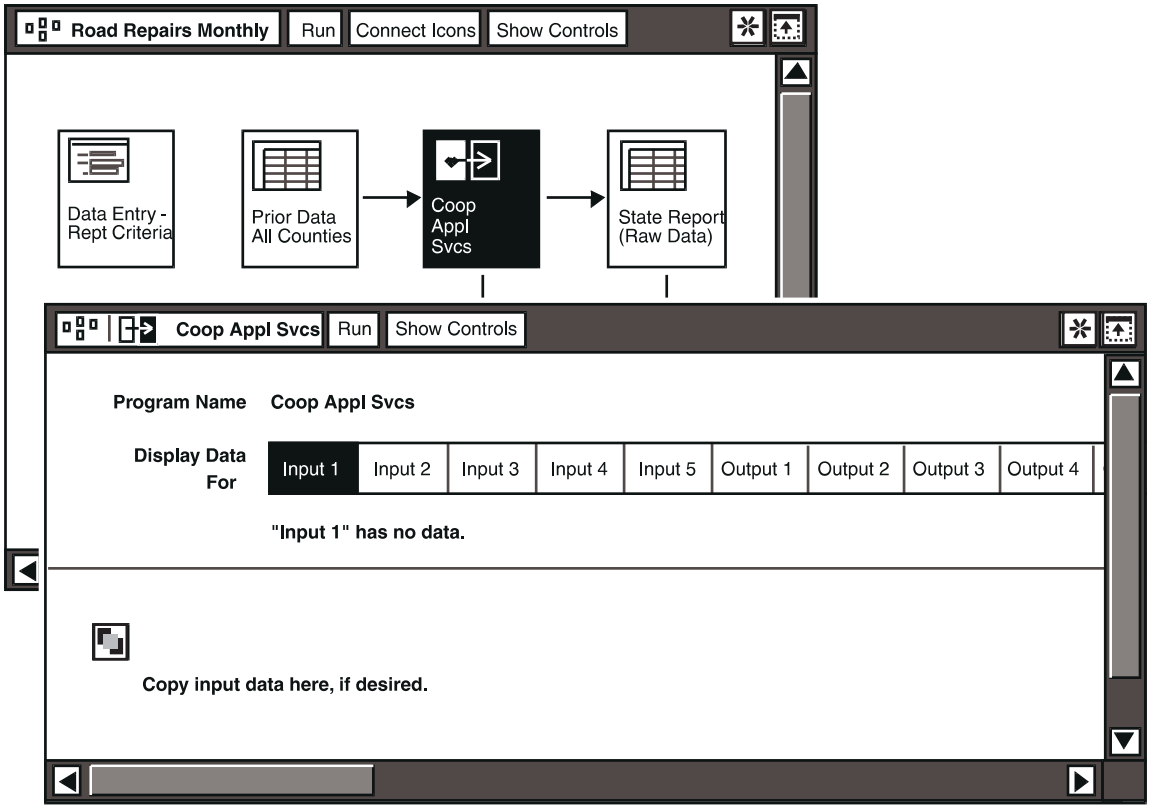


Figure 9. The open CAS transformer icon

In the lower window, titled Coop Appl Svcs, you see a row of selection boxes for the input and output regions. To see more output regions, simply scroll to the right.

Note that your installation has received the Cooperative Application Services transformer icon in two different forms:

Coop Appl Svcs, which has five input regions, 10 output regions and one log region (for log data coming from the host)

Coop Appl Svcs 30, which has five input regions, 30 output regions and one log region

If your application requires that you transfer many different outputs back from the host system, you will probably need the transformer with the larger output capacity.

The first input region is automatically highlighted the first time you open a new, blank CAS transformer icon. The message "Input 1" has no data appears below the boxes; this is normal. After you direct input to an input region, or output to an output region, you select that region to see the region's contents displayed. An example of a transformer with a filled input region is shown a little later.

Specifying the Script to the Transformer

To specify the script that should direct the host processing in this application, do the following steps:

1. Click on the Transformer Controls window (as shown in Figure 10).

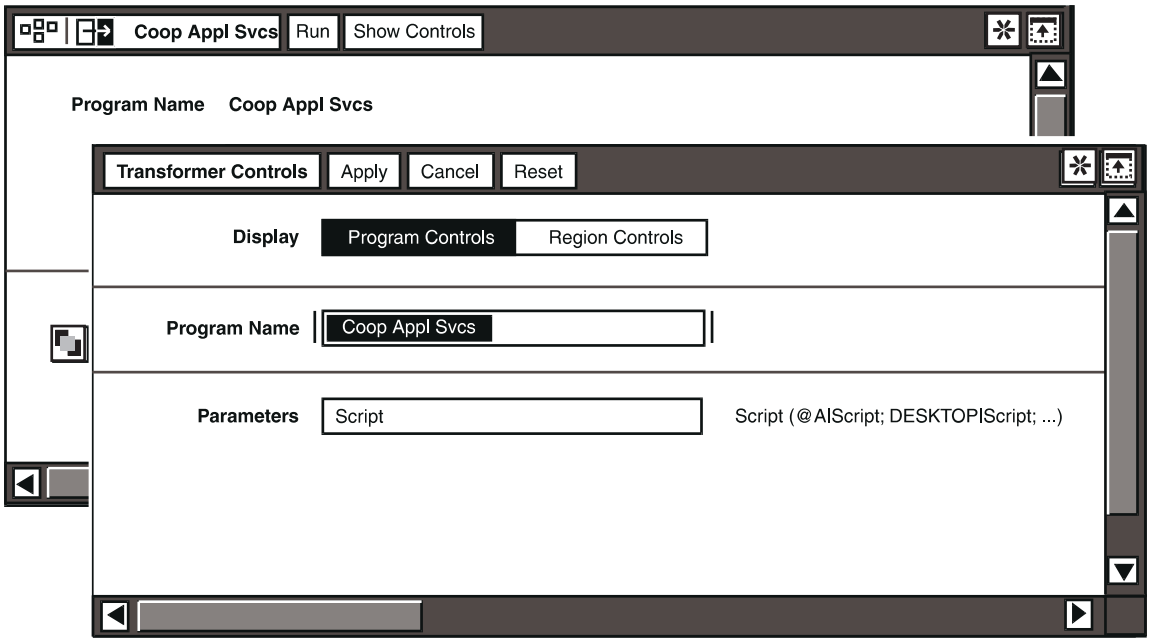


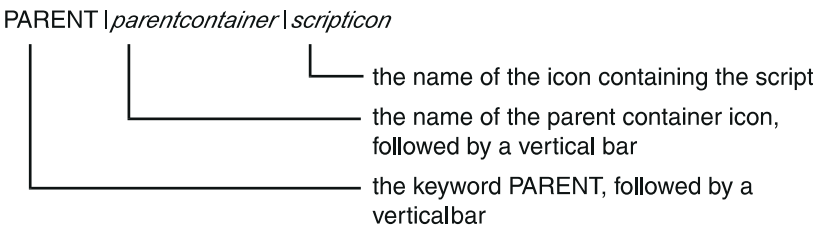
Figure 10. Transformer Controls window

2. Display the fields you need by selecting the **Display Program Controls** field. Now you see the **Program Name** and **Parameters** fields below the line.
3. In the **Program Name** field, notice the program name for this icon, Coop Appl Svcs.
4. In the **Parameters** field, enter the exact name of the Text icon that contains the script. In this case, since the script is in an icon named Script and since that icon resides in the same Capsule icon as the transformer, the correct entry is Script.

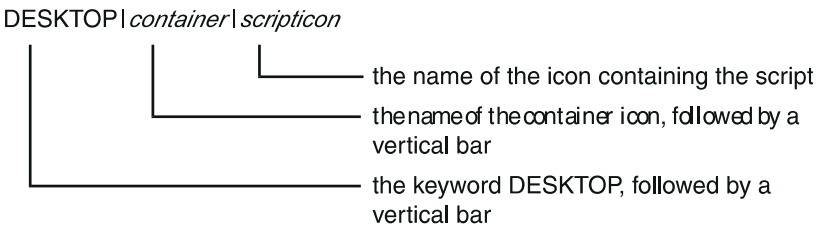
   If your script is not located in the Capsule icon, you must use its full path name in the **Parameters** field. You can find a full discussion of path names and keywords in *Transformers User's Guide*. Briefly, the rules for pathnames and keywords are as follows:

   If the script icon is embedded in the same container as the transformer, it is at current level. Use only the script icon's name.

   If the script icon is either inside or embedded inside a container that is a level above the container holding the transformer, it is at the parent level. Use this format:



If the script is either inside or embedded inside a container on the same desktop as the transformer, but is not within the same Capsule icon or series of Capsule icons as the transformer, it is at the desktop level. Use this format:



If a script is in a file drawer, you can place a copy of the file drawer in your Capsule icon to make the naming process easier.

Close the window to enter and save your input.

Customizing Region Names (Optional)

At some point you can customize the input and output regions by changing their names from **Input 1**, **Output 1** and so forth, to something more descriptive.

To name regions, do the following:

1. Open the transformer icon.
2. In the window header, click on the **Show Controls** button.
3. In the Transformer Controls window, select the **Display Region Controls** field.
4. The Region Controls window now appears. There is a field for each input region and each output region where you can enter your new region names.

The Region Controls window is shown in Figure 11 on page 18.

If you wish to save the input data from each run to the next, select **Yes** in the **Save Input Data** field. Otherwise, the data is cleared from the first four input regions after each time the capsule application runs. (Data in the last input region, **Input 5**, is always cleared.)

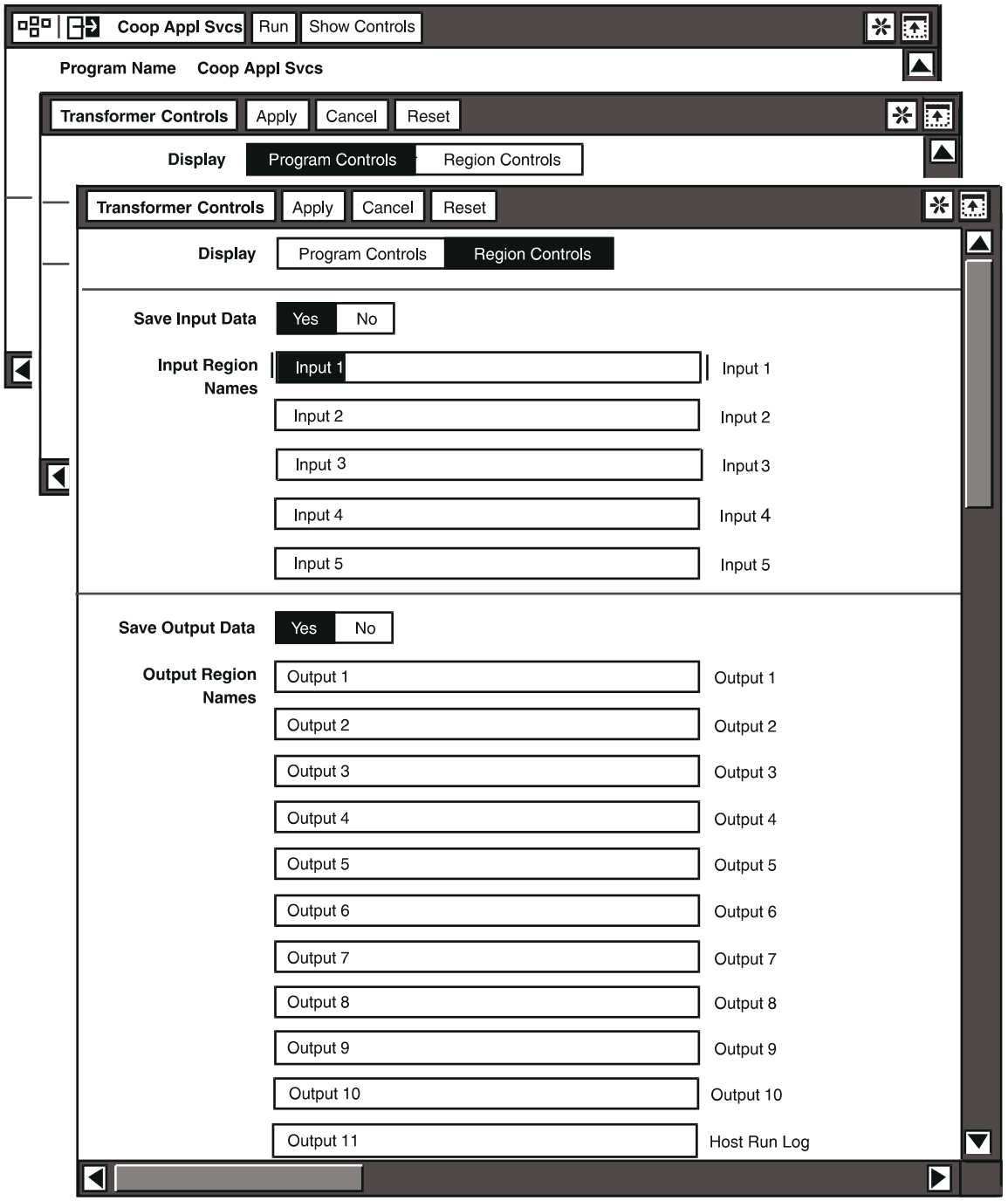


Figure 11. Region Controls window of a transformer (5 input regions, 11 output regions)

Specifying Logon and Service Data

To log on to the host system through a database gateway service, you must go to the transformer's Icon Options window, shown in Figure 12.

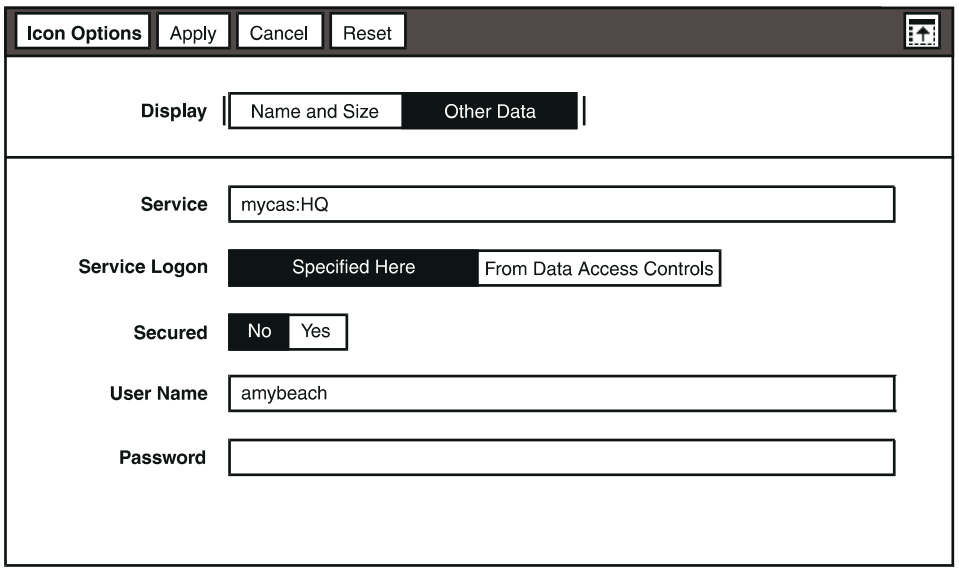


Figure 12. CAS transformer Icon Options window

To get to the window, select (but do not open) the CAS transformer icon, open the options window, and do the following steps.

1. Type the gateway service name in the **Service** field.
2. Select a **Service Logon** choice. If you want to specify the logon identification data in this window, select **Specified Here**. If you want to have that data supplied automatically, select **From Data Access Controls**. (Data Access Controls are available through the Special menu in the bottom right corner of your IDS desktop.)
3. Type the appropriate host logon ID in the **User Name** field.
4. Type the appropriate host password in the **Password** field.
5. Close the window.

If you wish to restrict access so that the **User Name** field is restricted to use under the desktop ID of the current user, select **Yes** in the **Secured** field. Note, however, that the Secured operation is irreversible. The default choice is **No**.

Displaying Input and Output

To send input from an icon to one of the transformer's input regions, you use a connecting arrow; see the first figure in this chapter for an illustration. The directions for setting up an arrow are next.

If you are opening the transformer for the first time, not having run the capsule application yet, you see the **Input 1** field highlighted and, below it, the following message: "Input 1" has no data.

After the application has run, you see data in **Input 1**, looking something like the window shown in Figure 13. The input regions are cleared before each run unless you have chosen **Yes** in the **Save Input Data** field (Region Controls window).

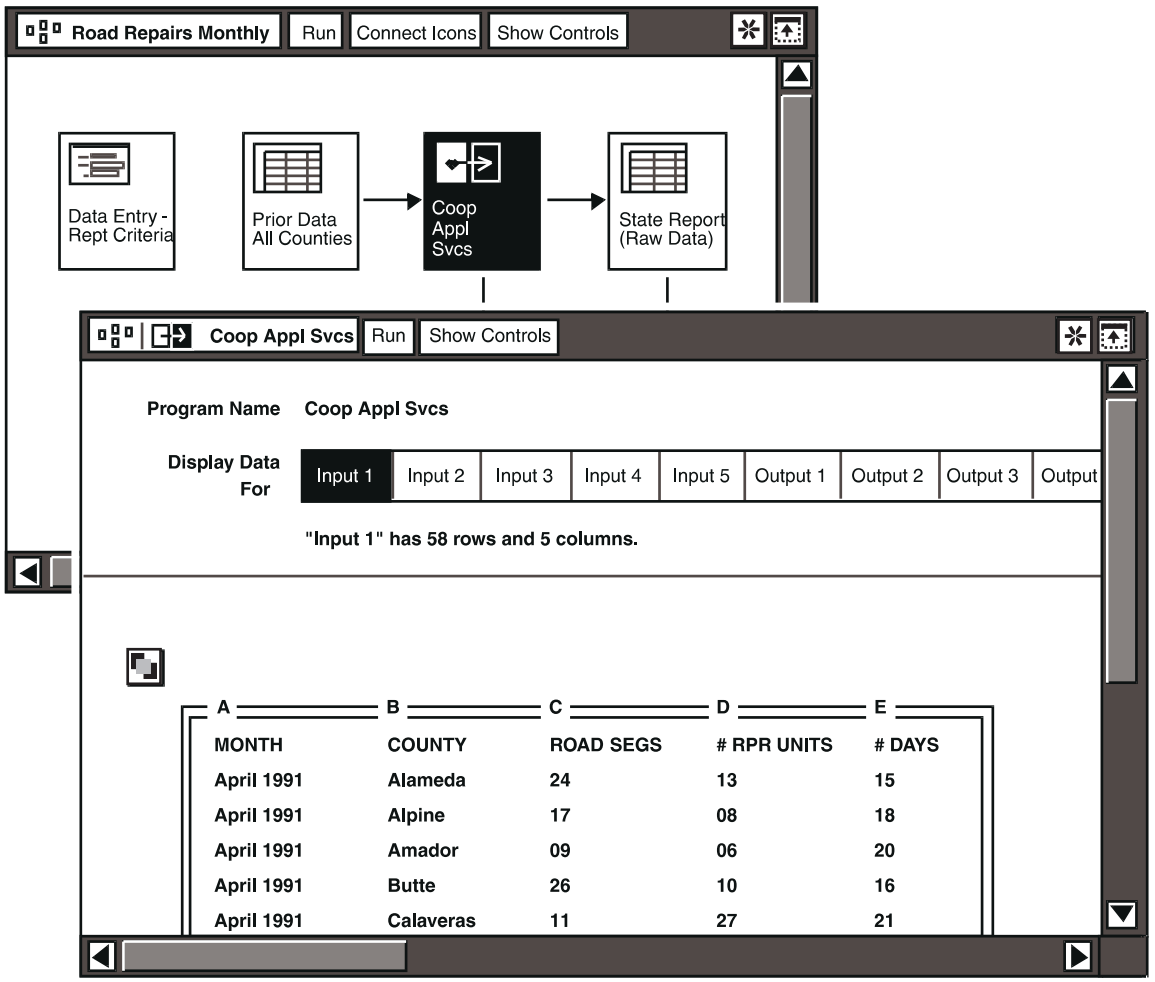


Figure 13. CAS transformer with input data

If you were building the capsule application described in this chapter, and by now had set up the arrow between the Spreadsheet icon named Prior Data All Counties and the transformer, specifying **Input 5** as the destination, you would see no data below the line. This message would appear: In a Capsule, the Transformer uses, but does not save, the last Input Region. Check Output Region(s) for results. Copy input

data here, if desired. This is true only for **Input 5**, not for any of the other input regions.

When you select an **Input** or **Output** block into which data has been transferred, the rows and columns of data appear below the line. Because the transformer's output regions have a limited display capacity (which depends on the number of rows and columns in the incoming data, as well as on your workstation's available space), you can not see all the output you expect here. If there is more output data than the transformer's output region has room to display, you would see a dotted line at the bottom of the output display.

So that all the output data is available for viewing, you should always direct the data from the output region to an output icon such as the State Report (Raw Data) spreadsheet shown at top right in Figure 13 on page 20. See “Directing Output” on page 24 for instructions.

Directing Input

To direct the flow of processing in a capsule application from one icon (that is, from one stage of the process, or one data area) to the next, you use Capsule arrows. The *Capsule User's Guide* provides complete details on building all types of capsule applications, including how to use arrows. In this section are brief instructions for using arrows to direct input to the transformer's input and output regions from icons in the same Capsule icon.

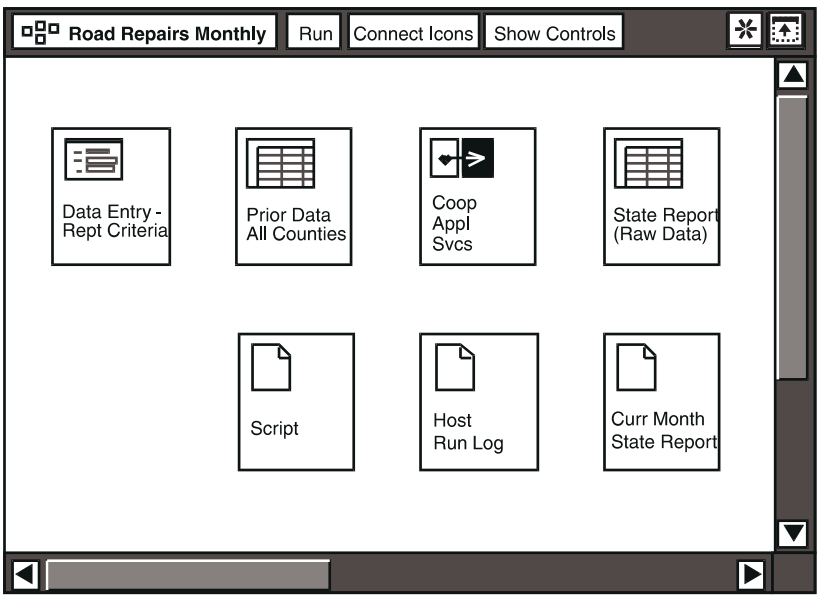


Figure 14. Capsule window before connecting enclosed icons

In Figure 14, you see the Capsule icon before the necessary arrows have been set in place. Comparing this figure with Figure 15 on page 22, you notice that in the finished Capsule icon there are arrows from:

Prior Data All Counties to Coop Appl Svcs

Coop Appl Svcs to State Report (Raw Data)

Coop Appl Svcs to Host Run Log

State Report (Raw Data) to Current Month State Report

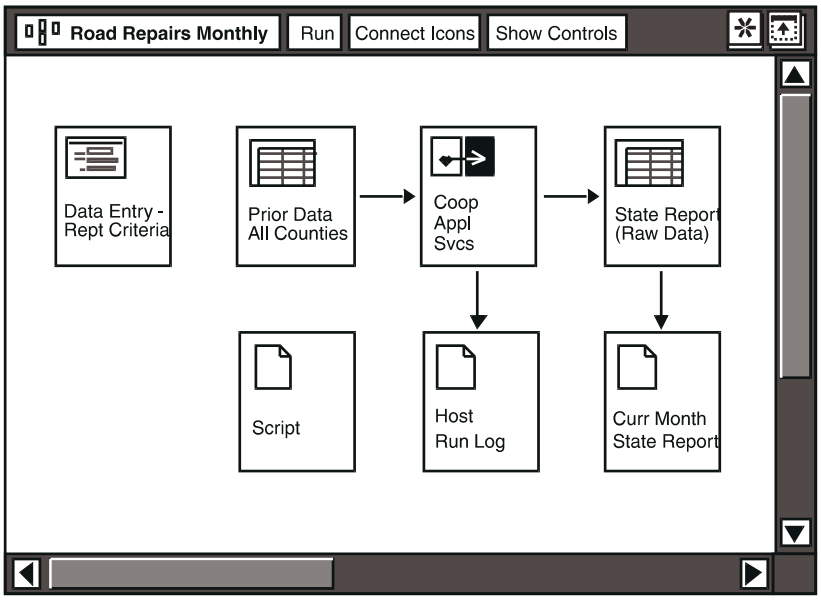


Figure 15. Capsule icon after connecting arrows have been added

To set the first arrow, and connect the Prior Data All Counties icon to the Coop Appl Svcs icon, follow these steps:

1. Select the Prior Data All Counties icon.
2. Click on the **Connect Icons** button in the Capsule window header.
3. Notice that the message Connecting. -- Select destination icon appears. Select the Coop Appl Services transformer icon.

   Notice the arrow that has now appeared, itself selected, as shown in the top screen of Figure 16 on page 23.

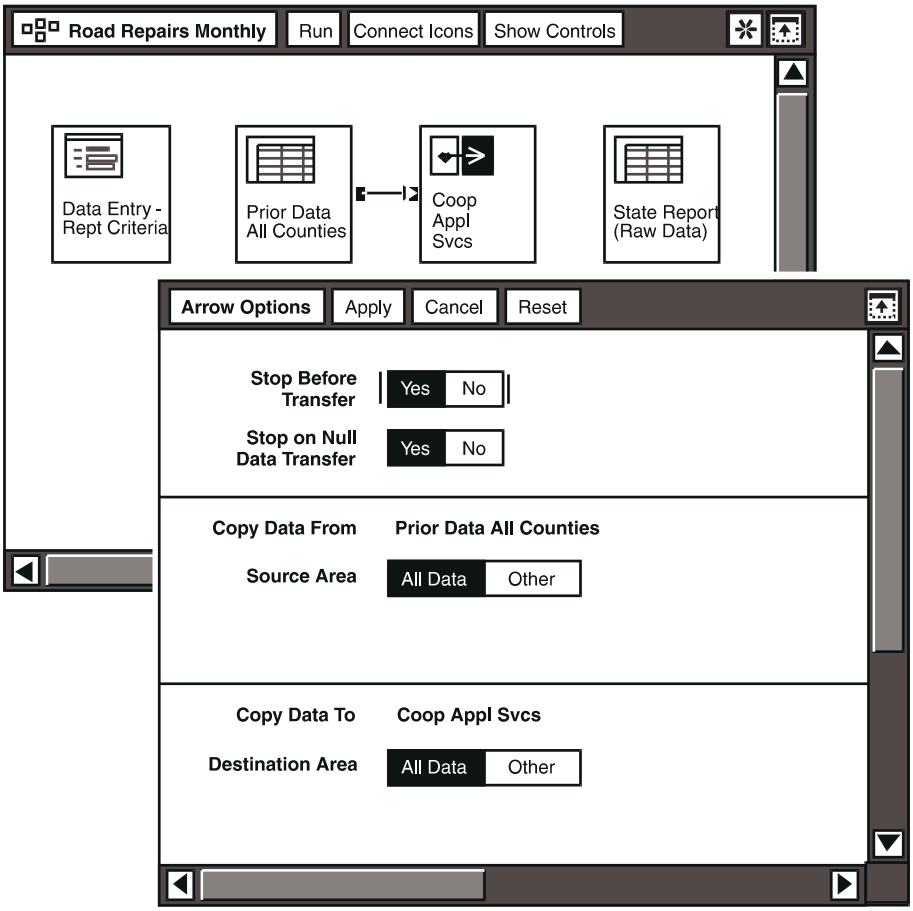


Figure 16. Capsule icon with selected arrow and Arrow Options window

4. With the arrow still selected, open the options window. The Arrow Options window is shown in Figure 16 as it first appears, before your input.
5. Leave **All Data** selected in the **Source Area** field. (As it happens in this model, all the data is being copied from the Spreadsheet icon to the transformer's input region. In another case, you might wish to copy specific regions of the source icon into the transformer's input region. To do so, you would select **Other** instead of **All Data**.)

   Notice that the destination icon is displayed in quotation marks, opposite **Copy Data To**.
6. To specify the last input region of the CAS transformer, select **Other** opposite **Destination Area**. You now see a blank field appear below the **Other** field. Type Input 5 into the field, making sure you include the initial capital (***Input***) and leave one space between ***Input*** and ***5***. Select the **title block** in the window header.

The Arrow Options window disappears and the arrow between the two icons is still selected. As soon as you begin to set up the next arrow by selecting the next source icon, the first arrow will be deselected. Before you click on the **Run** button, be sure all arrows and icons are deselected.

Be sure that the last input region you specify is **Input 5**, since it is the use of that region that prompts the transformer to begin the processing sequence.

An application can be designed without any input to be conveyed to the host system. For example, the purpose of a particular application could be to retrieve data from the host and manipulate it on the desktop. However, if the application you are writing includes any input for the transformer, then the last input region must be used.

Directing Output

You direct data from the CAS transformer's output regions to output icons in the same way:

1. Connect the two icons with an arrow.
2. While the arrow is selected, open the options window.
3. Specify the source region by selecting **Other** in the **Source Area** field and entering the output region name. An example is **Output 1**, including the initial capital**.**
4. Leave the **All Data** field opposite **Destination Area** selected.
5. Select the title block to close the Arrow Options window.

Using the Correct Data Types and Formats

Data that passes through the input or output regions of a CAS transformer must conform to certain rules. When data leaves your IDS desktop and goes out to a host system, it can be transformed to be read and processed appropriately by the host-resident software that your script calls. Similarly, data returning from the host system to your desktop might need manipulation.

Following are suggestions for ensuring that your data will be in an acceptable format, and some basic rules for data format.

Using Other Transformers Inside the Capsule Icon

One way to make sure data is “clean” (either going to the host or coming back from the host) is to use one of these other transformers:

Clean, which removes from a spreadsheet any columns and/or rows that contain blanks, N/As, zeros, or null values

Compress, which combines and/or deletes two or more columns or rows of data

Select, which removes and/or reorders columns of data

Row Clean, which selectively removes rows of data from a Spreadsheet or Query icon, or from another row-oriented data source

Replace, which selectively substitutes values in data from a Spreadsheet or Query icon

You can place the appropriate transformer icon between the icon containing the input data and the transformer to massage data going to the host, or between the transformer and the output icon to affect data coming back. Again, you connect them with arrows, specifying sources and destinations through the Arrow Options window.

For more information on using these transformers, see the *Transformers User's Guide*.

Data Format Requirements for Input and Output Regions

Before applying other transformers to your data, you will probably want to know what data formats work with CAS. Data types and formats are discussed in greater detail in Chapter 3, “Using the API” on page 37. Information follows on the types of data that the transformer's input and output regions can accept.

Data coming into a CAS transformer must conform to certain rules:

The data must be tabular, that is, in the form of a table, with rows across and columns down.

All the data in a given column must be of the same type, except for data in the first row. The first row (across all columns) can consist of titles. If the first row contains all text, the transformer automatically identifies it as a title row.

The data must be rectangular, that is, all the rows must have data in all their columns. Rectangular and non-rectangular data are illustrated in Figure 17 on page 26 and Figure 18 on page 26.

Data going into a transformer region must not contain any null values, unless the corresponding target file on the host is in Integration Exchange Format (IXF).

The CAS transformer's input and output regions do accept blanks, errors, N/As, and Boolean values (true/false); the transformer treats them as text. Therefore, they can occur in a column of similar data (their own type, or text), but they can not occur in a column of numerical data.

An N/A value in an input region is converted to a null value in the host target file if that file is in IXF format.

Figure 17 on page 26 shows the shape of data that the transformer regions can accept.

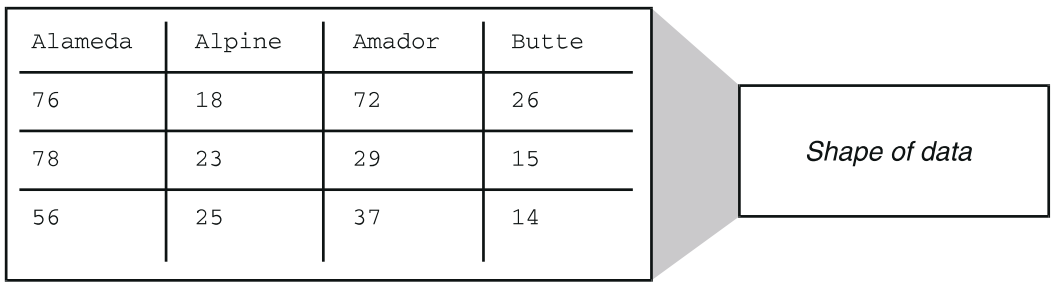


Figure 17. Rectangular data, acceptable for input and output regions

In Figure 18 you see a representation of non-rectangular data, which the transformer cannot accept.

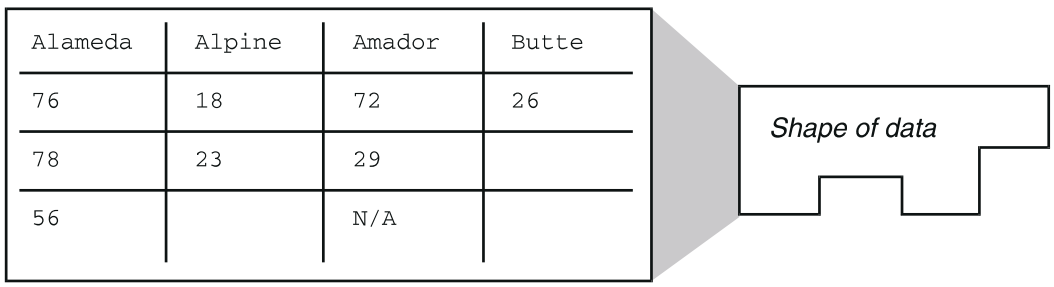


Figure 18. Non-rectangular data

Non-rectangular data is not acceptable for input or output regions unless corresponding host files are in IXF format.

Requirements for Header Rows

The CAS transformer can read data coming into its input and output regions as data for calculation, or as header rows (column names, not to be included in calculations), depending on the type of data it encounters. To make sure that your data is treated as you intend, observe the rules explained in Table 1 on page 27.

A header row can be all blanks.

Table 1. Header rows and data types

Data Going to Host through Input Region		
If first row of data contains:	**The following happens:**	**Data Returning from Host through Output Region**
All numbers, or a mix of numbers and text	The transformer reads the first row as data, and automatically supplies its own header row of all nulls. You can supply your own meaningful header row.	The rules for data coming from the host back to the transformer are different. The entire data file is returned as data, with no header rows (rows of column names). If you want column names in data returning from the host:
All text	The transformer reads the first row as a header row. To avoid losing data from calculations, always supply a header row.	If the data is in IXF format, it already contains IXF column names. If the data is in flat-file format, you can specify column names. See Table 5 on page 43 for instructions.

Taking Advantage of Run-Time Aids

IDS provides several facilities that you might find helpful when you are running cooperative applications. They are described in the following sections:

"Host Run Log"

"Capsule Run Log" on page 29

"Stopping a Capsule Process" on page 34

"Capsule Services" on page 34

"Capsule Summary" on page 34

Host Run Log

When the script has finished processing, the host software automatically sends the host system's run data to the last output region of the transformer. To ensure that the run data is moved automatically from the output region to the Text icon named Host Run Log, follow the directions in the previous section, "Directing Output" on page 24. Always obtain the host run log from the last output region, which is reserved for the log.

If the Run is Successful

Under normal circumstances, all the user has to do is open the Host Run Log icon and read the run log. If there is a failure at some point in the run, there can be no new data in the Host Run Log icon. However, the user can still find the new host run log by following the steps described next.

If the Run Fails

If an application using CAS fails during a host-based processing step, the host run data is still sent automatically to the transformer's last output region, but it can not be transferred automatically to the Host Run Log icon. Therefore, in the applications you

design, you can include user instructions to this effect: If the application fails, the user might need to manually transfer the run log from the last output region to the log icon, and then open that icon to view the run log.

To manually transfer all data from the log region to the Run Log icon, the user follows these instructions:

1. Open the Host Run Log icon. If it contains old data that you do not need now, click on the Select All button in the window header and delete the data. (The Host Run Log icon's Select All button is shown in Figure 19.) Leave the icon open on your desktop.

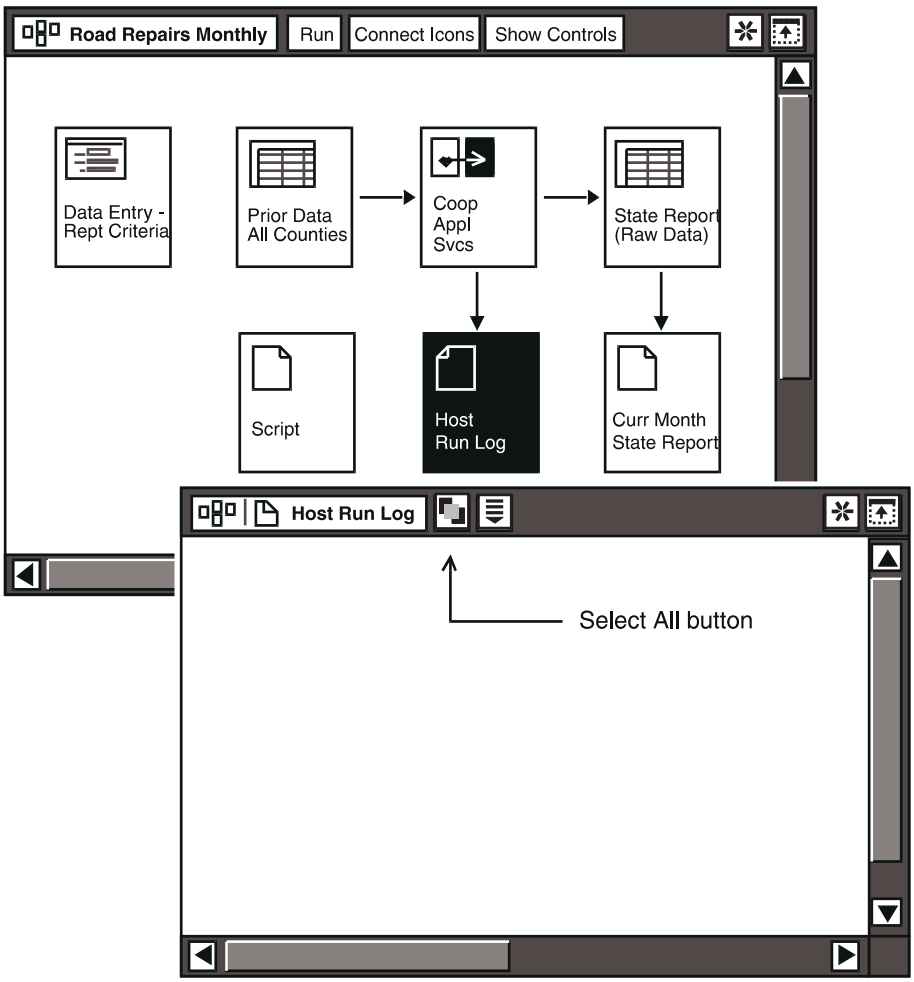


Figure 19. Select All button in Run Log icon window header

2. Open the transformer.
3. Scroll, if necessary, to the last output region and select it.

4. Click on the Select All button in the display area of the transformer window. (The transformer's Select All button is shown in Figure 20 on page 29.) Note that the contents of the region display window are selected.

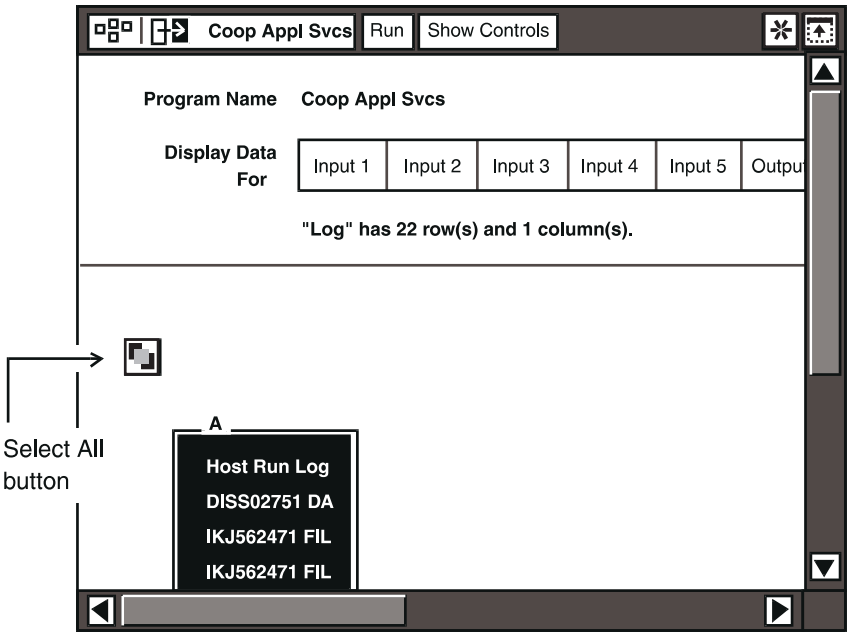


Figure 20. Select All button in transformer window

5. Press either the Copy key or the Move key.
6. Select within the window of the Host Run Log icon. The entire text of the run log now appears there.

Capsule Run Log

If you pull down the Special menu from the Capsule icon's window header as shown in Figure 21 on page 30, you see the choice Create Run Log.

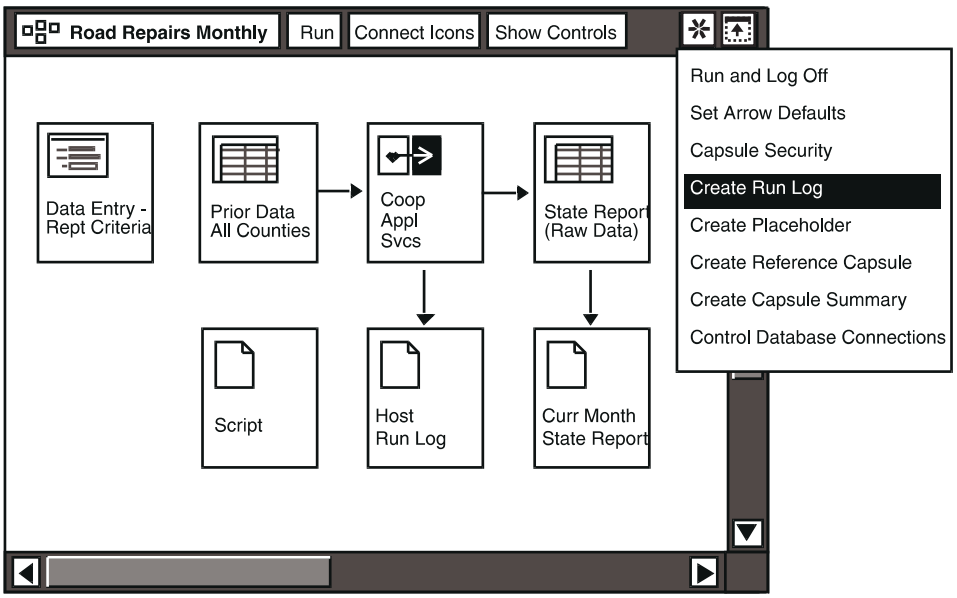


Figure 21. Capsule icon showing Special button pull-down menu and Create Run Log option

When you select Control Run Log window opens.

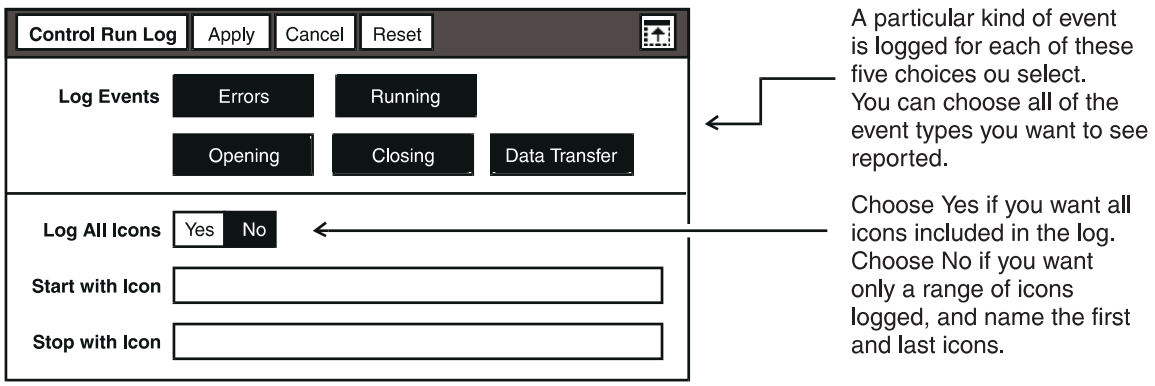


Figure 22. Control Run Log window

When you finish making your selections and close the Control Run Log window, the next capsule run automatically creates a log of data about the run. The log, which IDS names Capsule Run Log, appears under or among the other icons in your Capsule icon, as shown in Figure 23 on page 31. The log is reused with each run, and it stays in your Capsule window until you remove it.

If there is no extra space among the icons in your Capsule icon, the Capsule Run Log icon can be placed below the last visible row. In that case, pull the window open farther to see Capsule Run Log.

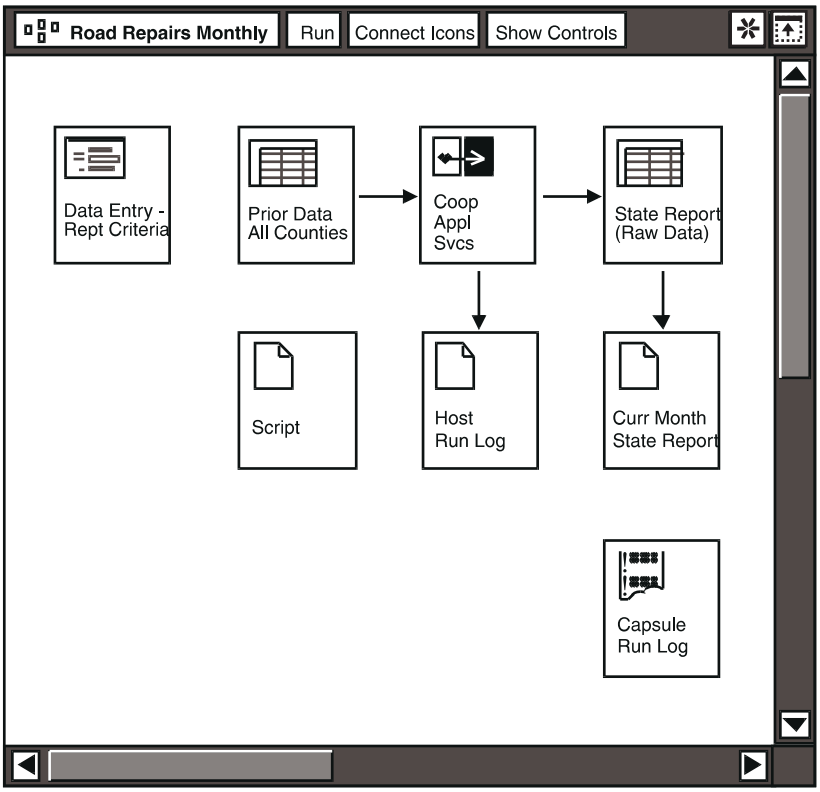


Figure 23. Capsule icon showing new Capsule Run Log icon

When you run the capsule application, the Capsule Run Log icon records the following information:

Table 2 (Page 1 of 2). Types of Capsule Run Log events and their meanings

Log Event	What It Means
Errors	The time a problem or error occurred during the run, and any information available about the cause of the error. This includes all stops, whether automatic or manual.
Running	The time each logged icon began running.
Opening	The time each logged icon was opened. For most icons, this means reading in the stored data and preparing the icon to run. For data access icons, this includes establishing a connection to a database.

Table 2 (Page 2 of 2). Types of Capsule Run Log events and their meanings

Log Event	What It Means
Closing	The time each logged icon was closed and information obtained from the icon was filed. For data access icons, this includes disconnecting from the database.
Data Transfer	The time data transfer between icons started.

If you open the Capsule Run Log icon after the Capsule processing is finished, you might see a status report like the one shown in Figure 24 on page 33.

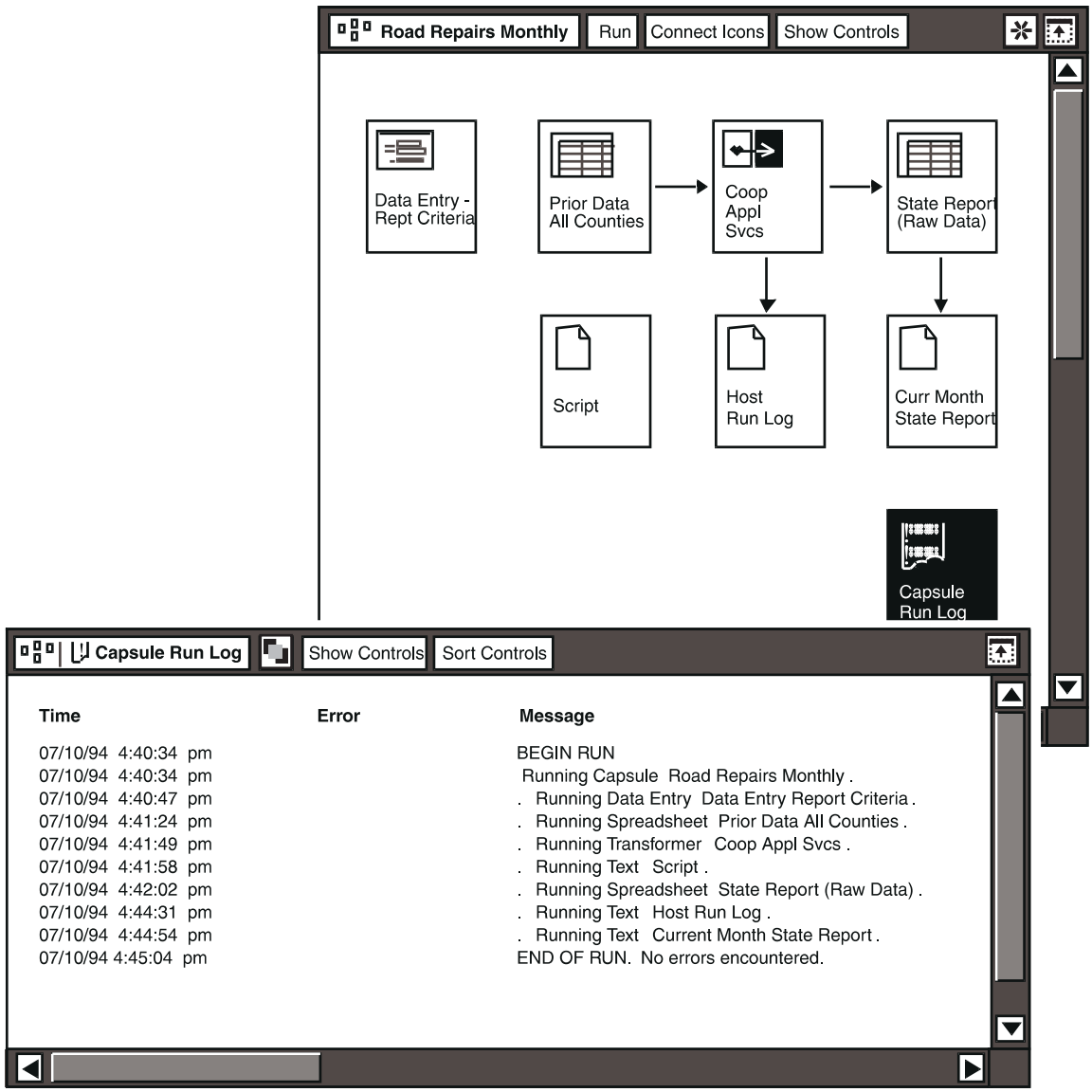


Figure 24. A Capsule Run Log after a successful capsule run

The information in the Capsule Run Log icon is only about how icons in the capsule application ran. It does not bring in any data on the host run. By using both your host run log and your Capsule Run Log, you can get more help in diagnosing any processing problems you encounter with CAS.

Please refer to *Capsule User's Guide* for detailed information on the Capsule Run Log and other features of the Capsule tool.

Stopping a Capsule Process

If you must stop a Capsule process while it is running, press the Stop key on your keyboard.

There are be several reasons for stopping a capsule application during processing. For example, you might have started running an application and discovered that host resource contention is too high, or turnaround time too long, to process your application immediately. Or, you might want to stop the processing to correct a script, or to add or delete a component from the Capsule window.

Whenever you deliberately stop a capsule application, be sure to start over from the beginning. Any data retrieved before you press Stop will be incomplete.

If processing is being done on the host system at the moment the Stop key is pressed, a run log is returned to the last output region of the CAS transformer, and then is transferred to any Text icon you have set up to receive the run log.

Using Stop can, on occasion, produce some of the same results as an application failure, including the failure of the run log to move from the transformer's output region to the log icon. If you must use Stop, be sure to check both the source output region and the target output icon for run data.

You can provide this information to users of any application in which you include CAS.

The capsule run log, described briefly in the foregoing pages, records both system stops and user stops. However, all host run data is sent to the host run log. To determine the cause of a processing problem, you might need to consult both logs.

Capsule Services

Would you like to be able to submit a Capsule job to run and then, while it is running, either do other work using your desktop or log off your desktop? If so, you should use the Capsule Service tool. Instead of clicking on the **Run** button in your Capsule icon's window header, which locks your desktop during processing, you close the Capsule icon and copy it to a Capsule Service icon. By doing this, you submit the job to run on the Capsule server.

A job submitted to the Capsule server can be scheduled for a specific time, or held in the job queue. Whether it is in the queue or running, a job continues toward completion regardless of whether you have logged off the desktop.

For details on Capsule Services, please refer to *Capsule User's Guide*.

Capsule Summary

The Special menu in the Capsule window header includes the choice Create Capsule Summary, shown in Figure 25 on page 35.

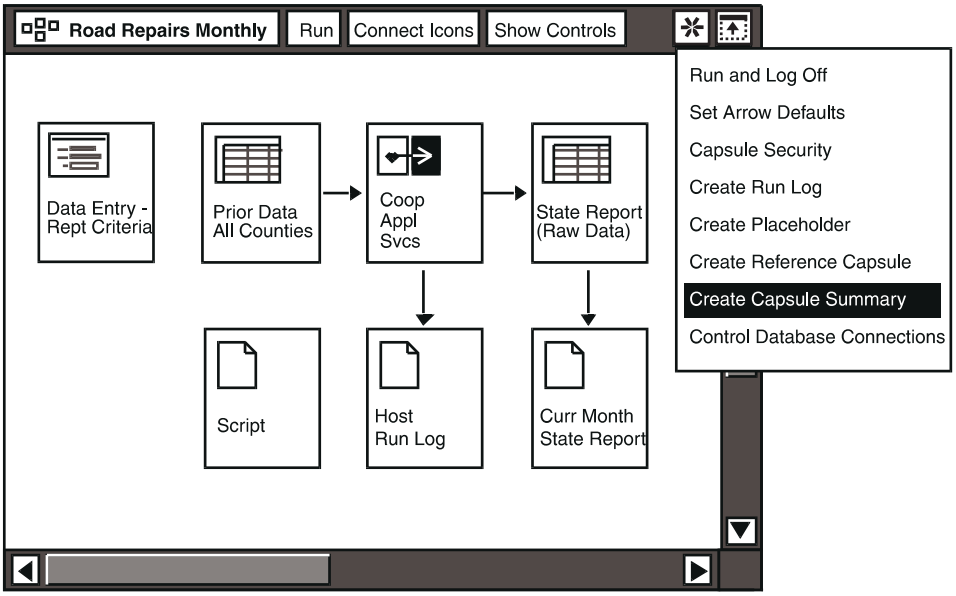


Figure 25. Capsule icon showing Special menu and Create Capsule Summary option

The Capsule Summary option provides you automatic documentation of your capsule application. Besides error messages associated with each icon, the Capsule summary offers you:

Information about individual icons (their names, types, creation and last-modification dates, and size)

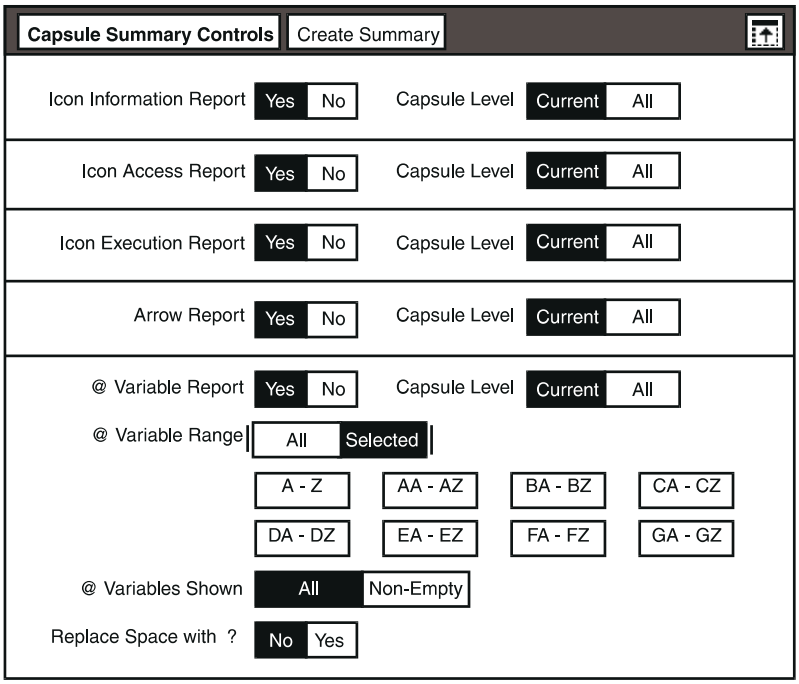
Security information (owner's name, levels of access, and security status for each icon)

Execution order and database connection settings for each icon

Arrow information (source and destination icons for each arrow, regions, transfer types, and stop settings)

@-variable information (**User Input Control** fields used in the capsule application, and the icon that uses each variable)

When you select **Create Capsule Summary**, this window appears:



In this window you can choose the summary information you want for each capsule run.

For more detailed information on the Capsule Summary feature, refer to *Capsule User's Guide*.

Chapter 3. Using the API

In this chapter are guidelines for building command scripts using the application program interface (API) that comes with Cooperative Application Services (CAS).

First come a few general observations about the API:

"The General Purpose of Each API"

"The API and REXX" on page 38

"The Usual Order of API Functions" on page 38

"API Return Codes" on page 38

Following that, you will find a description of each API function and instructions for invoking it on "API Syntax and Use" on page 39.

The functions are listed in alphabetical order for your convenience in referring to them.

At the end of the chapter, the following relevant details are discussed:

"Defining Data Format" on page 42

"Converting Data Types" on page 46

"Specifying Header Rows" on page 47

The General Purpose of Each API

As was mentioned in Chapter 2, "Building and Running a Simple Application" on page 6, you use the Transformer Controls window to associate a script with a CAS transformer. In a script (or within associated routines, which are shown in Appendix B, "Sample Supporting Routines" on page 98), you can invoke six different functions, four of which refer to input and output regions of the transformer. The functions fall into three main groups, as shown in Table 3.

Table 3. General purposes of the API functions

Function Name	General Purpose
METFGET METFOPEN METFPUT METFCLOS	Assist in transferring data back and forth between the IDS desktop and the host
METMGET	Obtain error information from the previous API call
METMDSP	Post messages to the desktop

© Copyright IBM Corp. 1996

The API and REXX

As was mentioned in “Writing Scripts with REXX” on page 14, you can use CAS without writing your scripts in the REXX language. However, the functions of the API must be invoked from a REXX procedure. If your script is written as a CLIST, for example, then you must call a REXX procedure from the CLIST to issue API calls.

The Usual Order of API Functions

When you are writing a script, you will probably find that you use four of the functions in this order:

Function	Specific Purpose
METFGET	Get an input region
METFOPEN	Open an output region
METFPUT	Put results in the output region
METFCLOS	Close an output region

Two of these functions, METFOPEN and METFCLOS, can be optional; see “METFCLOS—Close an Output Region” on page 39 and “METFOPEN—Open an Output Region” on page 40 for details.

You can use the other two functions, METMGET and METMDSP, in any order.

API Return Codes

Each function, when called in a script, initiates a particular action and then returns a code showing the result of the call, as shown in Table 4.

Table 4. API return codes

Return Code	Meaning
0	Function completed successfully
4	Function completed with a warning
8	Function failed to complete successfully

For example, you might invoke the function METFGET in a REXX script as follows:

```
IF METFGET(1,INPUT1,IXF)> THEN
    SIGNAL error
ELSE
      instructions
    ...
error:
      instructions
```

This call obtains the data in input region 1 of the transformer (the first 1 in parentheses) and writes it in IXF format to the host-resident file identified as INPUT1. It then sets a value that corresponds to the completion status of the function, and that can be used in evaluating the IF... THEN clause.

If a MET function call does not complete successfully, you can obtain an error message by using METMGET. You can also find further error information in the run log, which goes to the CAS transformer's output-log region, and from there, if you wish, to a Text icon that you have set up to contain the output log.

API Syntax and Use

Next are descriptions and syntax for each API function. Please notice two important points:

The following syntax examples often include quotation marks, according to REXX usage rules. The quotes can be either single or double, but they must be consistent within a function call.

When you are coding MET function calls, remember that no space is allowed between the function name and the left (opening) parenthesis. In the following example, METFGET is the function name:

```
IF METFGET(1,INPUT1,IXF)> THEN
```

METFCLOS—Close an Output Region

The METFCLOS function closes a specified output region of the transformer. Generally, METFCLOS is optional: After the command script has been processed, CAS automatically insures that all regions have been closed. However, you may need to invoke METFCLOS to close an output region because earlier you used METFOPEN or METFPUT with the 'NOCLOSE' option.

See “METFPUT—Write a File to an Output Region” on page 40 for more information on the 'B'NOCLOSE' option.

Syntax

METFCLOS(n

where

n specifies the number of the output region to close.

METFGET—Get an Input Region

The METFGET function opens the specified input region of the transformer, pulls the entire input region from the desktop into a file on the host, and closes the input region again. In the function call, you include the name of the resulting host file; the format of the file is specified by the data description or the *'host format'* value.

The amount of space available in the transformer's input regions depends on the amount of space available on your workstation.

Syntax

METFGET(*n,fileid,*{'*data description*'|'*host format*'})

where:

n	specifies the number of the input region to be retrieved.
fileid	specifies the name of the host file, which is a DD name (data definition name) for MVS.
'data description'	specifies the format of the resulting host file. (You must use quotation marks, either single or double, but consistently.) The format of the resulting file can be defined explicitly, column by column, with a data description. The syntax for data descriptions is in "Defining Data Format" on page 42.
'host format'	specifies the format of the resulting host file. (Use quotation marks as explained above.) MVS supports IXF format.

Remarks

For MVS, the file must be allocated before the METFGET call. This is done through the TSO ALLOCATE command, or by adding a DD name to the host service address-space JCL.

METFOPEN—Open an Output Region

The METFOPEN function opens a specified output region of a CAS transformer. It is optional: If you do not invoke it before the METFPUT function, METFPUT itself will open the output region.

Syntax

METFOPEN(*n*)

where

n	specifies the number of the output region to be opened.

Remarks

You may choose to use METFPUT in most or all cases, instead of METFOPEN; or you may prefer to invoke METFOPEN and METFCLOS functions symmetrically in your code.

METFPUT—Write a File to an Output Region

The METFPUT function writes all the data identified by fileid to a specified output region of a CAS transformer. If a METFOPEN function for output region n has not already been invoked, METFPUT will open the region.

Using the optional keyword NOCLOSE keeps the output region open after the file has been written to it; omitting the keyword allows the region to close. If an output region was left open by a previous NOCLOSE, a METFPUT to that same region appends the fileid data to the data already there in the region. If an output region is not already open, METFPUT insures that, when processing is complete, the output region will contain only the fileid data.

The amount of space available in the transformer's output regions depends on the available space on your workstation.

Syntax

METFPUT(*n*,*fileid*['CLOSE' | 'NOCLOSE'] ,'*data description*' | '*host format*')

where

n	specifies the number of the target output region.
fileid	specifies the name of the host file. For MVS, this is a DD name (data definition name).
'CLOSE'	closes the target output region after this operation. When 'CLOSE' is used, any later METFPUTs for that same region will overwrite the region, rather than appending data to it. 'CLOSE' is the default.
'NOCLOSE'	keeps the target output region open after this operation. This operand is the opposite of 'CLOSE': When 'NOCLOSE' is specified, any later calls to METFPUT will append data to the region rather than overwriting its contents. CLOSE is the default.
'data description'	states the format of the host file. You can define the format explicitly, column by column, by specifying a data description. The syntax for data descriptions is in “Defining Data Format” on page 42.
'host format'	specifies the format of the host file. MVS supports IXF format.

METMDSP—Display Important Message

The METMDSP function sends a message to the Important Message window of the desktop that initiated the application on the host.

In keeping with its purpose, the Important Message space is somewhat limited. You can send longer and/or more technical messages to the run log. For more information, see “Directing Output” on page 24 and “Host Run Log” on page 27.

Syntax

rc = METMGET('message')

if rc =

then rc = METMDSP(*message*)

where

message specifies the variable containing the message that is to be sent to the desktop's Important Message window. When you are using METMDSP, do not enclose the variable in quotation marks.

METMGET — Get CAS Error Message

The METMGET function obtains an error information message, if one is available, from the last call made to an API function. If a message is not available, METMGET returns a value of 4 (“function completed with a warning”).

If more or less than one parameter is specified, METMGET returns a value of 8.

Syntax

rc = METMGET('*message*')

if rc =

then say 'Error message was' message

where

message specifies the variable that contains the message.

When specifying the variable in the METMGET function, enclose it in quotation marks to ensure that variable substitution is not performed before METMGET is invoked. When you are using the variable later in the script, omit the quotation marks.

Defining Data Format

In the functions METFGET and METFPUT, you can either specify the self-describing format of a host file (for CAS, it is IXF) or describe the file's data format explicitly.

The next section shows the syntax for specifying IXF format.

After that is a section showing details on defining data format explicitly.

IXF Format Syntax

`IXF[,maxcolsize=*n*] [,MODE= CHAR / MACHINE]'

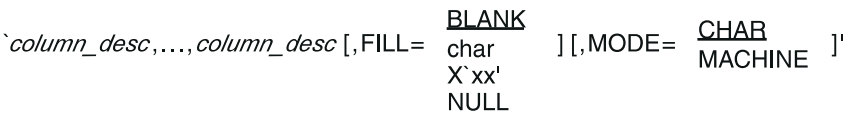
where

n specifies the maximum column size for a character column coming from the desktop. The default is 255.

This parameter has no meaning for IXF files that are sent from the host.

Data Format Syntax

A data format description consists of one or more column descriptions separated by commas, including an optional FILL or MODE operand.



where

column_desc specifies the data type and starting position, and, if needed, the length and column name, using the following format:

type[(*qualifier*)] *starting_position*[*length*][*column_name*]

Each operand functions according to Table 5. For examples of use, see the REXX scripts supplied along with the sample applications in Appendix A, “Sample Capsule Applications” on page 48.

FILL specifies a one-byte value that fills all undefined positions within the record. The value assigned as the fill character can be expressed as a character, as a HEX value in the form X'xx', or by either the word NULL or the word BLANK. It is used only by METFGET, and it is ignored by METFPUT.

FILL is optional. If it is not specified, BLANK is used as the default value. FILL can be abbreviated to FI or FIL.

MODE specifies whether the format of the data is in character mode or machine mode. Character mode means that the data in the file is represented by character strings. Machine mode means that it is represented by hardware-specific bits. (See the specific data types for details.)

MODE is optional. If it is not specified, then the data is assumed to be in character mode (CHAR is the default). MODE can be abbreviated to M, MO, or MOD.

Remarks

To abbreviate a type value, you can use any unique combination that is at least two characters long. TS can be used as an abbreviation for TIMESTAMP; TI, TIM and TIME specifically mean TIME. The data description is not case-sensitive.

Table 5 (Page 1 of 3). Column-description operands

Operand Value	Description
type[(*qualifier*)]	

Table 5 (Page 2 of 3). Column-description operands

Operand Value	Description
;DATE(*qualifier*)	The mandatory qualifier describes the format of the DATE character string.
	Format. Any arrangement of *yyyy* or yy for year, *mm* for month, and *dd* for day, with or without separators
	Separators allowed. /, -, and . (period)
	Default length. 10 (minimum: 6, maximum: 10)
	Machine- and character-mode representations are the same.
;TIME	Format. *hh.mm.ss* or *hh:mm* AM or PM.
	Default length. 8 bytes (minimum: 5, maximum: 8).
	Machine- and character-mode representations are the same.
;TIMESTAMP	Format. *yyyy-mm-dd-hh.mm.ss.nnnnnn*.
	Default length. 26 bytes (minimum: 16, maximum: 26).
	Machine- and character-mode representations are the same.
;VARCHAR	A variable-length string headed by a length field. The length must be specified.
	Minimum data length. 1 byte
	Maximum data length. 254 bytes
	When you specify the length, you must include the number of positions taken up by the length field. For machine-mode representations, this is 2 bytes; for character-mode representations, 5. Therefore the minimum length of total field is 3 (machine mode) or 6 (character mode). The maximum field length is 256 (machine) or 259 (character).
;CHAR	A string of maximum length 254. The length must be specified.
	Minimum length. 1
	Machine- and character-mode representations are the same.
;FLOAT	Machine mode. A 4-byte single-precision (or 8-byte double-precision) floating-point number. The length must be 4 or 8. Single-precision numbers can go up to 14 digits; double-precision numbers can go up to 28.
	Character mode. A character string between 5 and 23 bytes long, formatted in floating-point representation, for example, 9.9E9. A 23-character float contains one character for sign, 17 digits, one decimal point, the character E, and a three-character signed exponent. Range: from (5.4x10-79) to (7.2x1075).

Table 5 (Page 3 of 3). Column-description operands

Operand Value	Description
;DECIMAL(qualifier)	Machine mode. IBM packed-decimal format. Character mode. A character string with a format such as 9.9. A qualifier in the form (*w* [,*d*]) specifies decimal format, where: *w* is the field width in digits. *d* is the number of implied digits to the right of the decimal point. If *w* is omitted, 5 is assumed. Minimum value for *w* is 1 and the maximum is 15. If *d* is omitted, 0 is assumed. The value of *d* must be less than the value of *w*. The length specification is optional; the true length is computed from (*w* [,*d*]). The true length is *w*+1 if *d* is 0, or *w*+2 if *d* is non-zero. It is an error to specify a length that differs from what the format specifier implies.
;INTEGER	Machine mode. A 4-byte binary value; only length permitted: 4. Character mode. A character string between 1 and 11 bytes long. Values can range between -2147483648 and 2147483647.
;SMALLINT	Machine mode. A 2-byte binary value; only length permitted: 2. Character mode. A character string between 1 and 6 bytes long. Values can range between -32768 and 32767.
starting_position	The starting position of data in the flat file record. The first byte of data in the record is in position 1.
length	The length of the data. Optional for DATE, TIME, TIMESTAMP, INTEGER, SMALLINT, and DECIMAL; required for CHAR, VARCHAR, and FLOAT. For DATE, TIME, and TIMESTAMP types: If the specified length is less than the specified pattern length, right truncation occurs. If the specified length is greater than the specified pattern length, the field is padded with blanks. For VARCHAR: A 2-byte binary count (machine mode) or a 5-character integer count (character mode) is inserted in the record at the position indicated by the *starting_position* operand. Characters immediately follow the count information. To describe data of type VARCHAR, length specification must be 2 (or 5) more than the actual number of characters in the longest string.
column_name	The name associated with the column being described. This operand is optional, and has meaning only when used in a METFPUT function call. If the operand is omitted, the column name is set to blanks.

Converting Data Types

When data going between the desktop and the host is different from the type specified in the column description, the data type is converted. For example, in a METFGET call, an integer value is contained in the first column of a spreadsheet row, but the application requires this data to be in character format in the flat-file record. When you specify CHAR as the column type, METFGET converts the (binary) integer to EBCDIC format.

Not every type can be freely converted. The following rules apply:

Any numeric type can be converted to either CHAR or VARCHAR.

CHAR can be converted to VARCHAR, and vice versa.

INTEGER can not be converted to another numeric type.

DATE can not be converted to any other type.

These rules are presented in the next two tables. Note that, for the purposes of these tables, some data type names are abbreviated.

Table 6 shows the three data types that can originate on the desktop, and to which data types on a host file they can be converted. They might be transferred to the host, for example, in a METFGET call.

Table 6. METFGET-type data conversion (from desktop to host)

Data Type Coming from WS	Data Type Going to Host File								
	CHAR	VCHAR	FLOAT	DEC	INTGR	SMLINT	DATE	TIME	TS
CHAR	x	x	x	x	x	x		x	x
NUMBER	x	x	x	x	x	x			
DATE							x		

Table 7 shows conversion in the opposite direction: the nine data types that can originate on the host, and to which data types on the desktop they can be converted.

Table 7 (Page 1 of 2). METFPUT-type data conversion (from host to desktop)

Data Type Residing on Host	Data Type Going to Desktop		
	CHAR	NUMBER	DATE
CHAR	x		
VCHAR	x		
FLOAT		x	
DECIMAL		x	
INTEGER		x	
SMALLINT		x	
DATE			x

Table 7 (Page 2 of 2). METFPUT-type data conversion (from host to desktop)

Data Type Residing on Host	Data Type Going to Desktop		
	CHAR	**NUMBER**	**DATE**
TIME	x		
TIMESTAMP	x		

In a METFPUT function call, the column description operands define the records of the flat file as it currently exists on the host, and there is no option to convert these columns to different types before the file is sent to the desktop.

For example, if you have a column of numbers in character format on the host file, and you want to send it to the desktop as numeric data, then you must define that column as having a numeric data type (floating point, decimal, integer, or small integer). Otherwise, the column will arrive at the desktop as a character string, and you will not be able to treat it as a number.

Remember that you can specify MODE=CHAR to describe numeric data types that are represented as character strings.

Specifying Header Rows

The CAS transformer can read data coming into its input and output regions as data for calculation, or as header rows (column names, not to be included in calculations), depending on the type of data it encounters. To make sure that your data is treated as you intend, observe the rules explained in Table 1 on page 27.

Appendix A. Sample Capsule Applications

Since Cooperative Application Services (CAS) makes host processing readily available from your desktop, you can use the product to put together applications for many different purposes. Chapter 2, “Building and Running a Simple Application” on page 6 explains how to build a basic Capsule application that includes CAS. This appendix contains several sample Capsule applications to suggest the variety of ways this product can be useful to you.

This appendix focuses on highlights of each sample application. However, each application is provided for you in its entirety on your Intelligent Decision Server (IDS) desktop. Your system administrator can tell you where to find these samples.

The following applications are discussed in this appendix:

“Sample 1: A File Transfer Routine”

“Sample 2: A Flat-File Filter Application” on page 53

“Sample 3: A Cross-Tabular Analysis Using SAS” on page 62

“Sample 4: A Regression Analysis Using DB2, DXT, and SAS” on page 73

Example programs and code are provided only for illustration. You may need to modify any code that you choose to use in your environment.

Sample 1: A File Transfer Routine

The first sample application demonstrates reading from and writing to the input and output regions of a CAS transformer. This sample Capsule application contains:

A Text icon named **Operating Instructions** for the user to open and read before running the Capsule application for the first time.

A Spreadsheet icon called **Input Data**, containing data that will be transferred through the transformer to the host system.

A CAS transformer.

A Text icon, **METEX00**, containing a REXX script.

A second Spreadsheet icon named **Output Data**. It will receive data that has been returned from the host and placed in the CAS transformer's last output region. After the Capsule application has been run, this icon contains the finished data, but not the finished report.

A second Text icon called **Host Run Log**, reserved for system output returning from the host through the last of the transformer's output regions.

In Figure 26 on page 50, the parts of the Capsule icon that this sample focuses upon are called out:

The REXX script, **METEX00**

© Copyright IBM Corp. 1996

Two supporting routines that reside on the host system, and that are called by the script to perform certain functions within the host system for the Capsule application

After the figure, the purpose of each routine is discussed. The actual routines are provided with the product, as well as in hardcopy form in Appendix B, “Sample Supporting Routines” on page 98.

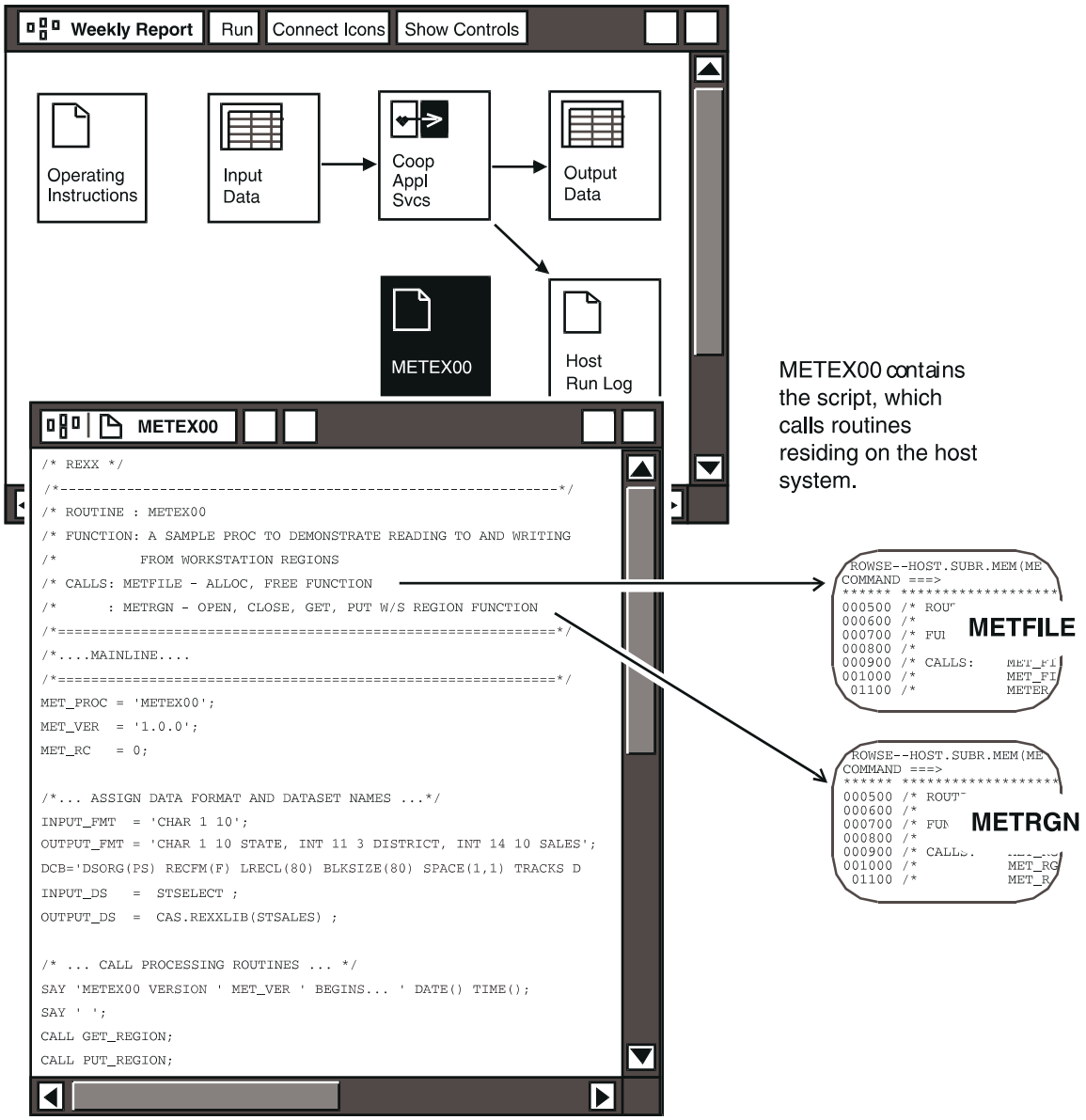
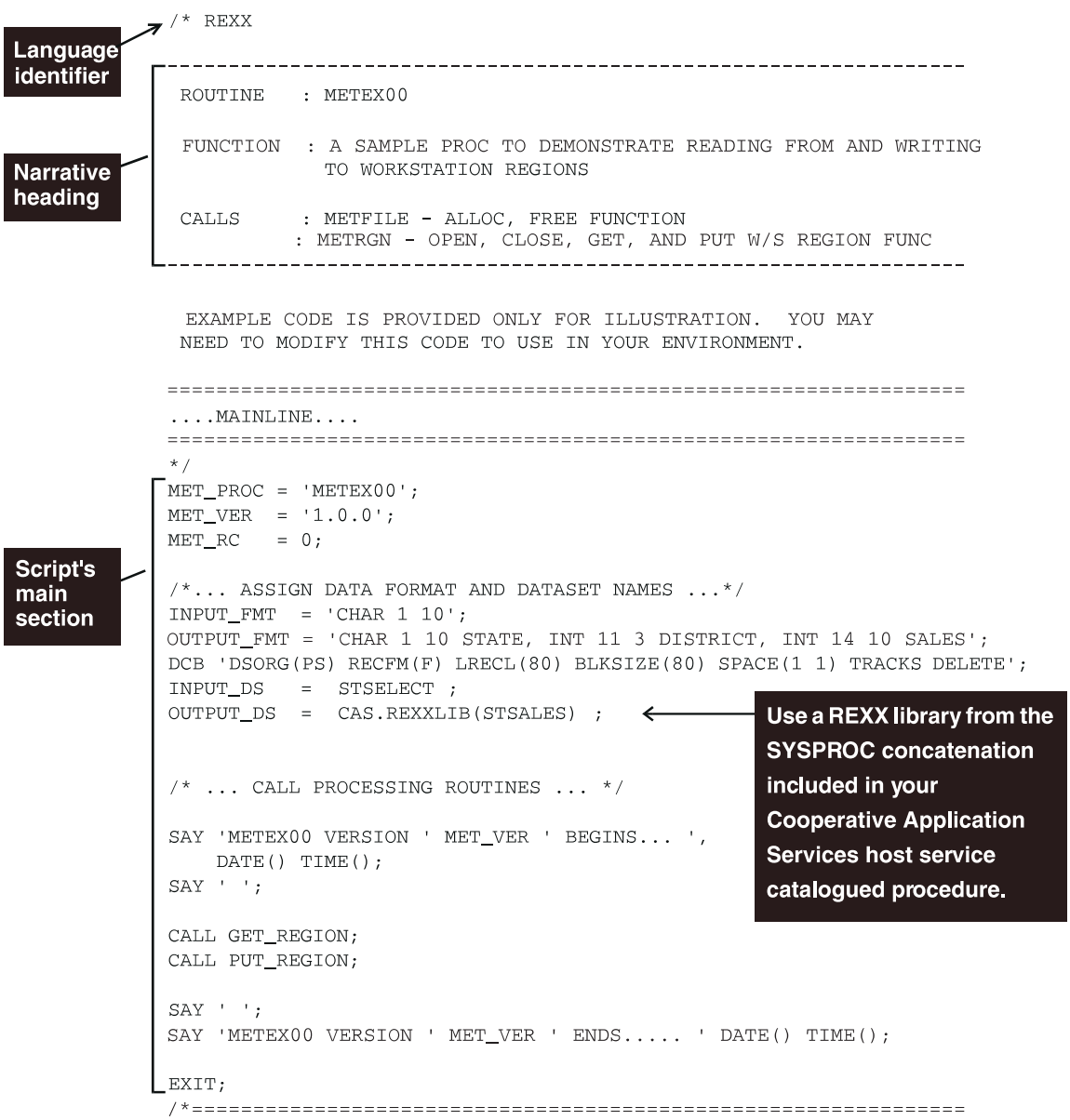


Figure 26. Sample 1: Capsule window showing script and supporting routines

The following sample shows the entire REXX script that processes the application.

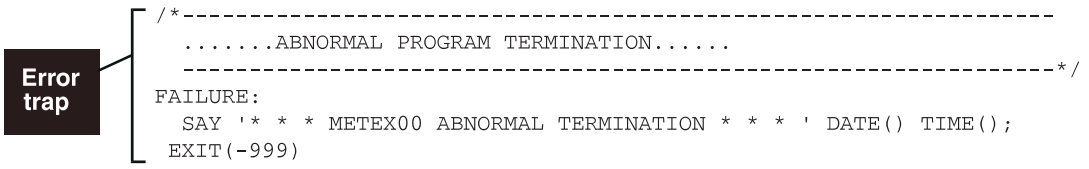

Language identifier
Narrative heading
Script's main section
/* REXX
ROUTINE : METEX00
FUNCTION : A SAMPLE PROC TO DEMONSTRATE READING FROM AND WRITING
TO WORKSTATION REGIONS
CALLS : METFILE - ALLOC, FREE FUNCTION
: METRGN - OPEN, CLOSE, GET, AND PUT W/S REGION FUNC
EXAMPLE CODE IS PROVIDED ONLY FOR ILLUSTRATION. YOU MAY
NEED TO MODIFY THIS CODE TO USE IN YOUR ENVIRONMENT.
....MAINLINE....
*/
MET_PROC = 'METEX00';
MET_VER = '1.0.0';
MET_RC = 0;
/*... ASSIGN DATA FORMAT AND DATASET NAMES ...*/
INPUT_FMT = 'CHAR 1 10';
OUTPUT_FMT = 'CHAR 1 10 STATE, INT 11 3 DISTRICT, INT 14 10 SALES';
DCB 'DSORG(PS) RECFM(F) LRECL(80) BLKSIZE(80) SPACE(1 1) TRACKS DELETE';
INPUT_DS = STSELECT ;
OUTPUT_DS = CAS.REXXLIB(STSALES) ;
Use a REXX library from the SYSPROC concatenation included in your Cooperative Application Services host service catalogued procedure.
/* ... CALL PROCESSING ROUTINES ... */
SAY 'METEX00 VERSION ' MET_VER ' BEGINS... ',
DATE() TIME();
SAY ' ';
CALL GET_REGION;
CALL PUT_REGION;
SAY ' ';
SAY 'METEX00 VERSION ' MET_VER ' ENDS..... ' DATE() TIME();
EXIT;
/*

```
/*-------------------------------------------------------------------
  .....GET REGION 5 FROM WS CONTAINING SELECTED STATE NAMES ....
  -------------------------------------------------------------------*/
GET_REGION:
 SAY '---> GET INPUT REGION';
 'NEWSTACK';

/*... FREE INPUT TARGET DATASET ...*/
 MET_RC = METFILE('FREE','SELECTOR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*... ALLOCATE INPUT TARGET DATASET ...*/
 MET_RC = METFILE('ALLOC', 'SELECTOR',INPUT_DS,DCB);
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*...COPY WORKSTATION REGION 5 TO TARGET DATASET ...*/
 MET_RC = METRGN('GET','5','SELECTOR',INPUT_FMT,'CHAR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*... FREE INPUT TARGET DATASET ...*/
 MET_RC = METFILE('FREE','SELECTOR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 'DELSTACK';
RETURN;
```

Move data from transformer input region to host data set.

```
/*-------------------------------------------------------------------
  ...PUT HOST FILE CONTAINING SELECTED DATA TO OUTPUT REGION 2...
  -------------------------------------------------------------------*/
PUT_REGION:

 SAY '---> PUT OUTPUT REGION';

 'NEWSTACK';

/*... FREE SOURCE DATASET ...*/
 MET_RC =  METFILE('FREE','SALES');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*... ALLOCATE SOURCE DATASET ...*/
 MET_RC = METFILE('ALLOC', 'SALES',OUTPUT_DS,'SHR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*... COPY SOURCE DATASET TO OUTPUT REGION 1 ...*/
 MET_RC = METRGN('PUT','1','SALES','CLOSE',OUTPUT_FMT,'CHAR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

/*... FREE SOURCE DATASET ...*/
 MET_RC = METFILE('FREE','SALES');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 'DELSTACK';
RETURN;
```

Move data back into transformer output region.



The script in this sample calls three related routines: **METFILE**, **METRGN,** and **METERROR**. Notice that **METFILE** and **METRGN** are mentioned in the script's narrative heading; each routine internally calls **METERROR**. These three routines can be placed inside the script if that is more convenient for you. However, in this case, they are outside the script and reside on the host system.

In REXX scripts that have been written for other sample applications in this appendix, you will find these and other routines used. All the supporting routines used by these sample applications are listed in Appendix B, “Sample Supporting Routines” on page 98, and copies of the routines are provided with this product.

Each routine is designed for a specific function, as described in Table 8. Two routines call the CAS API (application program interface) that is discussed in Chapter 3, “Using the API” on page 37. Note that you can also place API calls directly in a REXX script if that is appropriate to your application.

Table 8. Routines associated with a REXX script

Routine	Function
METRGN	Uses the API to open, close, get, and put workstation regions. If a call to the API fails, this routine gathers error information, writes to the host run log, notifies the workstation, and terminates with a non-zero return code. This eventually causes the CAS transformer to fail.
METFILE	Uses host functions to allocate and free data sets. If a host function fails, this routine gathers and acts on error data as does **METRGN**.
METERROR	Posts error information to the host run log and to the workstation, by means of a call to the API. This routine is called by **METRGN** and **METFILE** when a failure is detected.
METSUBMT	Submits jobs under TSO for batch processing.
METCMD	Invokes host commands and posts results to the host run log.

Sample 2: A Flat-File Filter Application

The following sample illustrates how an application can be built to query a database on the network, get a flat file from the host, filter it through criteria proposed by the user in Capsule controls, and produce a state-by-state report of sales distribution throughout the country.

Figure 27 on page 54 shows how the Capsule icon looks when it is opened.

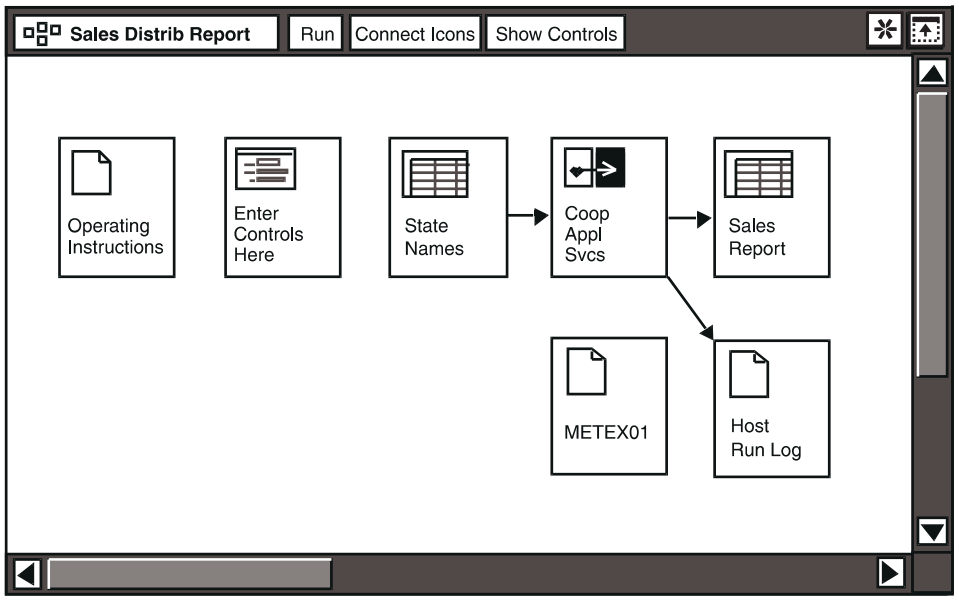


Figure 27. Sample 2: Capsule application (a monthly sales distribution report)

Using the Data Entry tool named **Enter Controls Here**, the user specifies a list of state names to be used as filtering criteria, to extract a subset of data from a sales data set (a flat file) residing on the host. The state names are edited into the **INCLUDE COND** statement of the sort control data set. The resulting extract data set is then transmitted to the workstation. At this point, the user may review the data either by opening the appropriate output region of the CAS transformer, or by opening the Spreadsheet icon named **Sales Report**, where the report has been formatted.

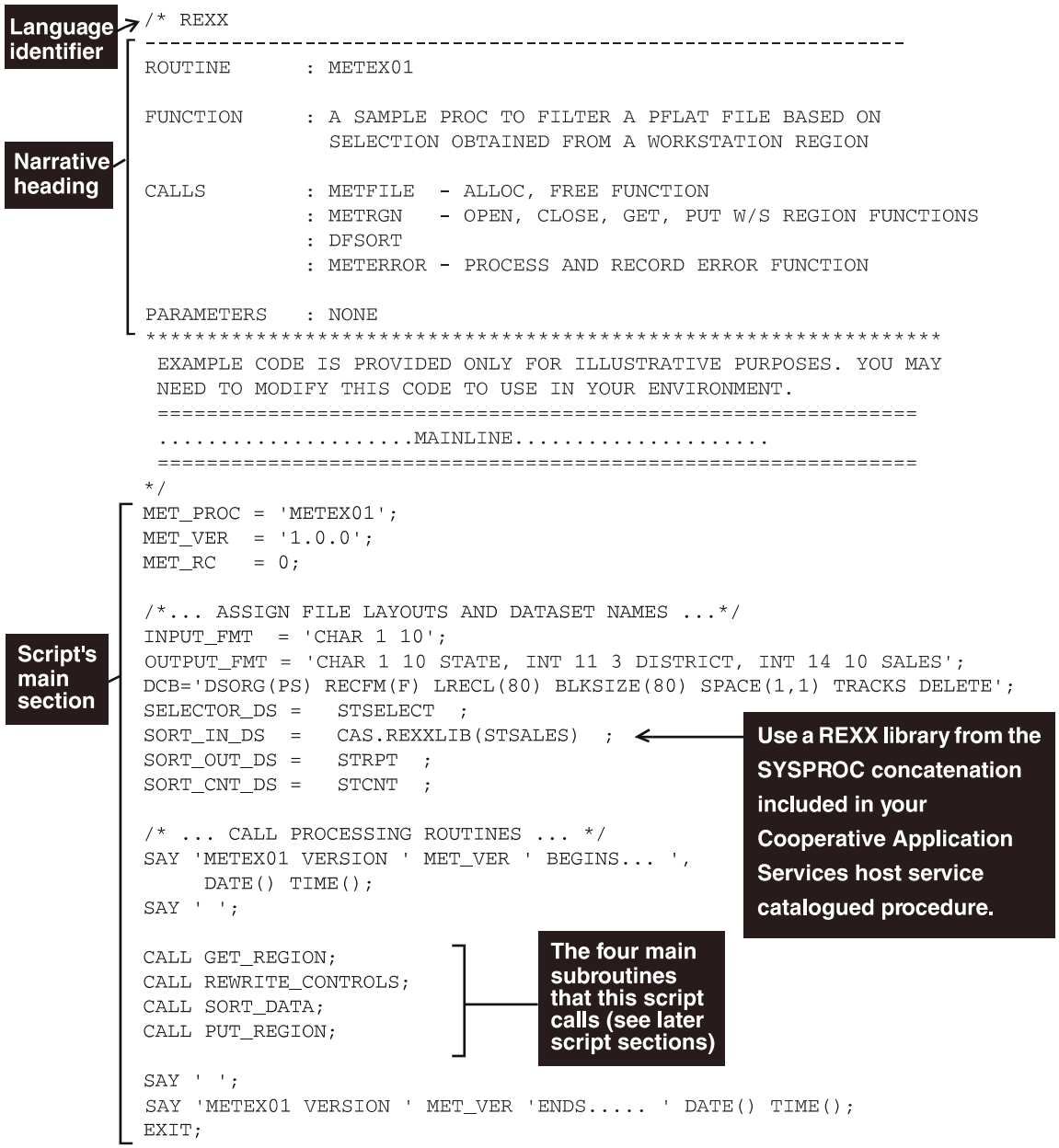
This Capsule application receives data on how the host processes have run. This data is sent from the host to the transformer's last output region and from there to the Text icon named **Host Run Log**.

In this sample you will find:

The REXX script, annotated function by function

A sample of the data that would appear in the Host Run Log icon after a successful run

Figure 28 on page 55, which spans several pages, shows the REXX script that runs this application on the host. As in Sample 1, the transformer moves a copy of the script to the host and hands over control to the script there.




```
/* REXX
------------------------------------------------------------
ROUTINE      : METEX01

FUNCTION     : A SAMPLE PROC TO FILTER A PFLAT FILE BASED ON
               SELECTION OBTAINED FROM A WORKSTATION REGION

CALLS        : METFILE  - ALLOC, FREE FUNCTION
             : METRGN   - OPEN, CLOSE, GET, PUT W/S REGION FUNCTIONS
             : DFSORT
             : METERROR - PROCESS AND RECORD ERROR FUNCTION

PARAMETERS   : NONE
************************************************************
 EXAMPLE CODE IS PROVIDED ONLY FOR ILLUSTRATIVE PURPOSES. YOU MAY
 NEED TO MODIFY THIS CODE TO USE IN YOUR ENVIRONMENT.
 ============================================================
 .....................MAINLINE.....................
 ============================================================
*/
MET_PROC = 'METEX01';
MET_VER  = '1.0.0';
MET_RC   = 0;

/*... ASSIGN FILE LAYOUTS AND DATASET NAMES ...*/
INPUT_FMT  = 'CHAR 1 10';
OUTPUT_FMT = 'CHAR 1 10 STATE, INT 11 3 DISTRICT, INT 14 10 SALES';
DCB='DSORG(PS) RECFM(F) LRECL(80) BLKSIZE(80) SPACE(1,1) TRACKS DELETE';
SELECTOR_DS =   STSELECT  ;
SORT_IN_DS  =   CAS.REXXLIB(STSALES)  ;
SORT_OUT_DS =   STRPT  ;
SORT_CNT_DS =   STCNT  ;

/* ... CALL PROCESSING ROUTINES ... */
SAY 'METEX01 VERSION ' MET_VER ' BEGINS... ',
     DATE() TIME();
SAY ' ';

CALL GET_REGION;
CALL REWRITE_CONTROLS;
CALL SORT_DATA;
CALL PUT_REGION;

SAY ' ';
SAY 'METEX01 VERSION ' MET_VER 'ENDS..... ' DATE() TIME();
EXIT;
```

Figure 28. Sample 2: REXX script to filter a flat file (1 of 6)

```
/*-------------------------------------------------------------
  .....GET REGION 5 FROM WS CONTAINING SELECTED STATE NAMES ....
  -------------------------------------------------------------*/
*/
GET_REGION:

 SAY '---> GET INPUT REGION';

 /* ... INITIALIZE VARIABLES FOR ERROR TRAP ...*/
 MET_RTN  = 'MET_PROC || ' .GET_REGION';
 MET_SRV  = 'FAILURE';
 MET_PMSG = 'GET OF SELECTOR STATES FAILED';
 MET_SMSG = 'EXECIO FAILED CLOSING ' || SELECTOR_DS;

 `NEWSTACK';

 /* ... FREE SELECTOR DATASET ...*/
 MET_RC = METFILE('FREE','SELECTOR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 /* ... ALLOCATE SELECTOR DATASET ...*/
 MET_RC = METFILE('ALLOC', 'SELECTOR',SELECTOR_DS,DCB);
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 /*...COPY WORKSTATION REGION 5 TO SELECTOR DATASET ...*/
 MET_RC = METRGN('GET','5','SELECTOR',INPUT_FMT,'CHAR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 /*... READ UPDATED HOST FILE INTO PROGRAM VARIABLE ...*/
 'EXECIO * DISKR SELECTOR (STEM STATES.';
 IF RC <> 0
    THEN STATES.0 = 0;

 'EXECIO 0 DISKR SELECTOR (FINIS';
 IF RC <> 0 THEN DO:
    CALL METERROR MET_RTN,MET_SRV,RC,MET_PMSG,MET_SMSG;
    SIGNAL FAILURE;
 END;

 /*... FREE TARGET DATASET ...*/
 MET_RC = METFILE('FREE','SELECTOR');
 IF MET_RC <> 0 THEN SIGNAL FAILURE;

 'DELSTACK';

RETURN;
```

GET_REGION
copy state names to host file

METFILE
free state-names data set

METFILE
allocate state-names data set

METRGN
copy workstation region to host file

EXECIO DISKR
read state names into script variable

Figure 29. Sample 2: REXX script to filter a flat file (2 of 6)

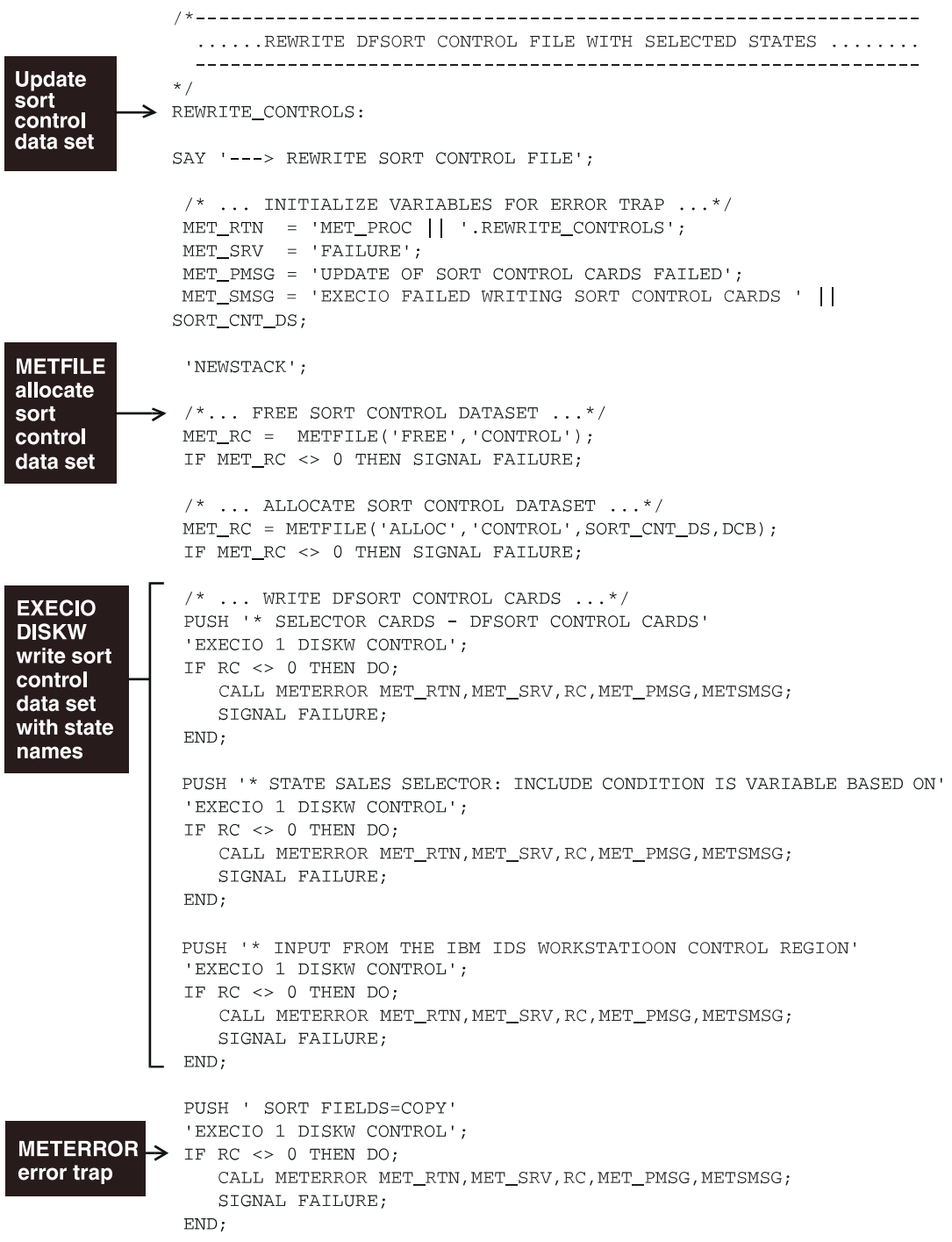


Figure 30. Sample 2: REXX script to filter a flat file (3 of 6)

```
/*... WRITE VARIABLE INCLUDE CONDITION BASED ON W/S INPUT REGION ..*/
 PREFIX.1 = ' INCLUDE COND=(1,10,CH,EQ,C';
 PREFIX.2 = ' OR,1,10,CH,EQ,C';
 QUOTE    = '''';

 DO I = 1 TO STATES.0
    IF I = 1 THEN
       BUFFER = PREFIX.1 || QUOTE \\ LEFT(STATES.I,10) || QUOTE;
    ELSE
       BUFFER = PREFIX.2 || QUOTE \\ LEFT(STATES.I,10) || QUOTE;
    IF I = STATES.0 THEN
       BUFFER = BUFFER || ')';
    ELSE
       BUFFER = BUFFER || ',';
    PUSH BUFFER;
    'EXECIO 1 DISKW CONTROL';
    IF RC <> 0 THEN DO;
       CALL METERROR MET_RTN,MET_SRV,RC,MET_PMSG,METSMSG;
       SIGNAL FAILURE;
    END;
 END;

 PUSH '*';
 'EXECIO 1 DISKW CONTROL (FINIS';
 IF RC <> 0 THEN DO;
    CALL METERROR MET_RTN,MET_SRV,RC,MET_PMSG,METSMSG;
    SIGNAL FAILURE;
 END;

'DELSTACK';
RETURN;

/*-------------------------------------------------------------
  ...SORT SALES FILE USING PREVIOUSLY UPDATED SORT CONTROLS ...
  -------------------------------------------------------------
*/
SORT_DATA:

 `NEWSTACK';

 /* ... INITIALIZE VARIABLES FOR ERROR TRAP ... */
 MET_RTN  = 'MET_PROC || '.SORT_DATA';
 MET_SRV  = 'FAILURE';
 MET_PMSG = 'SORT OF SALES DATA FAILED';
 MET_SMSG ='DFSORT FAILED:' ||,
           ';IN=' || INPUT_DS || ';OUT=' || OUTPUT_DS ||,
           ';CONTROL=' || SORT_CONTROL;
```

SORT_DATA call DFSORT to extract specified states

Figure 31. Sample 2: REXX script to filter a flat file (4 of 6)

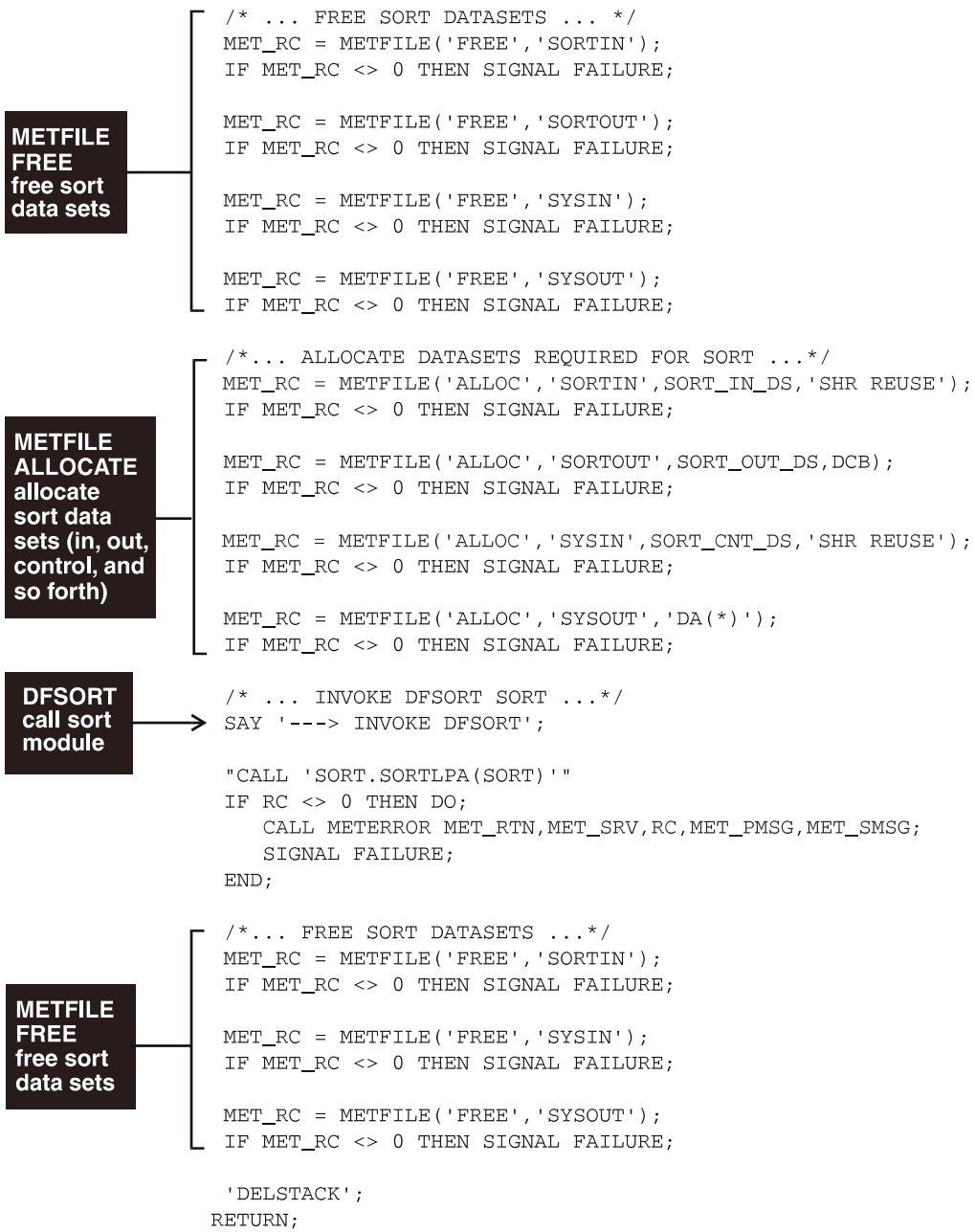


Figure 32. Sample 2: REXX script to filter a flat file (5 of 6)

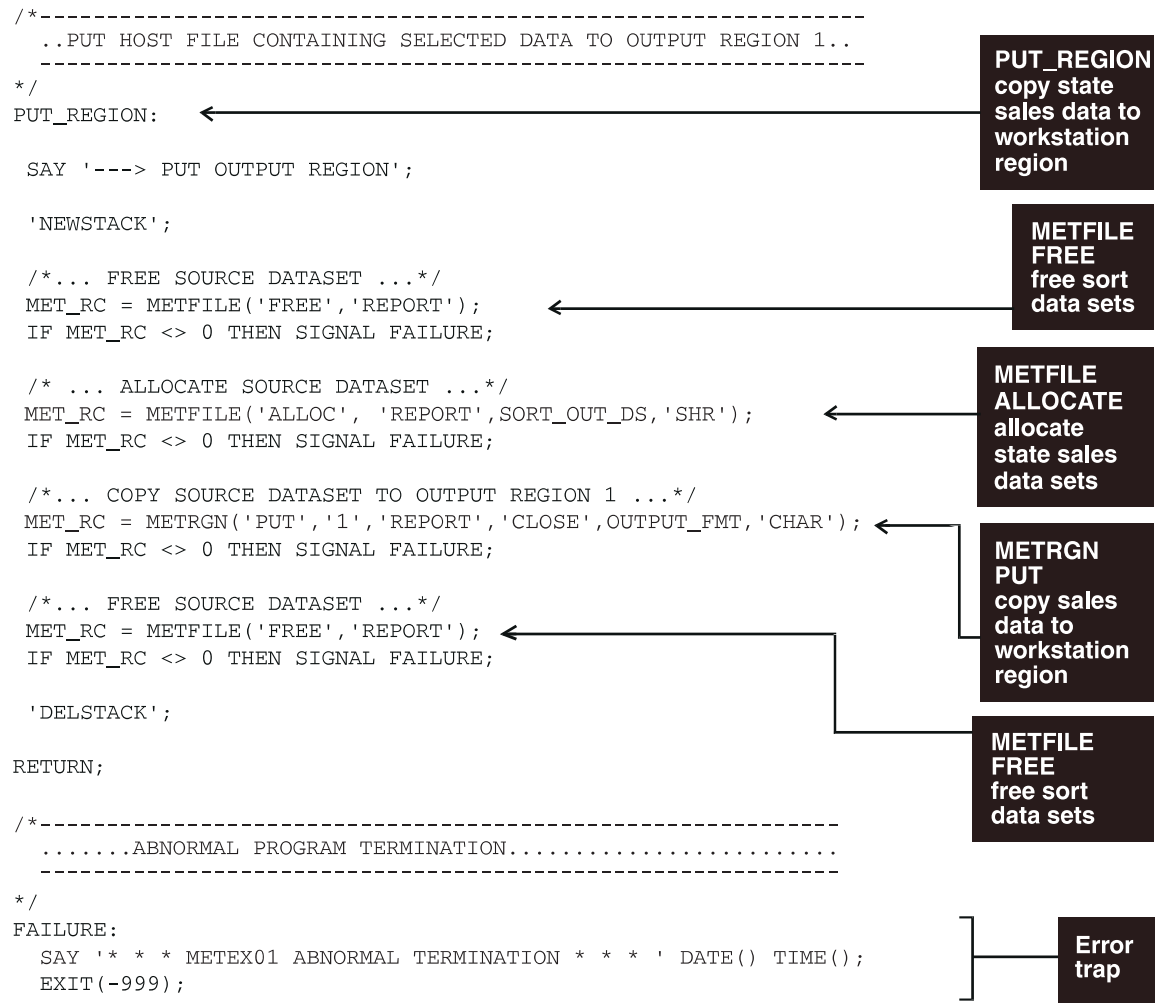


Figure 33. Sample 2: REXX script to filter a flat file (6 of 6)

Next, in Figure 34 on page 61, is a sample of the data that you would see in the Host Run Log icon after processing finished successfully.

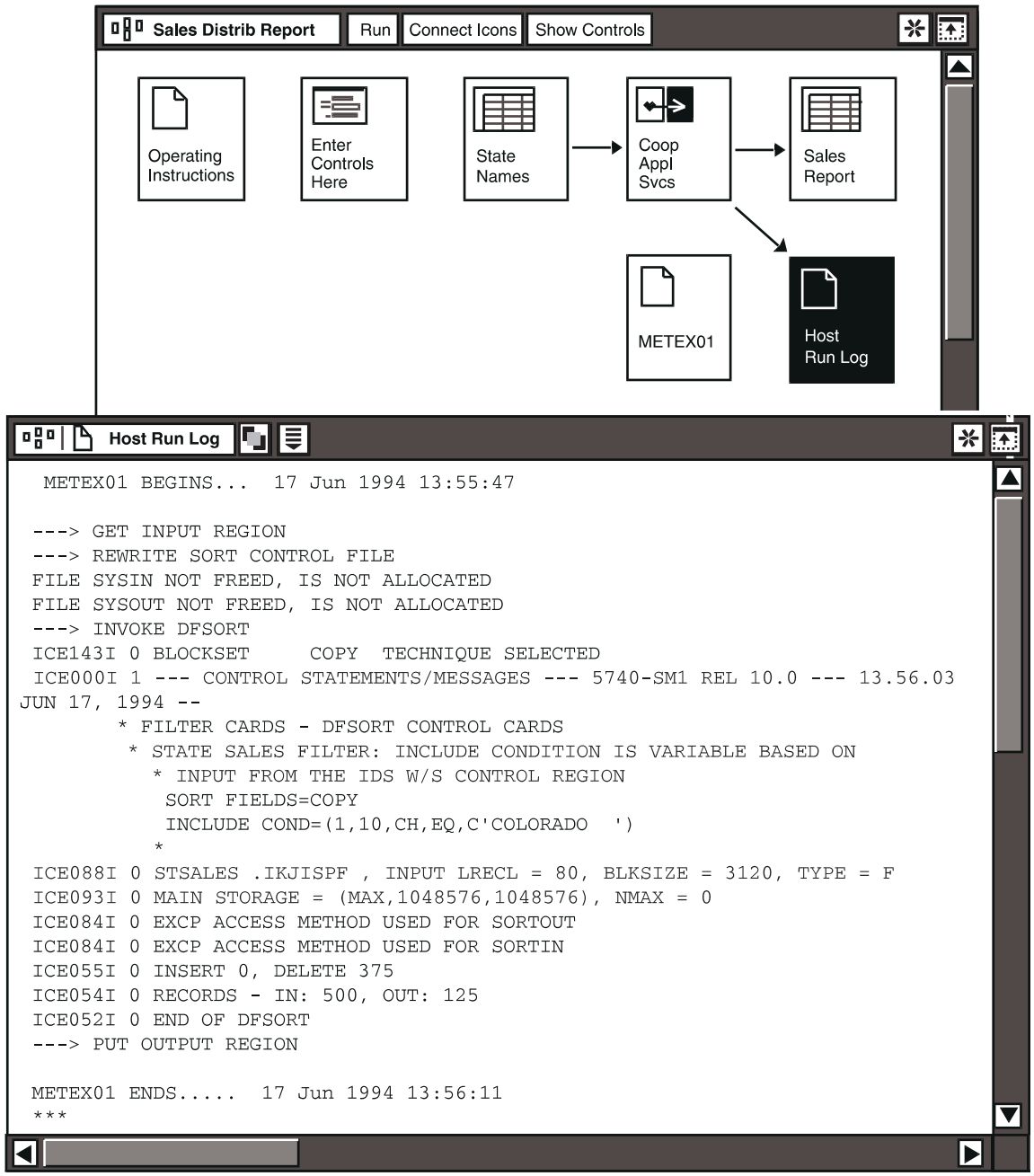


Figure 34. Sample 2: Contents of Text icon Host Run Log

Sample 3: A Cross-Tabular Analysis Using SAS

This sample demonstrates how you can use CAS to retrieve data from a remote database, process it through statistical-analysis software (SAS) that resides on an MVS host system, bring the results back to the desktop and format them into a series of IDS plots and documents. Sample 3 uses different facilities of IDS to lend the application greater power and complexity, and it gives the user several forms of output.

Figure 35 shows the Capsule icon that contains the application.

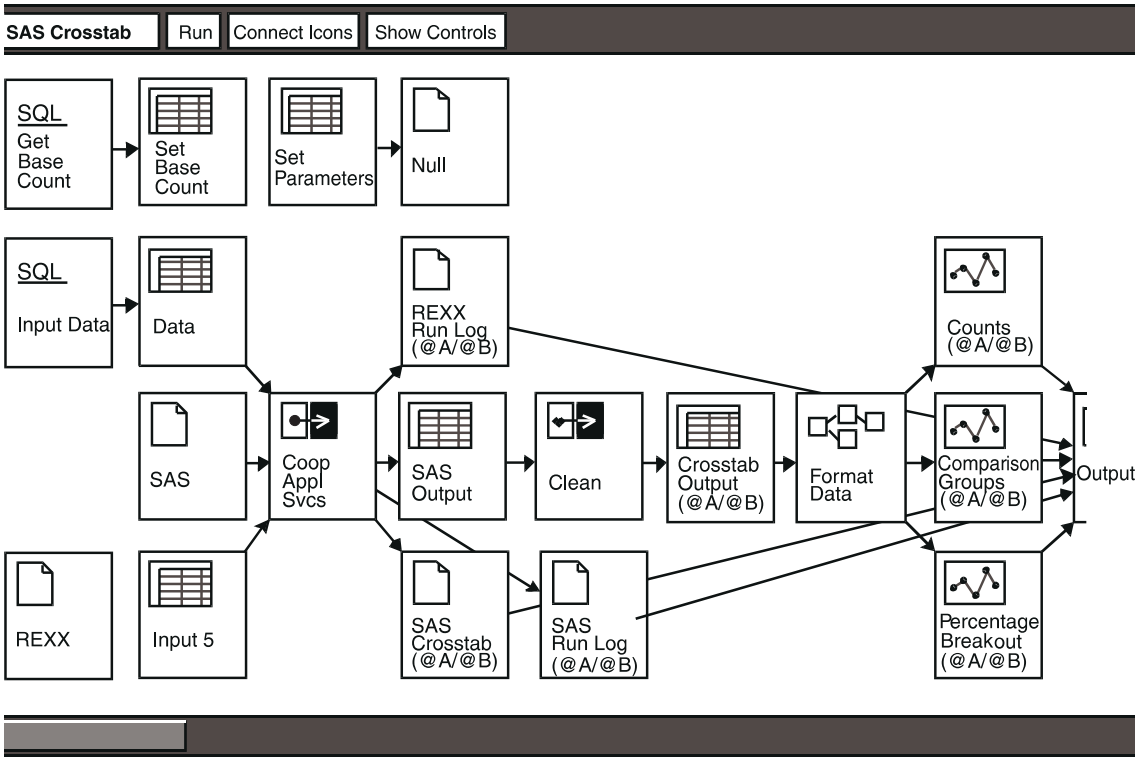


Figure 35. Sample 3: Capsule window containing the application Two-Way SAS Crosstab

An overview of the steps in the **Two-Way SAS Crosstab** process follows. In the opened Data Entry icon, **Set Choices Here**, the user enters variables: a title for the end output and criteria for the content of the report and plots. The user clicks on the **Enter** button and closes the window.

Figure 36 on page 63 shows the opened Data Entry icon.

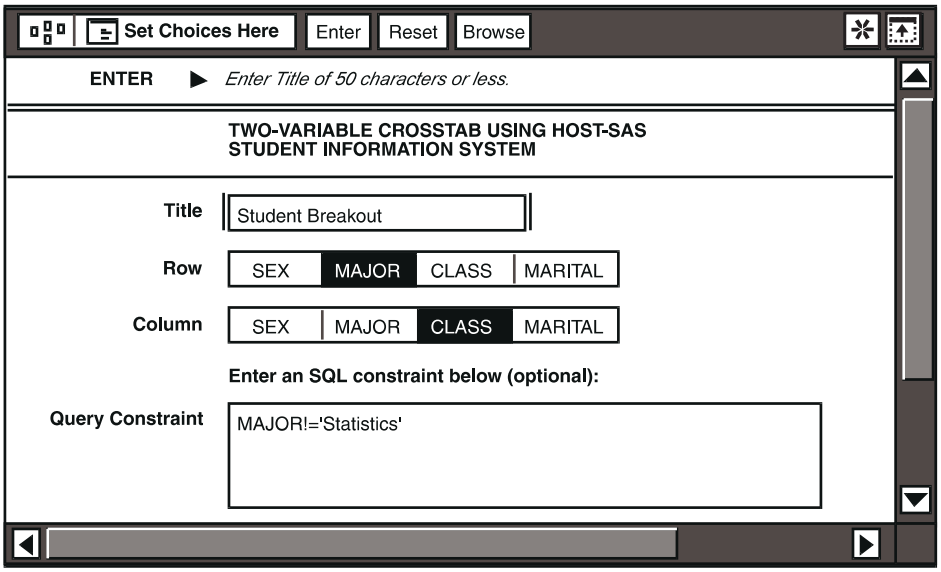


Figure 36. Sample 3: Data Entry icon named Set Choices Here

In this case the user has:

Titled the report **Student Breakout**

Chosen to report the number of students of each **MAJOR** by their **CLASS** years

Constrained the query to include all majors except Statistics

The Data Entry tool updates the variables in the User Input Control window.

If you click on the **Show Controls** button in the Capsule icon's window header, the User Input Control window appears, as shown in Figure 37 on page 64.

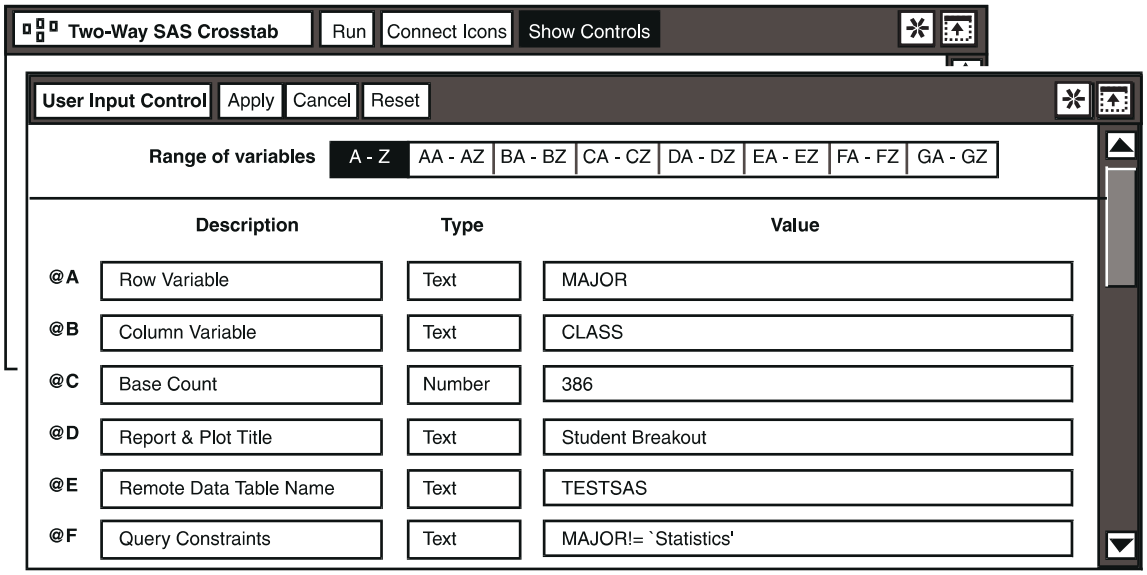


Figure 37. Sample 3: Partial Capsule control window as updated by Set Choices Here icon

In the User Input Control window, you see the choices that the user has entered in the Data Entry icon window, as well as two other pieces of data: the **Base Count** of 386 data items, which is set by the Spreadsheet icon named **Set Base Count**, and the name of the table in the remote database from which the application reads its data.

Note that this is only the top part of the User Input Control window. The @-variable fields actually continue down the window through **@Z**.

With the control window now closed again, and all icons and arrows deselected, the user clicks on the **Run** button in the Capsule window header to start the application processing.

The SQL icon named **Get Base Count** (top row of icons in the outer Capsule icon) obtains a count of the data elements being reported, and moves that number to the Spreadsheet icon **Set Base Count**.

The SQL icon named **Input Data** (second row of icons) receives raw data from the remote database. In this case, it looks like the data in Figure 38 on page 65.

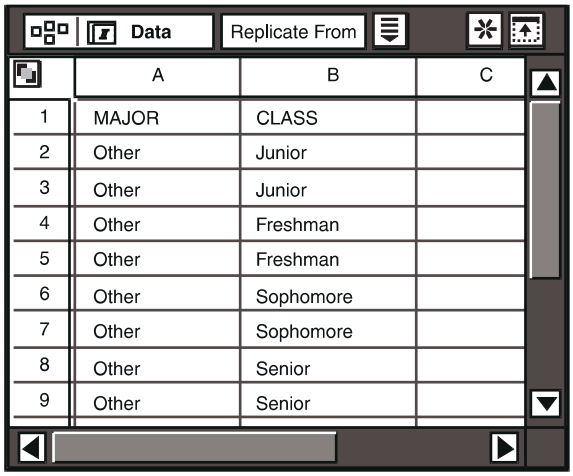


Figure 38. Sample 3: Database data formatted in spreadsheet

This data is then transferred to one of the transformer's input regions.

1. The transformer establishes a connection to the host system. It then calls the command script housed in the Text icon **REXX** (fourth row of icons), moves a copy of the script to the host, and hands over control to the script.

   The script icon is named in the Transformer Controls window as shown in Figure 39.

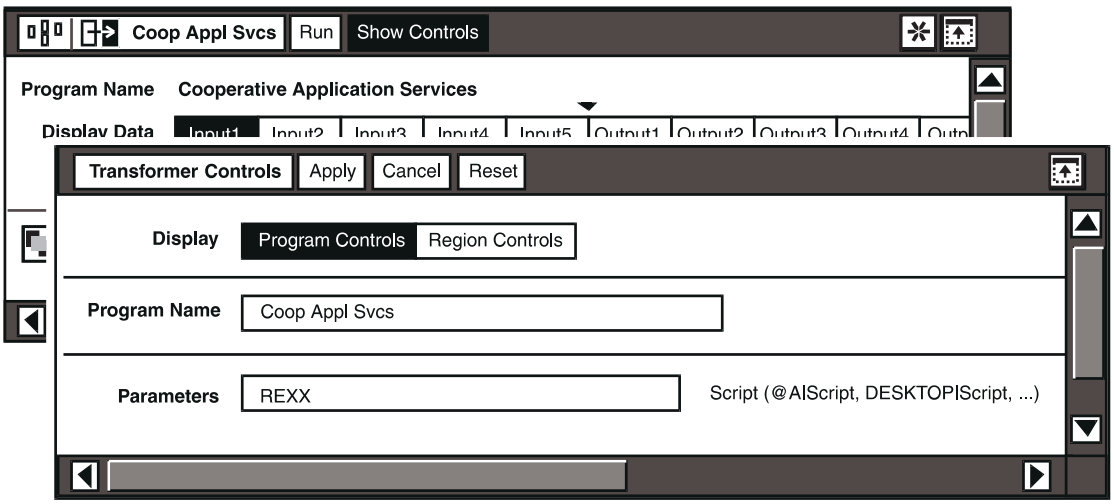


Figure 39. Sample 3: Transformer Controls window showing script name in Parameters

2. The script basically does four tasks:

   It allocates data sets.

It reads input regions into host files.

It calls the SAS routine (housed in the SAS icon, third row of icons) that has been transferred from the desktop to the host, through an input region of the transformer. On the host, the SAS code runs in conjunction with host-resident SAS software, incorporating @-variable values from the Data Entry icon. When the SAS code has finished, it creates output files and a SAS run log, which are transferred back to the desktop.

The script writes the output to the transformer's output regions.

The REXX script and the SAS program are shown in Figure 40 on page 67 and Figure 42 on page 69, respectively.

The field markers that appear in the first page of the script and in the SAS program are visible only when the code is on the IDS desktop.

```
/* REXX
  ---------------------------------------------------------------
  PROGRAM:  REXXTAB
  FUNCTION: TWO-WAY CROSSTAB ANALYSIS
  SCENARIO: DATA EXISTS ON REMOTE DATABASE SERVER. DATA IS PROCESSED
        ON HOST-BASED SAS, THEN RESULTS ARE ANALYZED FURTHER AND
        PRESENTED USING IDS.  END-USER INTERACTION IS SOLELY WITH IDS.
  EXTERNAL PROGRAM CALLS:
        METFILE - ALLOC, FREE FUNCTION
        METRGN - OPEN, CLOSE, GET, PUT W/S REGION FUNCTION
        METERROR - PRINT ERRORS AND EXIT ROUTINE
        SAS - RELEASE 6.06
  ---------------------------------------------------------------
  EXAMPLE CODE IS PROVIDED ONLY FOR ILLUSTRATIVE PURPOSES. YOU MAY
  NEED TO MODIFY THIS CODE TO USE IN YOUR ENVIRONMENT.
  ===============================================================
                 M A I N  P R O G R A M
  ===============================================================*/
CAS_RC = 0
DCB1='DSORG(PS) RECFM(F,B) LRECL(80) BLKSIZE(3120) SPACE(2,2) TRACKS DELETE';
DCB2='DSORG(PS) RECFM(F) LRECL(133) BLKSIZE(133) SPACE(2,2) TRACKS DELETE';
DCB3='DSORG(PS) RECFM(F,B) LRECL(80) BLKSIZE(3120) SPACE(2,2) TRACKS DELETE';
DCB4='DSORG(PS) RECFM(F) LRECL(133) BLKSIZE(133) SPACE(2,2) TRACKS DELETE';
DCB5='DSORG(PS) RECFM(F) LRECL(133) BLKSIZE(133) SPACE(2,2) TRACKS DELETE';
DATA_FMT = 'CHAR 1 ⌈10⌋, CHAR ⌈11⌋ 10'
TEXT_FMT = 'CHAR 1 80'
OUT_FMT = 'CH 21 ⌈12⌋, CH ⌈33⌋ ⌈10⌋, CH 44 11, CH 55 25'
HOST_FMT = /CHAR 1 133'

SAY 'REXXTAB BEGINS... ' DATE()  TIME()
SAY ' '
CALL ALLOC_DATA_SETS
CALL GET_REGIONS
CALL EXEC_SAS
CALL PUT_REGION
SAY ' '
SAY 'REXXTAB ENDS... '  DATE()  TIME()
EXIT

/*===============================================================
                    ALLOCATE DATASETS
  ===============================================================
ALLOC_DATA_SETS:
CALL METFILE 'ALLOC', 'INPUTF', 'XTAB.DATA1', DCB1
CALL METFILE 'ALLOC', 'OUTLOG', 'XTAB.DATA2', DCB2
CALL METFILE 'ALLOC', 'SAS', 'XTAB.DATA3', DCB3
CALL METFILE 'ALLOC', 'OUTF', 'XTAB.DATA4', DCB4
CALL METFILE 'ALLOC', 'OUTF2', 'XTAB.DATA5', DCB5
```

Data description values within field markers (from @-variables)

Figure 40. Sample 3: Script written in REXX (1 of 2)

```
/*============================================================
          READ INPUT REGIONS INTO HOST FILES
  ============================================================
*/
GET_REGIONS:
SAY '---> GET INPUT REGIONS'
'NEWSTACK'
CALL METRGN 'GET', '1', 'INPUTF', DATA_FMT
CALL METRGN 'GET', '3', 'SAS', TEXT_FMT
'DELSTACK'
RETURN
/*============================================================
                    EXECUTE SAS PROGRAM
  ============================================================
*/
EXEC_SAS:
'NEWSTACK'
/* ERROR-TRAPPING VARIABLES */
CAS_RTN = 'REXXTAB.EXEC_SAS'
CAS_SRV = 'FAILURE'
CAS_PMSG = 'SAS EXECUTION FAILED ON HOST'
CAS_SMSG = 'FAILED:SAS606 INPUT STATEMENT'
SAY '---> INVOKE SAS 6.06 ...'

/* SWITCH TO TSO COMMAND LINE */
"SAS606 INPUT ('XTAB.DATA3') LOG('XTAB.DATA2')"
IF RC <> 0 THEN
   CALL METERROR CAS_RTN, CAS_SRV, RC, CAS_PMSG, CAS_SMSG/
'DELSTACK'
RETURN

/*============================================================
              WRITE OUTPUT AND LOG REGIONS
  ============================================================
*/
PUT_REGION:
SAY '---> PUT OUTPUT & LOG REGIONS'
'NEWSTACK'
CALL METRGN 'PUT', '3', 'OUTF', 'CLOSE', HOST_FMT
CALL METRGN 'PUT', '4', 'OUTF2', 'CLOSE', OUT_FMT
CALL METRGN 'PUT', '5', 'OUTLOG', 'CLOSE', HOST_FMT
/* FREE DATASETS */
CALL METFILE 'FREE', 'OUTF'
CALL METFILE 'FREE', 'OUTF2'
CALL METFILE 'FREE', 'SAS'
CALL METFILE 'FREE', 'OUTLOG'
CALL METFILE 'FREE', 'INPUTF'
'DELSTACK'
RETURN
```

Figure 41. Sample 3: Script written in REXX (2 of 2)

Figure 42 on page 69 shows the SAS program called by the REXX command script.

```
/* SAS PROGRAM
 ------------------------------------------------------------------
  PROGRAM:  SASCODE

  FUNCTION: CROSS-TAB ANALYSIS OF STUDENT MAJOR BY CLASS.
             GIVES TABULAR AND LIST OUTPUT TO TWO DATA SETS.

  PROCS:    FREQ, PRINTTO

 ------------------------------------------------------------------

 EXAMPLE CODE IS PROVIDED ONLY FOR ILLUSTRATIVE PURPOSES. YOU MAY
 NEED TO MODIFY THIS CODE TO USE IN YOUR ENVIROMNMENT.

 ------------------------------------------------------------------
*/
OPTIONS LINESIZE=80;
/*============================================================
    READ INPUT FILE
  ============================================================
*/
DATA ONE;
  INFILE INPUTF;
  INPUT ⌈MAJOR⌋ $ ⌈10⌋. ⌈CLASS⌋ $ ⌈9⌋.;

/*============================================================
    RUN PROC FREQ AND SEND REPORT OUTPUT TO OUTF
  ============================================================
*/
PROC PRINTTO PRINT=OUTF NEW;
PROC FREQ;
  TABLES ⌈MAJOR⌋ * ⌈CLASS⌋;
  TITLE '⌈Student Breakout⌋';
PROC PRINTTO;

/*============================================================
  RUN PROC FREQ WITH DIFFERENT OPTIONS, SEND REPT OUTPUT TO OUTF2
  ============================================================
*/
PROC PRINTTO PRINT=OUTF2 NEW;
PROC FREQ;
  TABLES ⌈MAJOR⌋ * ⌈CLASS⌋ /LIST NOCUM;
  TITLE '⌈Student Breakout⌋';
PROC PRINTTO;
RUN;
```

Data description values within field markers (from @-variables)

Figure 42. Sample 3: SAS program

3. The application creates three types of output:

   Information on how the job ran is transferred through two of the transformer's output regions to the Text icons **REXX Run Log** and **SAS Run Log**.

   A textual version of the output also goes by way of an output region to the Text icon **SAS Crosstab (@A/@B)**, on the bottom row of icons. When you open the icon, the report looks like the one in Figure 43 on page 70.

Another version of the output data travels through the Spreadsheet icon called **SAS Output** and the Clean transformer (third row of icons) to the Spreadsheet icon called **Crosstab Output (@A/@B)**. The result is shown in Figure 44 on page 71.

From there, the spreadsheet data is sent through the Capsule icon called **Format Data** in the third row of icons. **Format Data** contains many manipulating and formatting processes. Finally, the spreadsheet data is used to create several plots, which, along with run logs and other output, are filed and labeled automatically in the Folder icon named **Output** (last icon on the right).

Two of the plots, **Percentage Breakout (MAJOR/CLASS)** and **Comparison Groups (MAJOR/CLASS)**, are shown in Figure 45 on page 72 and Figure 46 on page 73, respectively.

1 Student Breakout 18:25 Friday, June 17, 1994

1

TABLE OF MAJOR BY CLASS

MAJOR CLASS

Frequency Percent Row Pct Col Pct	Freshman	Junior	Senior	Sophomore	Total
Biology	10	6	8	9	33
	3.68	2.21	2.94	3.31	12.13
	30.30	18.18	24.24	27.27	
	12.20	11.11	14.04	11.39	
Other	66	37	39	67	209
	24.26	13.60	14.34	24.63	76.84
	31.58	17.70	18.66	32.06	
	80.49	68.52	68.42	84.81	
Psychology	6	11	10	3	30
	2.21	4.04	3.68	1.10	11.03
	20.00	36.67	33.33	10.00	
	7.32	20.37	17.54	3.80	
Total	82	54	57	79	272
	30.15	19.85	20.96	29.04	100.00

Base Count = 272

Figure 43. Sample 3: Text report contained in SAS Crosstab (@A/@B) icon

Crosstab Output (@A/@B) | Replicate From | Recalculate | Sort Controls | Copy/Move Controls | Clear

	A	B	C	D	E	F	G	H
1								
2	MAJOR	CLASS	Frequency	Percent				
3					=MAJOR	=CLASS	COUNT	PERCENT
4	Biology	Freshman	10	3.7	=Biology	=Freshman	= 10	=
5	Biology	Junior	6	2.2	=Biology	=Junior	= 6	=
6	Biology	Senior	8	2.9	=Biology	=Senior	= 8	=
7	Biology	Sophomore	9	3.3	=Biology	=Sophomore	= 9	=
8	Other	Freshman	66	24.3	=Other	=Freshman	= 66	=
9	Other	Junior	37	13.6	=Other	=Junior	= 37	=
10	Other	Senior	39	14.3	=Other	=Senior	= 39	=
11	Other	Sophomore	67	24.6	=Other	=Sophomore	= 67	=
12	Psychology	Freshman	6	2.2	=Psychology	=Freshman	= 6	=
13	Psychology	Junior	11	4.0	=Psychology	=Junior	= 11	=
14	Psychology	Senior	10	3.7	=Psychology	=Senior	= 10	=
15	Psychology	Sophomore	3	1.1	=Psychology	=Sophomore	= 3	=

Figure 44. Sample 3: Crosstab Output (@A/@B) icon showing report data for plots

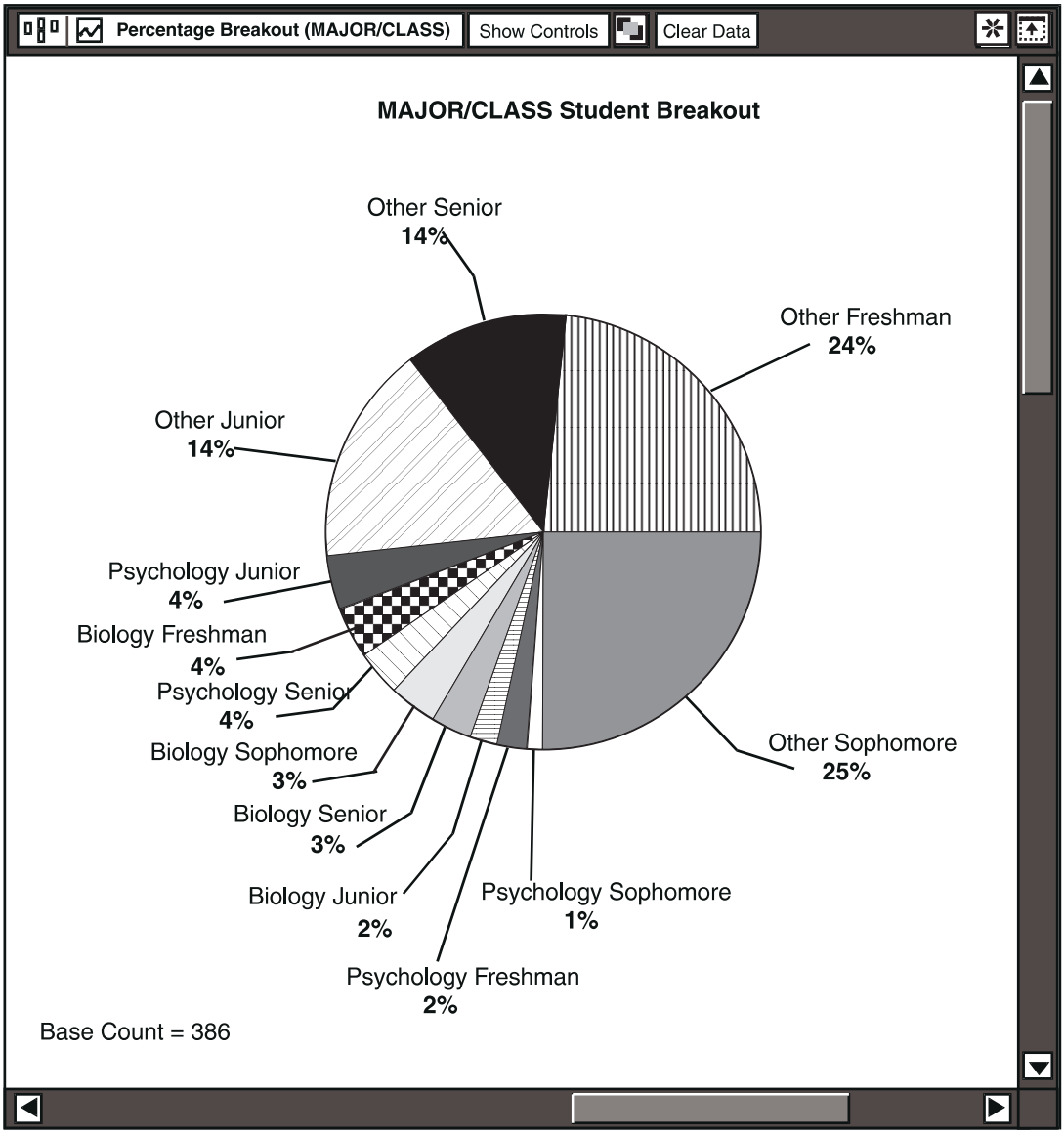


Figure 45. Sample 3: Percentage Breakout (MAJOR/CLASS) plot icon

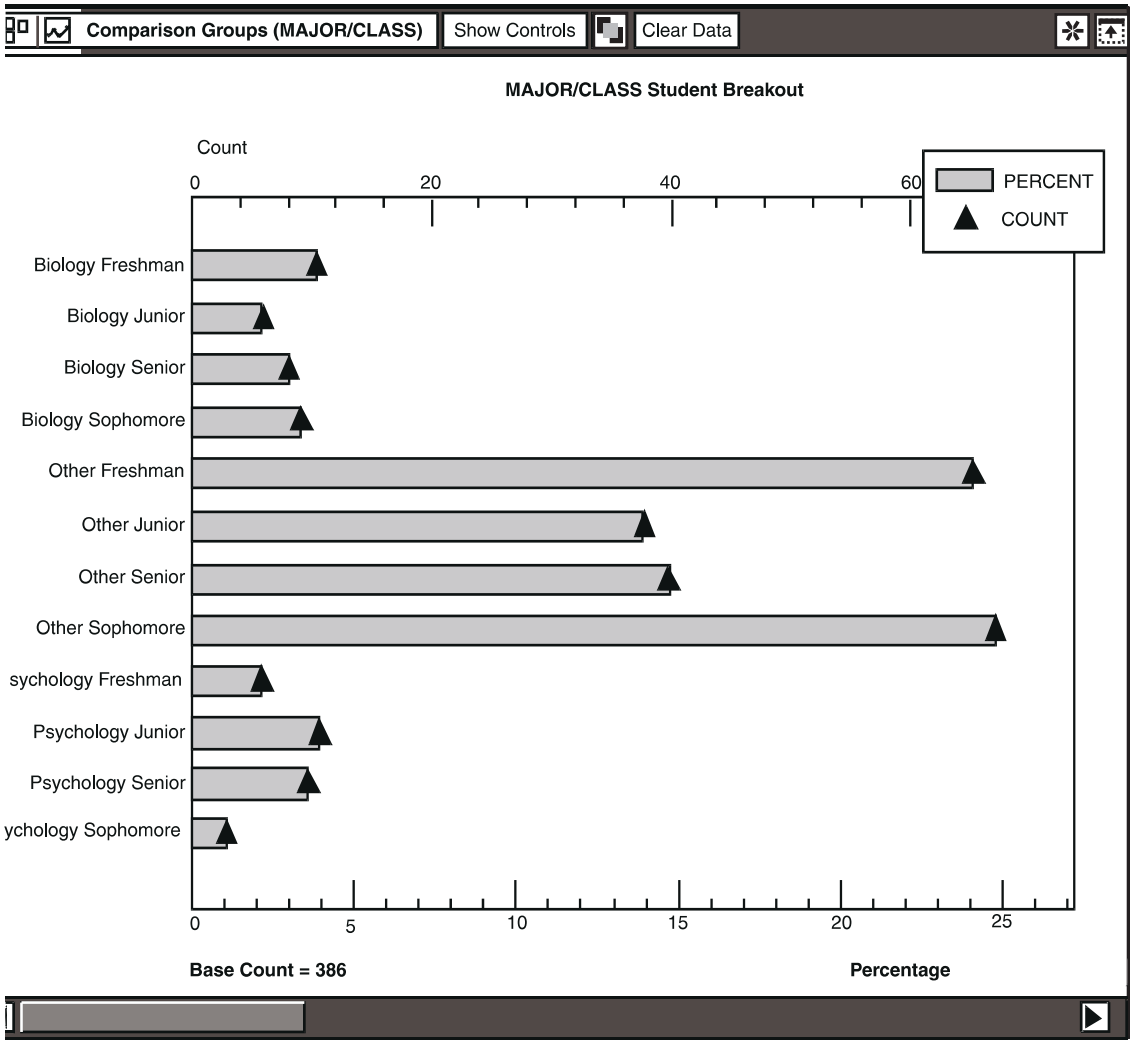


Figure 46. Sample 3: Plot icon Comparison Groups (MAJOR/CLASS)

Sample 4: A Regression Analysis Using DB2, DXT, and SAS

You have seen in the previous samples that you can use CAS to invoke any host-based application from your IDS desktop. The following sample is a more powerful and complex statistical application: a regression analysis of the type that marketing researchers frequently use in forecasting. This application invokes Data Extract (DXT) to access DATABASE 2 (DB2) and to pass data to SAS.

In Figure 47 on page 74 you see the Capsule architecture.

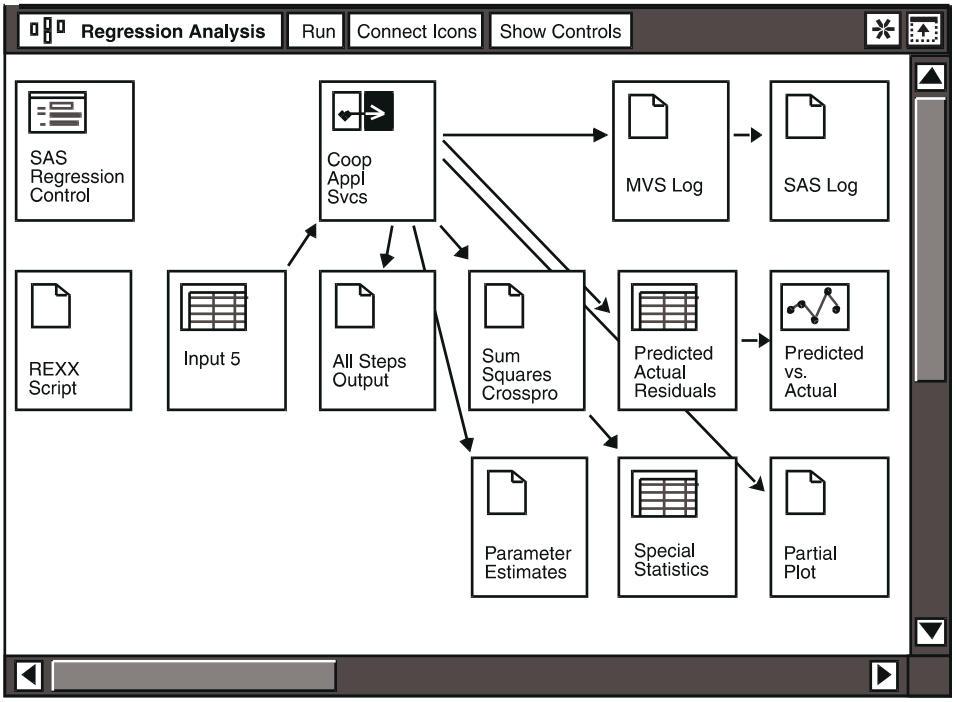


Figure 47. Sample 4: Capsule application Regression Analysis

Once again, as in previous sample applications, the user sets the parameters and controls for the desired regression analysis inside the Data Entry icon (top left), which has been named **SAS Regression Control**. The Data Entry window can be designed to offer the user several levels of choice, as you will see in the following figures.

The first level of the Data Entry window, shown in Figure 48 on page 75, is designed for the user to enter parameters such as dependent and independent variables. Note that **SAS Regression Parameter Options** (the left-most section of the first field) appears selected.

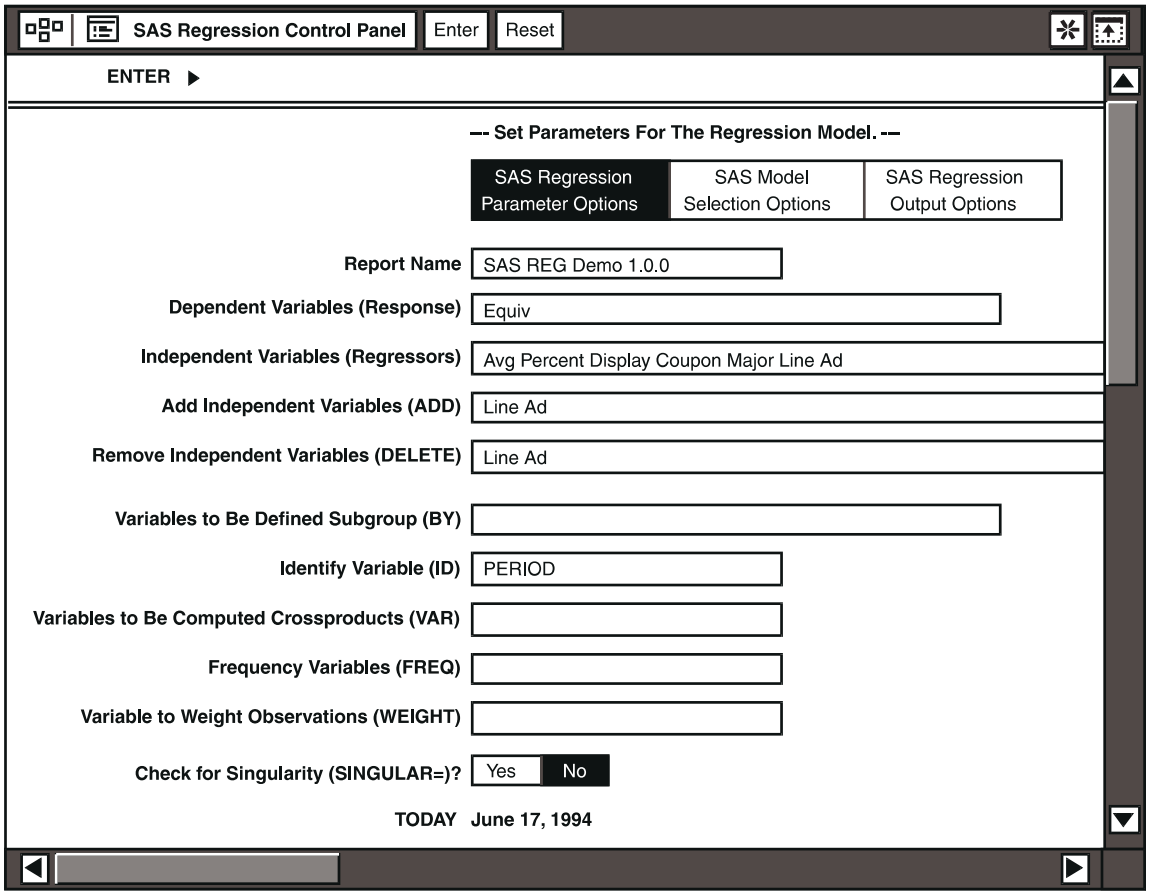


Figure 48. Sample 4: SAS Regression Control Panel (Data Entry) icon, first window: parameters

If you are building a similar application, you may wish to do as the designer of this Capsule application has done. Notice that he has placed the options that are likely to be used more frequently at the top of the window, and the more complex options in the lower part. In this level of the window, the more complex options start at **Variables to Be Defined Subgroup (BY)**.

Figure 49 on page 76 shows the second level of the Data Entry icon window, which the user reaches by the following two steps:

1. Clicking on the **Enter** button in the window header.
2. Selecting the second section, **SAS Model Selection Options**, of the topmost field.

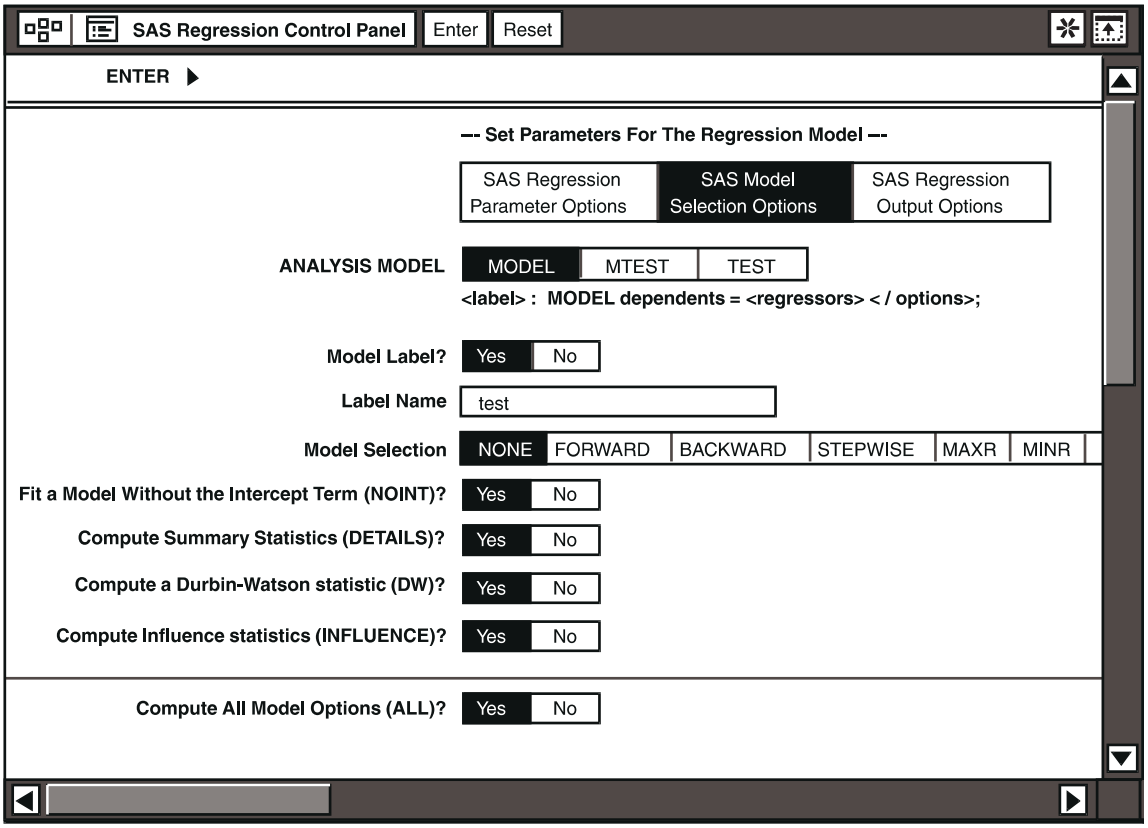


Figure 49. Sample 4: SAS Regression Control Panel icon, second window: regression models

In this second level of the SAS Model Selection Options window, the user first goes to the **ANALYSIS MODEL** field and chooses whether to use one of eight preconfigured regression models (**MODEL**), a model with special constraints (**MTEST**), or a single-assumption test (**TEST**).

If the user selects **MODEL**, the fields that you see in Figure 49 appear under the **ANALYSIS MODEL** field. The user must then choose from the **Model Selection** field (in the figure, not all the model names are visible). For each model selected, a different set of fields appears in the lower part of the window.

The designer of this sample application has also provided a **Compute All Model Options?** field.

Finally, to finish specifying how the regression analysis is to run, the user goes to the third level of the Data Entry icon window by:

1. Clicking on the **Enter** button in the window header.
2. Selecting the third section, **SAS Regression Output Options**, of the topmost field.

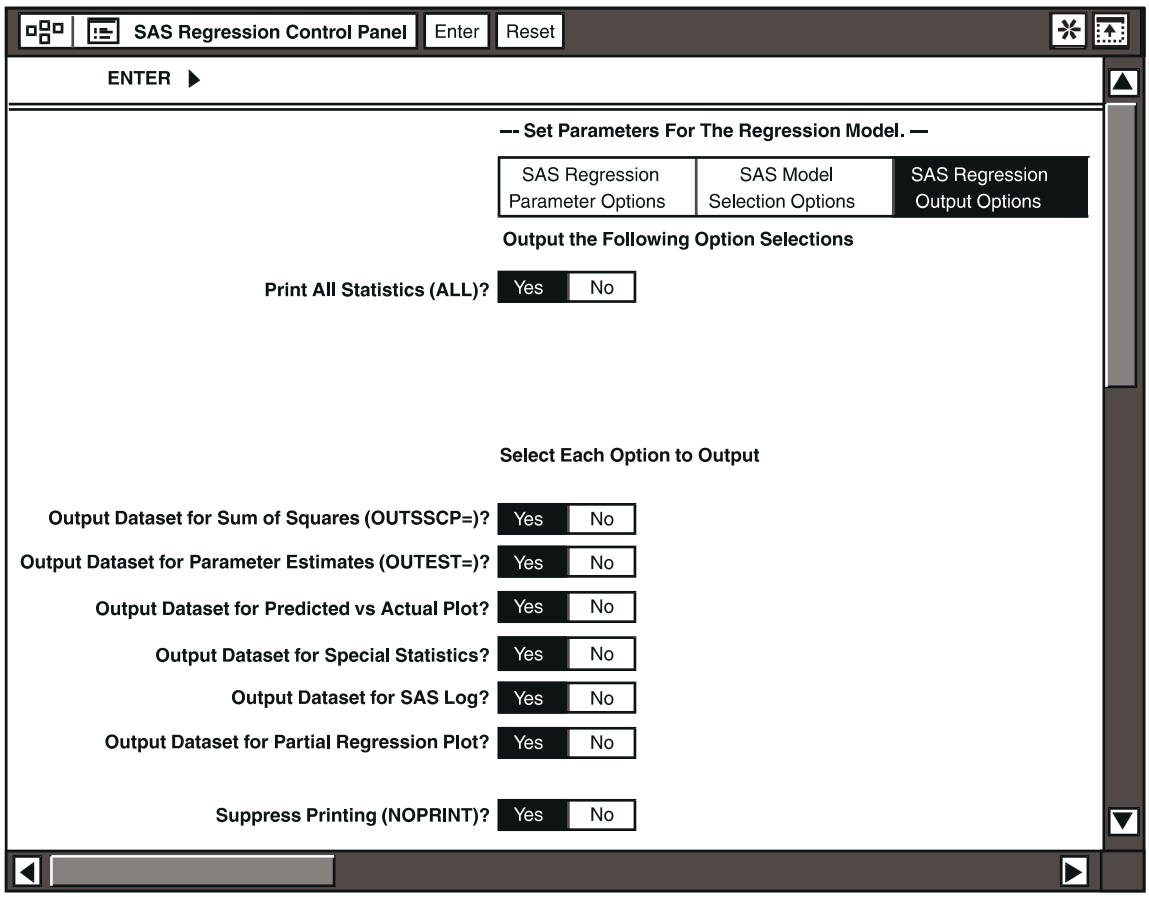


Figure 50. Sample 4: SAS Regression Control Panel icon, third window: output options

In the third level of the Data Entry window, shown in Figure 50, the user decides which of many available types of output to generate during this regression analysis. If you select the first choice, **Print All Statistics (ALL)?**, the other choices become unnecessary. If, on the other hand, only certain outputs are required, the user may select from the fields in the lower half of the window. Finally, there is a printing suppression option.

Every choice under **Select Each Option to Output** matches an icon inside the Capsule icon. For example, if the user selects **Yes** beside **Output Dataset for Sum of Squares (OUTSSCP=)?**, at the end of the run there will be data in the Text icon named **Sum Squares Crossproduct**.

Two of the icons not represented in this selection list, **MVS Log** and **All Steps Output**, receive data automatically every run. The Spreadsheet icon **Predicted Actual Residuals** receives data if the user selects **Yes** beside **Output Dataset for Predicted vs Actual Plot?**

After selecting all the options needed for the run, the user clicks on the **Enter** button, closes the Data Entry window, and clicks on the **Run** button in the Capsule window header. Processing begins.

As discussed earlier in this appendix, the Data Entry icon updates the Capsule control window with all the data the user selected. The Capsule controls pass on the data to the CAS transformer, which calls the Text icon named **REXX Script**.

To call the script, the Transformer Controls window has been set up as in previous samples, with the script's icon name, **REXX Script**, in the **Parameters** field. See the foregoing sections of this appendix if you would like details on linking the script to the transformer.

The transformer passes a copy of the script to the host and hands over control to the script, which in turn calls a SAS module that resides on the host. The SAS module calls other host-resident SAS software that is designed for regression analysis.

You can see the various calls in the following figures. The REXX script and the host-resident SAS module appear in Figure 51 on page 79 and Figure 65 on page 93 respectively.

```
/* REXX
  ---------------------------------------------------
  PROGRAM: REXXREG

  FUNCTION:CAS REGRESSION ANALYSIS

  EXTERNAL PROGRAM CALLS:
           METFILE - ALLOC, FREE FUNCTION
           METRGN - OPEN, CLOSE, GET, PUT REGIONS
           METERROR - PRINT ERRORS AND EXIT ROUTINE
           SAS - RELEASE 6.06
           DXT
  ============================================================

  EXAMPLE CODE IS PROVIDED ONLY FOR ILLUSTRATRATION. YOU MAY
  NEED TO MODIFY THIS CODE TO USE IN YOUR ENVIRONT<MENT.

  ============================================================
                       MAIN PROGRAM
  ============================================================

  INITIAL I/O DATA DESCRIPTION*/
RC = 0
SAY " ***Begin: Specify the data descriptions for I/O..."
Out7Desc = 'char 1 132'
OUT1Desc = 'char 1 132'
OUT2Desc = 'char 1 132'
OUT3Desc = 'char 1 132'
OUT6Desc = 'char 1 132'
OUT4Desc = 'ch 1 18 Period,fl 21 15 Equiv,fl 37 15 Predicted,,
fl 52 15 Resid,fl 68 15 ZValue,fl 84 15 UpperLim,fl 100 15 LowerrLim'
OUT5Desc = 'ch 1 18 Period,fl 20 15 Cookd,fl 36 15 Beta_va,,
fl 52 15 std_inf,fl 68 15 leverage,fl 84 15 Resid_di'
In5Desc = 'ch 1 18,fl 20 15,fl 36 15,fl 52 15,fl 68 15,fl 84 15,,
fl 100 15,fl 116 15'

DCB3='DSORG(PS) RECFM(V) LRECL(265) BLKSIZE(1012) SPACE(2,2) TRACKS DELETE';
DCB2='DSORG(PS) RECFM(F) LRECL(133) SPACE(2,2) TRACKS DELETE';
DCB1='DSORG(PS) RECFM(V) LRECL(265) BLKSIZE(5010) SPACE(2,2) TRACKS DELETE';

SAY " ***End: Specify the data descriptions for I/O..."

/* MAIN PROGRAM MAKES THE SUBROUTINES CALL */
SAY '***** REXX REGRESSION BEGINS... ' DATE() TIME()
SAY ' '
Call Allocate;
Call Get_Regions;
```

Figure 51. Sample 4: REXX script (1 of 14)

```
If ( '@FB' = 'input5' ) then
  Call Get_Input5
else
  Call Exe_dxt;
Call Exe_sas;
Call Put_Regions;

SAY ' '
SAY '***** REXX REGRESSION ENDS... ' DATE() TIME()

Exit;
/*================================================
           ALLOCATE THE OUTPUT5 AND LOG REGIONS
  ================================================
*/
Allocate:
rc = metfile('alloc', 'out1', 'reg.out1', DCB2);
rc = metfile('alloc', 'out2', 'reg.out2', DCB2);
rc = metfile('alloc', 'out3', 'reg.out3', DCB2);
rc = metfile('alloc', 'out4', 'reg.out4', DCB3);
rc = metfile('alloc', 'out5', 'reg.out5', DCB3);
rc = metfile('alloc', 'out6', 'reg.out6', DCB1);
rc = metfile('alloc', 'out7', 'reg.out7', DCB2);

If ( '@FB' \= 'input5' ) then
  rc = Metfile('alloc', 'In5', 'sas.dxtout', 'shr');
else
  rc = Metfile('alloc', 'In5', 'inData', DCB3);

RETURN
/*================================================
                EXECUTE DXT PROGRAM
  ================================================
*/
Exe_dxt:
/* EXTRACT DATA FROM DB2 TABLE TO A SEQUENTIAL DATA SET    */
/* NAMED DXTOUT.  THIS DATA SET IS AN INPUT SAS DATA SET */
'NEWSTACK'

SAY '  ***** BEGIN: EXEC DXT ...'
CAS_RTN = 'REXXREG.EXEC_DXT'
CAS_SRV = 'FAILURE'
CAS_PMSG = 'DXT EXECUTION FAILED ON HOST'
CAS_SMSG = 'FAILED:DXT INPUT STATEMENT'

/* SWITCH TO TSO COMMAND LINE */
'EXEC ''dis.sas.rexx0'''
IF RC <> 0 THEN
CALL METERROR CAS_RTN, CAS_SRV, RC, CAS_PMSG, CAS_SMSG
SAY '  ***** END: EXEC DXT ...'
'DELSTACK'
RETURN
```

Figure 52. Sample 4: REXX script (2 of 14)

```
/*================================================
                READ REGIONS: INPUT5
  ================================================
*/
Get_Input5:
say "  ***Begin: Get Input 5 From workstation..."
'NewStack'

rc = MetFGet(5, In5, In5Desc);
if( rc > 0 ) then rc = METMDSP("Error! Problem Get Input 5 = " rc)

'DelStack'
say "  ***End: Get Input 5 From workstation..."
RETURN

/*================================================
                EXECUTE SAS6.06 PROGRAM
  ================================================
*/
Exe_sas:
'NEWSTACK'
SAY '  ***** BEGIN: INVOKE SAS606 ...'
/* ERROR-TRAPPING VARIABLES */
CAS_RTN = 'REXXREG.EXEC_SAS'
CAS_SRV = 'FAILURE'
CAS_PMSG = 'SAS EXECUTION FAILED ON HOST'
CAS_SMSG = 'FAILED:SAS606 INPUT STATEMENT'
/* SWITCH TO TSO COMMAND LINE */
"SAS606 INPUT('sas.reg0') OPTIONS('nocenter minstg ps=500')"
IF RC <> 0 THEN
CALL METERROR CAS_RTN, CAS_SRV, RC, CAS_PMSG, CAS_SMSG
SAY '  ***** END: INVOKE SAS606 ...'
'DELSTACK'
RETURN

/*=======================================================
    ALLOCATE, WRITE AND FREE THE OUTPUT AND LOG REGIONS
  =======================================================
*/
Put_Regions:
say " ***Begin: Allocate, Transfer data, and Free files..."
'NewStack'

say'    ===>Extract ALL STEPS OUTPUT!';
if( rc > 0 ) then rc = METMDSP("Error! Problem ut1 Alloc= " rc)
rc = MetRgn( 'Put', '1', 'out1', 'Close', Out1Desc, 'Char' );
if( rc > 0 ) then do
  rc = METMDSP("Error! Problem Put Output1 = " rc)
  'Free File(out1)'
  Return;
end
'Free File(out1)'
```

Figure 53. Sample 4: REXX script (3 of 14)

```
say'    ===>Extract OUTSSCP!';
if( rc > 0 ) then rc = METMDSP("Error! Problem Out2 Alloc= " rc)
rc = MetRgn( 'Put', '2', 'out2', 'Close', Out2Desc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put Output2 = " rc)
    'Free File(out2)'
    Return;
end
'Free File(out2)'

say'    ===>Extract OUTEST!';
if( rc > 0 ) then rc = METMDSP("Error! Problem Out3 Alloc= " rc)
rc = MetRgn( 'Put', '3', 'out3', 'Close', Out3Desc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put Output3 = " rc)
    'Free File(out3)'
    Return;
end
'Free File(out3)'

say'    ===>Extract ACTUAL, PREDICTED, RESIDUALS!';
if( rc > 0 ) then rc = METMDSP("Error! Problem Out4 Alloc= " rc)
if( '@AJ' \= '' ) then
    rc = MetRgn( 'Put', '4','out4', 'Close', Out4Desc, 'Char' );
else
    rc = MetRgn( 'Put', '4','out4', 'Close', LOGDesc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put Output4 = " rc)
    'Free File(out4)'
    Return;
end
'Free File(out4)'

say'    ===>Extract SPECIAL STATISTICS!';
if( rc > 0 ) then rc = METMDSP("Error! Problem Out5 Alloc= " rc)
if( '@AK' \= '' ) then
    rc = MetRgn( 'Put', '5','out5', 'Close', Out5Desc, 'Char' );
else
    rc = MetRgn( 'Put', '5','out5', 'Close', LOGDesc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put Output5 = " rc)
    'Free File(out5)'
    Return;
end
'Free File(out5)'

say'    ===>Extract PARTIAL REGRESSION!';
if( rc > 0 ) then rc = METMDSP("Error! Problem Out6 Alloc= " rc)
rc = MetRgn( 'Put', '6', 'out6', 'Close', Out6Desc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put Output6 = " rc)
    'Free File(out6)'
    Return;
```

Figure 54. Sample 4: REXX script (4 of 14)

```
end
'Free File(out6)'

say'    ===>Extract SASLOG!';
if( rc > 0 ) then rc = METMDSP("Error! Problem LOG Alloc= " rc)
rc = MetRgn( 'Put', '7', 'out7', 'Close', Out7Desc, 'Char' );
if( rc > 0 ) then do
    rc = METMDSP("Error! Problem Put LOG = " rc)
    'Free File(log)'
    Return;
end

'DelStack'

say " ***End: Allocate, Transfer data, and Free files..."
RETURN

/*=========================================================
                READ REGIONS INTO HOST FILES
  =========================================================
Get_Regions:
say " ***Begin: Get Input Regions From workstation..."
'NewStack'

If ( '@FB' \= 'input5' ) then do
  rc = Metfile('alloc', 'In1', 'sas.dxt0', 'shr');
  "Execio 1 DISKRU In1 23 (LIFO"
  line = '  @FA;'
  PUSH line
  "Execio 1 DISKW In1 (FINIS"
  'Free File(In1)'
end

rc = Metfile('alloc', 'In2','sas.reg0', 'shr');
"Execio * DISKR In2 (STEM list."
"Execio * DISKR In2 (FINIS"
"FREE F(In2)"
if( '@AL' = 'SASLOG' ) then
  list.5 = 'PROC PRINTTO LOG=out7;';
else
  list.5 = '/* NO SAS OUTPUT */';

If ( '@FB' \= 'input5' ) then
  list.48 = '          input PERIOD $ 1-11 Equiv'
else
  list.48 = '          input PERIOD $ 1-18 Equiv'
if( '@AA' = 'ALL' ) then do
  if( '@S' \= ' ' ) then
    list.69 = 'PROC REG @AB @AA SINGULAR=@T'
  else
    list.69 = 'PROC REG @AB @AA'
End
```

Figure 55. Sample 4: REXX script (5 of 14)

```
Else do
  if( '@S' \= ' ' ) then
    list.69 = 'PROC REG @AB @AC @AD @AE SINGULAR=@T'
  else
    list.69 = 'PROC REG @AB @AC @AD @AE'
End

if( '@AI' \= ' ' & '@AG' \= ' ' ) then
  list.70 = '     outest=out2 outsscp=out3;'
if( '@AI' \= ' ' & '@AG' = ' ' ) then
  list.70 = '     outest=out2;'
if( '@AI' = ' ' & '@AG' \= ' ' ) then
  list.70 = '     outsscp=out3;'
if( '@AI' = ' ' & '@AG' = ' ' ) then
  list.70 = '     ;'

if ( '@EA' \= ' ' ) then
  Call Model;
if ( '@EB' \= ' ' ) then
  Call Mtest;
else do
  list.84 = '  /* MTEST command */'
  list.85 = '  /* options       */'
end

if ( '@EC' \= ' ' ) then
  Call Test;
else do
  list.86 = '  /* TEST command  */'
  list.87 = '  /* options       */'
end

if( '@G' \= ' ' ) then
  list.78 = '  ID @G;'
else
  list.78 = '  /* ID variables */'
if( '@F' \= ' ' ) then
  list.79 = '  BY @F;'
else
  list.79 = '  /* BY variables */'
if( '@H' \= ' ' ) then
  list.80 = '  VAR @H;'
else
  list.80 = '  /* VAR variables */'

if( '@I' \= ' ' ) then
  list.81 = '  FREQ @I;'
else
  list.81 = '  /* FREQ variables */'
if( '@J' \= ' ' ) then
  list.82 = '  WEIGHT @J;'
else
  list.82 = '  /* WEIGHT variables */'
```

Figure 56. Sample 4: REXX script (6 of 14)

```
if( '@AJ' \= ' ' ) then do
  list.89 = '  OUTPUT OUT=out4 p=predict r=Resdis  stdr=StdError'
  list.90 = '         student=Z_Value U95=UpperLim L95=LowerLim;'
end
else do
  list.89 = '/* OUTPUT 4 command here    */'
  list.90 = '/*        4                 */'
end

if( '@AK' \= ' ' ) then do
  list.91 = '  OUTPUT OUT=out5 CookD=CookD covratio=Beta_VA''
  list.92 = '         DFFITS=Std_Inf H=leverage Press=Resid_DI;'
end
else do
  list.91 = '/* OUTPUT 5 command here    */'
  list.92 = '/*        5                 */'
end

if( '@D' \= ' ' ) then do
  list.94 = 'ADD @D;'
  list.95 = 'RUN;'
end
else do
  list.94 = '/* ADD command here    */'
  list.95 = '/*        RUN          */'
end

if( '@E' \= ' ' ) then do
  list.96 = 'DELETE @E;'
  list.97 = 'RUN;'
end
else do
  list.96 = '/* DELETE command here */'
  list.97 = '/*        RUN          */'
end

if( '@AH' \= ' ' ) then do
  list.103 = 'PROC PRINTTO print=out6;';
  list.104 = 'PROC REG;';
  list.105 = '  PARTIAL: MODEL @B = @C';
  list.106 = '  / Partial;';
  list.107 = 'RUN;';
  list.108 = ' ';
  list.109 = 'DATA infor;';
  list.110 = '  input names $ 1-50;';
  list.111 = '  cards;';
  list.112 = '>>> WARNING! This output option is not selected!';
  list.113 = ' ';
  list.114 = ';';
  list.115 = 'RUN;';
  list.116 = ' ';
 End
```

Figure 57. Sample 4: REXX script (7 of 14)

```
else do
  list.103 = 'DATA infor;';
  list.104 = '  input names $ 1-50;';
  list.105 = '  cards;';
  list.106 = '>>> WARNING! This output option is not selected!';
  list.107 = ' ';
  list.108 = ';';
  list.109 = 'RUN;';
  list.110 = ' ';
  list.111 = ' ';
  list.112 = 'DATA _NULL_;';
  list.113 = '  SET infor;';
  list.114 = '  FILE out6;';
  list.115 = '  PUT names $ 1-50;';
  list.116 = 'RUN;';
End

if( '@AI' \= ' ' ) then do
  list.122 = 'PROC PRINTTO print=out2;';
  list.123 = 'PROC PRINT data=out2;';
  list.124 = ' ';
  list.125 = ' ';
  list.126 = ' '
End
else do
  list.122 = 'DATA _NULL_;';
  list.123 = '  SET infor;';
  list.124 = '  FILE out2;';
  list.125 = '  PUT names $ 1-50;';
  list.126 = 'RUN;';
End

if( '@AG' \= ' ' ) then do
  list.132 = 'PROC PRINTTO print=out3;';
  list.133 = 'PROC PRINT data=out3;';
  list.134 = ' ';
  list.135 = ' ';
  list.136 = ' '
End
else do
  list.132 = 'DATA _NULL_;';
  list.133 = '  SET infor;';
  list.134 = '  FILE out3;';
  list.135 = '  PUT names $ 1-50;';
  list.136 = 'RUN;';
End

if( '@AJ' \= ' ' ) then do
  list.142 = 'DATA _NULL_;';
  list.143 = '  SET OUT4;';
  list.144 = '  FILE out4;';
  list.145 = '  PUT PERIOD $ 1-18 Equiv 20-35 Predict 36-51 Resids 52-67 .3';
```

Figure 58. Sample 4: REXX script (8 of 14)

```
list.146 = '      Z_Value 68-83 .5 UpperLim 84-99 .2 LowerLim 1
```

```
list.156 = '  PUT PERIOD $ 1-18 Cookd 20-35 .5 Bete_VA 36-51 .5';
```