

TIBCO® Object Service Broker

Shareable Tools

*Software Release 6.0
July 2012*

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, The Power of Now, TIBCO Object Service Broker, and and TIBCO Service Gateway are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

The TIBCO Object Service Broker technologies described herein are protected under the following patent numbers:

Australia:	-	-	671137	671138	673682	646408
Canada:	2284250	-	-	2284245	2284248	2066724
Europe:	-	-	0588446	0588445	0588447	0489861
Japan:	-	-	-	-	-	2-513420
USA:	5584026	5586329	5586330	5594899	5596752	5682535

Copyright © 1999-2012 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information

Contents

Related Documentation	xv
TIBCO Object Service Broker Documentation	xv
Typographical Conventions	xx
Connecting with TIBCO Resources	xxiii
How to Join TIBCOCommunity	xxiii
How to Access All TIBCO Documentation	xxiii
How to Contact TIBCO Support	xxiii
Chapter 1 Introduction to the Shareable Tools	1
Overview	2
Main Types of Tools	2
Categories of Tools	2
Functional List of Shareable Tools	4
Batch Jobs (z/OS)	4
CICS Channels and Containers	4
Data Object Broker Information and Operations	5
Dates and Times	5
Debugging	6
Definitions of Objects	7
External Databases and Servers	7
External Memory and Routines	8
Installation of Components	8
Load from/Unload to External Files	9
Mathematical Calculation	9
Menus	10
Messages and Message Logs	10
Message Oriented Middleware	10
Printing and Output	11
Promotions	12
Read from/Write to External Files	13
Reports	13
Rules and Rules Libraries	14
Screens	15
Searches for Objects	16
Secondary Indexes	17
Security	17
Selection Lists	18

Session Options and Parameters	18
Strings and Text	18
Table Definitions and Data	21
Trigger or Validation Rules	23
Chapter 2 Using User Exits in Workbench Tools.	25
Overview	26
Purpose of the User Exits	26
Tools Supporting the User Exits	26
Description of the User Exits	27
@ENTRY_VALIDATE(<i>caller, type, name, library, new</i>)	27
@PRE_SAVE_OBJECT(<i>caller, type, name, library, new</i>)	27
@SAVED_OBJECT(<i>caller, type, name, library, new</i>)	27
Arguments	28
How to Use the Exits	29
Chapter 3 Tools.	33
ABS	34
\$ADD_DATE	35
ADMIN_RIGHTS	37
ALLOCDSN	38
@ARCH_ACCESSLOGI	39
AUDITLOG	40
BATCH	41
BATCH_ENABLE	55
BATCHLOAD_CARDS	56
\$BATCHOPT	69
BATCHUNLD_CARDS	71
\$BEEP	76
\$BLANKPAGE	77
\$BRCONTAINER	79
BROWSER	81
\$CALLRULE	84
CHANGE_SERVERID	86
CHANGERULE	88
CLEARTABLE_APPL	91
@CLOSEDSN	94
\$CLRTAB	95

@CONFIGURESERVER	98
CONFIRMACTION	100
COPY_DATA	102
COPY_DEFN	104
COPYDEFN	108
COPYLIB	116
COPYTABLE_APPL	117
COUNTOCCURRENCES	119
\$CREATE_DATE	120
CREATEUSERS	123
CROSSREFSEARCH	126
CURSOR_FLD COL	129
CURSORFIELD	130
CURSOROCC#	132
CURSOROCC_VALUE	134
CURSORTABLE	137
DASTATS	139
DATA COM	140
\$DATE_DEFAULT	141
\$DATE_LENGTH	143
\$DATE_PIC	146
\$DATE_REF	149
DBMAINTLVL	152
DEBUG	154
DEFINE_LIBRARY	155
DEFINE_MENU	156
DEFINE_OBJECTSET	157
DEFINE_OBJLIST	168
DEFINE_REPORT	173
DEFINE_TABLE	174
\$DELCONTAINER	175
DELETE_DATA	176
DELETE_DEFN	178
DELETESCREEN DATA	180
DIFF_DATA	182
DIFF_DEFN	185

DIFFDEFN	188
DISPLAY_MENU.....	194
DISPLAY_USERS.....	195
DRAW	197
EDITRULE	198
ENDMSG	199
ENTERKEY.....	200
ESTIMATETBLDFN.....	202
EVENTFIELD	205
EVENTSCREEN.....	206
EVENTSUBVIEW	209
EVENTTABLE	210
\$EXCEPTION	212
\$EXCEPTIONOBJECT.....	213
EXIT_DISPLAY	214
EXPOCC_SIZE.....	215
FCNKEY_MSG	216
FLDMGR.....	219
\$FLUSHPRINT	223
FORALLA	224
@FORALLA	231
FORALLB	238
FORALLE	241
FROM_UNICODE.....	243
\$FUNCTION	244
GEN_TED.....	246
GENBIN	250
GENERATE_REPORT	252
GENFLOAT.....	253
GENPACK.....	254
\$GETCONTAINER	256
\$GET_DECIMALS	258
\$GET_MAXSIZE.....	259
\$GET_SIZE.....	260
\$GET_SYNTAX.....	261
\$GET_TYPE.....	262

\$GETATTRIBUTE	263
\$GETBINARY	265
GETCHAR	267
\$GETCOLOUR	269
GETENDMSG	271
\$GETENVCOMMAREA	273
\$GETFLOAT	274
\$GETOPT	276
\$GETPACKED	278
\$GETTRANSACTION	280
\$GTFSET	281
HEADSTRING	284
HLIPREPROCESSOR	286
HOUR	288
HURON_STATS	289
\$HTTPREQUEST	291
IDMS	293
IMS	294
INDEXCHK	295
@INSTALL	297
INSTALLIB	299
KEYWORDMGR	300
KEYWORDSEARCH	303
LEAPYEAR	305
LENGTH	307
LIBID	308
\$LISTDSN	309
\$LISTPDS	314
LIT_TO_VAL	318
LLOAD	319
LOAD	322
LOADER	325
LOCALTIME	330
LOG_BROWSE	332
LOWER_EBCDIC	335
LOWER_UNICODE	337

LOWERCASE	339
@MAKEMEMBERS	341
MANAGE_APPLY	343
MANAGE_REQUESTS	344
MANAGE_RIGHTS	345
@MAP	346
MATCH	352
MAX	354
MESSAGE	355
MESSAGE_LOG	357
@MESSAGEDUMP	361
@MESSAGETRACE	363
MIN	368
MINUTE	369
@MNG_USERS	370
MOD	371
@MOMCLOSE	373
@MOMCOMMIT	374
@MOMCONNECT	375
@MOMDISCONN	377
@MOMGET	378
@MOMINIT	379
@MOMMAPLENGTH	381
@MOMOPEN	382
@MOMOPTION	383
@MOMPUT	384
@MOMROLLBACK	386
@MOMSETOPT	387
@MOMSPECIALCMD	388
@MOMVALIDRC	389
\$MOVECONTAINER	390
MOV TAB	391
@MQSMAP and @MQSMAP_PORT	394
MSGLOG	396
\$NEWPAGE	398
NLS	400

NOOP	403
NUM_CHK	404
OBJECT_MGMT	405
OBJECTMGR	408
@OPENDSN	411
OPSTATS	413
OPTIONLISTER	414
\$OTMA	419
@OTMA_MAP	421
PAD	428
PARMVALUE	430
PARSE	432
PARSE_TAM	443
PATTERN_MATCH	446
PEEL	447
PEEL_HEAD	449
PEEL_TAIL	451
@PEERSERVERID	453
\$PIC	455
@PRESENTATIONENV	462
PRINT_DATA	463
PRINT_DEFN	465
\$PRINTFIELD	467
\$PRINTLINE	469
PRINTTABLE	471
PROCESS_FCNKEY	473
PROCESS_TABLE	474
@PROMBINDOBJs	477
PROM_MAIN	478
@PROMUNBINDOBJs	479
PRT_VSCR	481
PURGELOG_BATCH	483
PURGELOG_SCREEN	485
\$PUTCONTAINER	487
\$PUTLINE	489
QUOTE	490

RANDOM	491
RANDOMSEED	493
@READDSN.....	495
REALTIME	497
\$REALTIMER	498
REFMAKER	499
REMAINDER.....	501
REMOTELOCATION.....	503
\$RESETPRINT	504
RESETXPARM	506
RETURN_CODE.....	508
RETURN_MESSAGE	509
RETURN_SYMSG	510
RMANAGE_REQUESTS	511
ROUND.....	512
\$RPTIMMEDIATE	513
\$RPTOCCLIMIT	515
\$RPTOVERLAP	517
\$RPTPARMS.....	519
\$RPTPRINT	521
\$RPTSKIPLINES	523
\$RULE_EXISTS	525
\$RULENAME	526
RULEPRINTER.....	531
S6BCALL	537
S6BFUNCTION.....	539
S6BNOTIFY	541
S6BTROFF	543
S6BTRON.....	545
SCREENCOL	548
SCREENMSG	549
SCREENROW	551
SCRIPT.....	552
SEARCH.....	561
SEARCH_REPLACE	567
SEARCHLIB	568

SEC_REBIND	571
SECOND	572
SECURITY	573
SELECT_OBJ	574
@SERVERERROR	577
@SESSION	583
@SESSIONCOUNTS	587
SESSMGR	590
\$SETATTRIBUTE	597
\$SETCHANNEL	599
\$SETCOLOUR	600
SETCURSOR	602
SETCURSOR_POS	604
\$SETENVCOMMAREA	606
SETNLSBIT	608
\$SETOPT	609
\$SETP#POS	614
\$SETPRINT	617
SETREMOTELOC	620
\$SETRPTATTRIBUTE	621
\$SETRPTMEDIUM	623
\$SETSESSIONEND	625
\$SETTITLE	627
\$SETTRANSACTION	629
SETXPARM	630
\$SHOWCHANNEL	632
\$SIGNAL	633
SIXBUILD	634
SIMPLESELECT	637
SIXBUILD_CARDS	639
SIXDELETE	644
\$SKIPLINE	647
\$SLEEP	649
SOE	650
@STATICSQL	652
STE	653

STEBROWSE	657
SUBSTRING	658
\$SYSTEMDATE	660
SYSTEMLIB	661
TABLEPRINT	662
TAILSTRING	666
TED	667
TEXTSETUP	672
TIME	679
\$TOCPRI NT	680
\$TOCPUT	682
TOKEN	685
TO_UNICODE	691
@TRACEMESSAGES	692
\$TRXDATE	695
\$TRXMODE	696
\$TYPECAST	697
@UNINSTALL	699
UNLOAD	701
UNLOAD_DATA	712
UNLOAD_DEFN	714
UNLOADLIBRARY	716
\$UNPIC	718
UNQUOTE	720
UPPER_EBCDIC	721
UPPER_UNICODE	723
UPPERCASE	725
USERID	727
UTCDATE	728
UTCTIME	729
VAL_TO_LIT	730
VALID_NAME	732
@WRITEDSN	733
XMLPARSE	735
XMLSTART	737
XMLSTARTDSN	743

XMLSTARTSETDEST745

XMLSTARTTAB747

YEAR.....748

Index749

Preface

TIBCO® Object Service Broker provides an application development environment that allows you to create applications that integrate various systems in your enterprise.

Topics

- [Related Documentation, page xv](#)
- [Typographical Conventions, page xx](#)
- [Connecting with TIBCO Resources, page xxiii](#)

Related Documentation

This section lists documentation resources you may find useful.

TIBCO Object Service Broker Documentation

The following documents form the TIBCO Object Service Broker documentation set:

Fundamental Information

The following manuals provide fundamental information about TIBCO Object Service Broker:

- *TIBCO Object Service Broker Getting Started* Provides the basic concepts and principles of TIBCO Object Service Broker and introduces its components and capabilities. It also describes how to use the default developer's workbench and includes a basic tutorial of how to build an application using the product. A product glossary is also included in the manual.
- *TIBCO Object Service Broker Messages with Identifiers* Provides a listing of the TIBCO Object Service Broker messages that are issued with alphanumeric identifiers. The description of each message includes the source and explanation of the message and recommended action to take.
- *TIBCO Object Service Broker Messages without Identifiers* Provides a listing of the TIBCO Object Service Broker messages that are issued without a message identifier. These messages use the percent symbol (%) or the number symbol (#) to represent such variable information as a rules name or the number of occurrences in a table. The description of each message includes the source and explanation of the message and recommended action to take.
- *TIBCO Object Service Broker Quick Reference* Presents summary information for use in the TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Shareable Tools* Lists and describes the TIBCO Object Service Broker shareable tools. Shareable tools are programs supplied with TIBCO Object Service Broker that facilitate rules language programming and application development.
- *TIBCO Object Service Broker Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Application Development and Management

The following manuals provide information about application development and management:

- *TIBCO Object Service Broker Application Administration* Provides information required to administer the TIBCO Object Service Broker application development environment. It describes how to use the administrator's workbench, set up the development environment, and optimize access to the database. It also describes how to manage the Pagestore, which is the native TIBCO Object Service Broker data store.
- *TIBCO Object Service Broker Managing Data* Describes how to define, manipulate, and manage data required for a TIBCO Object Service Broker application.
- *TIBCO Object Service Broker Managing External Data* Describes the TIBCO Object Service Broker interface to external files (not data in external databases) and describes how to define TIBCO Object Service Broker tables based on these files and how to access their data.
- *TIBCO Object Service Broker National Language Support* Provides information about implementing the National Language Support in a TIBCO Object Service Broker environment.
- *TIBCO Object Service Broker Object Integration Gateway* Provides information about installing and using the Object Integration Gateway which is the interface for TIBCO Object Service Broker to XML, J2EE, .NET and COM.
- *TIBCO Object Service Broker for Open Systems External Environments* Provides information on interfacing TIBCO Object Service Broker with the Windows and Solaris environments. It includes how to use SDK (C/C++) and SDK (Java) to access TIBCO Object Service Broker data, how to interface to TIBCO Enterprise Messaging Service (EMS), how to use the TIBCO Service Gateway for WMQ, how to use the Adapter for JDBC-ODBC, and how to access programs written in external programming languages from within TIBCO Object Service Broker.
- *TIBCO Object Service Broker for z/OS External Environments* Provides information on interfacing TIBCO Object Service Broker to various external environments within a TIBCO Object Service Broker z/OS environment. It also includes information on how to access TIBCO Object Service Broker from different terminal managers, how to write programs in external programming languages to access TIBCO Object Service Broker data, how to interface to TIBCO Enterprise Messaging Service (EMS), how to use the TIBCO Service Gateway for WMQ, and how to access programs written in external programming languages from within TIBCO Object Service Broker.

- *TIBCO Object Service Broker Parameters* Lists the TIBCO Object Service Broker Execution Environment and Data Object Broker parameters and describes their usage.
- *TIBCO Object Service Broker Programming in Rules* Explains how to use the TIBCO Object Service Broker rules language to create and modify application code. The rules language is the programming language used to access the TIBCO Object Service Broker database and create applications. The manual also explains how to edit, execute, and debug rules.
- *TIBCO Object Service Broker Managing Deployment* Describes how to submit, maintain, and manage promotion requests in the TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Defining Reports* Explains how to create both simple and complex reports using the reporting tools provided with TIBCO Object Service Broker. It explains how to create reports with simple features using the Report Generator and how to create reports with more complex features using the Report Definer.
- *TIBCO Object Service Broker Managing Security* Describes how to set up, use, and administer the security required for an TIBCO Object Service Broker application development environment.
- *TIBCO Object Service Broker Defining Screens and Menus* Provides the basic information to define screens, screen tables, and menus using TIBCO Object Service Broker facilities.
- *TIBCO Service Gateway for Files SDK* Describes how to use the SDK provided with the TIBCO Service Gateway for Files to create applications to access Adabas, CA Datacom, and VSAM LDS data.

System Administration on the z/OS Platform

The following manuals describe system administration on the z/OS platform:

- *TIBCO Object Service Broker for z/OS Installing and Operating* Describes how to install, migrate, update, maintain, and operate TIBCO Object Service Broker in a z/OS environment. It also describes the Execution Environment and Data Object Broker parameters used by TIBCO Object Service Broker.
- *TIBCO Object Service Broker for z/OS Managing Backup and Recovery* Explains the backup and recovery features of OSB for z/OS. It describes the key components of TIBCO Object Service Broker systems and describes how you can back up your data and recover from errors. You can use this information, along with assistance from TIBCO Support, to develop the best customized solution for your unique backup and recovery requirements.

- *TIBCO Object Service Broker for z/OS Monitoring Performance* Explains how to obtain and analyze performance statistics using TIBCO Object Service Broker tools and SMF records
- *TIBCO Object Service Broker for z/OS Utilities* Contains an alphabetically ordered listing of TIBCO Object Service Broker utilities for z/OS systems. These are TIBCO Object Service Broker administrator utilities that are typically run with JCL.

System Administration on Open Systems

The following manuals describe system administration on open systems such as Windows or UNIX:

- *TIBCO Object Service Broker for Open Systems Installing and Operating* Describes how to install, migrate, update, maintain, and operate TIBCO Object Service Broker in Windows and Solaris environments.
- *TIBCO Object Service Broker for Open Systems Managing Backup and Recovery* Explains the backup and recovery features of TIBCO Object Service Broker for Open Systems. It describes the key components of a TIBCO Object Service Broker system and describes how to back up your data and recover from errors. Use this information to develop a customized solution for your unique backup and recovery requirements.
- *TIBCO Object Service Broker for Open Systems Utilities* Contains an alphabetically ordered listing of TIBCO Object Service Broker utilities for Windows and Solaris systems. These TIBCO Object Service Broker administrator utilities are typically executed from the command line.

External Database Gateways

The following manuals describe external database gateways:

- *TIBCO Service Gateway for DB2 Installing and Operating* Describes the TIBCO Object Service Broker interface to DB2 data. Using this interface, you can access external DB2 data and define TIBCO Object Service Broker tables based on this data.
- *TIBCO Service Gateway for IDMS/DB Installing and Operating* Describes the TIBCO Object Service Broker interface to CA-IDMS data. Using this interface, you can access external CA-IDMS data and define TIBCO Object Service Broker tables based on this data.
- *TIBCO Service Gateway for IMS/DB Installing and Operating* Describes the TIBCO Object Service Broker interface to IMS/DB and DB2 data. Using this interface, you can access external IMS data and define TIBCO Object Service Broker tables based on it.

- *TIBCO Service Gateway for ODBC and for Oracle Installing and Operating*
Describes the TIBCO Object Service Broker ODBC Gateway and the TIBCO Object Service Broker Oracle Gateway interfaces to external DBMS data. Using this interface, you can access external DBMS data and define TIBCO Object Service Broker tables based on this data.

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
<i>TIBCO_HOME</i> <i>OSB_HOME</i>	<p>By default, all TIBCO products are installed into a folder referenced in the documentation as <i>TIBCO_HOME</i>.</p> <p>On open systems, TIBCO Object Service Broker installs by default into a directory within <i>TIBCO_HOME</i>. This directory is referenced in documentation as <i>OSB_HOME</i>. The default value of <i>OSB_HOME</i> depends on the operating system. For example on Windows systems, the default value is C:\tibco\OSB. Similarly, all TIBCO Service Gateways on open systems install by default into a directory in <i>TIBCO_HOME</i>. For example on Windows systems, the default value is C:\tibco\OSBgateways\6.0.</p> <p>On z/OS, no default installation directories exist.</p>
code font	<p>Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:</p> <p>Use MyCommand to start the foo process.</p>
bold code font	<p>Bold code font is used in the following ways:</p> <ul style="list-style-type: none">• In procedures, to indicate what a user types. For example: Type admin.• In large code samples, to indicate the parts of the sample that are of particular interest.• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, MyCommand is enabled: MyCommand [enable disable]
italic font	<p>Italic font is used in the following ways:</p> <ul style="list-style-type: none">• To indicate a document title. For example: See <i>TIBCO ActiveMatrix BusinessWorks Concepts</i>.• To introduce new terms For example: A portal page may contain several portlets. <i>Portlets</i> are mini-applications that run in a portal.• To indicate a variable in a command or code syntax that you must replace. For example: MyCommand <i>PathName</i>

Table 1 General Typographical Conventions (Cont'd)




Convention	Use
Key combinations	Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C. Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.
	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

Table 2 Syntax Typographical Conventions

Convention	Use
[]	An optional item in a command or code syntax. For example: <code>MyCommand [optional_parameter] required_parameter</code>
	A logical OR that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: <code>MyCommand para1 param2 param3</code>

Table 2 Syntax Typographical Conventions

Convention	Use
{ }	<p>A logical group of items in a command. Other syntax notations may appear within each logical group.</p> <p>For example, the following command requires two parameters, which can be either the pair param1 and param2, or the pair param3 and param4.</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command requires two parameters. The first parameter can be either param1 or param2 and the second can be either param3 or param4:</p> <pre>MyCommand {param1 param2} {param3 param4}</pre> <p>In the next example, the command can accept either two or three parameters. The first parameter must be param1. You can optionally include param2 as the second parameter. And the last parameter is either param3 or param4.</p> <pre>MyCommand param1 [param2] {param3 param4}</pre>

Connecting with TIBCO Resources

How to Join TIBCOCommunity

TIBCOCommunity is an online destination for TIBCO customers, partners, and resident experts, a place to share and access the collective experience of the TIBCO community. TIBCOCommunity offers forums, blogs, and access to a variety of resources. To register, go to <http://www.tibcommunity.com>.

How to Access All TIBCO Documentation

You can access TIBCO documentation here:

<http://docs.tibco.com>

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

<http://www.tibco.com/services/support>

- If you already have a valid maintenance or support contract, visit this site:

<https://support.tibco.com>

Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 **Introduction to the Shareable Tools**

This chapter introduces shareable tools, explains the different categories of shareable tools, and provides a list of those tools by category.

Topics

- [Overview, page 2](#)
- [Functional List of Shareable Tools, page 4](#)

Overview

Shareable tools are programs supplied with TIBCO Object Service Broker that expedite rules language programming and application development. The shareable tools perform common TIBCO Object Service Broker functions and facilitate such tasks as string manipulation, mathematical calculation, and object handling.

Main Types of Tools

Each tool is identified according to its behavior, as summarized in this table:

Identifier	Type of Tool	Description
C	Callable	Called from within a rule.
E	Executable	Invoked from the workbench.
CE	Callable & Executable	Called from within a rule or invoked from the workbench.
F	Function	Return a value that can be assigned.
TBL	Table	System interpreted table.

Categories of Tools

The shareable tools can be categorized according to the process they facilitate or the object they affect. The list in [Functional List of Shareable Tools on page 4](#) can be useful if you know what task you want to perform but do not know if there is a tool that can assist you. The tools are divided into the following categories

Category	Page
Batch Jobs (z/OS)	4
Data Object Broker Information and Operations	5
Dates and Times	5
Debugging	6
Definitions of Objects	7

Category	Page
External Databases and Servers	7
External Memory and Routines	8
Installation of Components	8
Load from/Unload to External Files	9
Mathematical Calculation	9
Menus	10
Messages and Message Logs	10
Message Oriented Middleware	10
Printing and Output	11
Promotions	12
Read from/Write to External Files	13
Reports	13
Rules and Rules Libraries	14
Screens	15
Searches for Objects	16
Secondary Indexes	17
Security	17
Selection Lists	18
Session Options and Parameters	18
Strings and Text	18
Strings and Text	18
Table Definitions and Data	21

Functional List of Shareable Tools

Following is a functional list of all the available TIBCO Object Service Broker tools, including the arguments and function of each tool. Some tools appear under more than one category.

Batch Jobs (z/OS)

BATCH – Submits a batch job to a particular queue, views the status of the batch jobs, and views the queues that are available. (E)

BATCHLOAD_CARDS – Defines input and output to the Batch Load utilities (S6BBRTBL/hrnbrtbl). (E)

\$BATCHOPT(*batch_option, option_value*) – Sets the batch options associated with a SCHEDULE TO statement's batch request, which sends the batch job to a queue. (C)

BATCHUNLD_CARDS(*unload_source*) – Defines the control cards required by the Batch Unload utilities. (E)

CICS Channels and Containers

\$BRCONTAINER(*channel, container-list, length, count*) – Lists the 16-character container names and displays the count of the containers associated with the channel. (C)

\$DELCONTAINER(*channel, container*) – Deletes a container from a channel and discards the container's data, if any. (C)

\$GETCONTAINER(*channel, container, area, length, intoccsid*) – Retrieves data from the specified channel container. (C)

\$MOVECONTAINER(*frchannel, frcontainer, tochannel, tocontainer*) – Moves a container and its contents from one channel to another. Afterwards, the source container no longer exists. (C)

\$PUTCONTAINER(*channel, container, area, length, fromccsid, datatype*) – Places data in a container associated with the specified channel. (C)

\$SETCHANNEL(*channel*) – Nominates a channel for passing to a program or transaction. (C)

\$SHOWCHANNEL – Returns the 16-character name of the current channel, if one exists; otherwise, returns blanks. (F)

Data Object Broker Information and Operations

DASTATS(*segment*) – Returns statistical data collected by the Data Object Broker for an individual segment. (F)

DISPLAY_USERS – Displays a list of all users currently logged in to TIBCO Object Service Broker. (E)

HURON_STATS – Displays statistics for performance analysis and problem determination. (E)

OPSTATS(*request-value*) – Returns statistical data collected by the Data Object Broker. (F)

S6BNOTIFY(*msgnum, severity, action, source, subsource, correlation, msgtext*) – Sends a Notification message to TIBCO Hawk. (C)

S6BTROFF – Terminates tracing initiated by the complementary shareable tool S6BTRON. (C)

S6BTRON(*data_set_or_file_name*) – Initiates tracing of rules execution in the current session. (C)

Dates and Times

\$ADD_DATE(*date, component, amount*) – Adds or subtracts a component (such as a day, week, month, or year) to or from a date and returns a new date value. (F)

\$CREATE_DATE(*pic_string, date_string*) – Converts a string with a specified format to a value of semantic type date. (F)

\$DATE_DEFAULT – Returns the default date format used by the installation. (F)

\$DATE_LENGTH(*pic_string*) – Returns the maximum string length of a given date format. (F)

\$DATE_PIC(*pic_string, date*) – Converts a value of semantic type date to a semantic type string. (F)

\$DATE_REF(*component, duration, date, round*) – Adds or subtracts a given number of days to or from a reference date, and converts the number of days returned to units of a day, a week, a month, or a year. (F)

HOURL – Returns the hour of the day when the current transaction started based on the local machine's time zone in which the Execution Environment is running. (F)

LEAPYEAR(*year*) – Returns a logical value indicating whether a given year is a leap year. (F)

LOCALTIME – Returns the local time when the transaction started. (F)

MINUTE – Returns the minute in the hour the transaction started based on the local machine's time zone in which the Execution Environment is running. (F)

REALTIME – Returns a string containing the current time of day. (F)

\$REALTIMER – Returns the number of micro-seconds since 1 January 1980. (F)

SECOND – Returns the second within the minute that the transaction started based on the local machine's time zone in which the Execution Environment is running. (F)

\$SLEEP(*milliseconds*) – Causes the Execution Environment to go dormant. (C)

\$SYSTEMDATE – Returns the date when \$SYSTEMDATE is called based on the local machine's time zone in which the Execution Environment is running. (F)

TIME – Returns a string containing the time of the day when the transaction started. (F)

\$TRXDATE – Returns the start date of the transaction that called this tool based on the local machine's time zone in which the Execution Environment is running. (F)

UTCDATE – Returns the Coordinated Universal Time (UTC) date when UTCDATE is called. (F)

UTCTIME – Returns a string containing the current Coordinated Universal Time (UTC) time. (F)

YEAR – Returns the two-digit year when the transaction started based on the local machine's time zone in which the Execution Environment is running. (F)

Debugging

DEBUG(*rulename*) – Invokes the interactive Rule Debugger. (CE)

\$GTFSET(*function, keyname, userid, termid, all, dob*) – Enables or disables the rules tracing facility in the Execution Environment and the Data Object Broker.

@MESSAGEDUMP – Writes traced messages to this table in HEX form, when @TRACEMESSAGES.DUMP is set to Y. (TBL)

@MESSAGETRACE – Stores table access message requests between the Execution Environment and the Data Object Broker collected when using trace facility. (TBL)

@TRACEMESSAGES – Records message traffic between the Execution Environment and the Data Object Broker. (TBL)

Definitions of Objects

COPY_DEFN(*objecttype, instancename, library, environment, srclocation, destlocation, parentonly*) – Copies the definition of one or more objects from a source location to a destination location. (C)

COPYDEFN – Copies the definition of one or more TIBCO Object Service Broker objects or object sets. (E)

DEFINE_OBJECTSET(*objsetname*) – Defines a set of objects or modifies an existing set. (E)

DELETE_DEFN(*objecttype, instancename, library, environment, location, parentonly*) – Deletes the definition of an object. (C)

DIFF_DEFN(*objecttype, instance1, library1, environment1, location1, instance2, library2, environment2, location2, details*) – Compares the definitions of two objects and list the differences. (F)

DIFF_DEFN – Compares the definitions of one or more pairs of objects and list the differences. (E)

External Databases and Servers

CHANGE_SERVERID(*table_name, old_serverid, new_serverid*) – Updates the server ID of any external TIBCO Object Service Broker data types. (E)

@CONFIGURESERVER(*type*) – Sets and modifies the server configuration parameters for a particular server ID. (E)

DATACOM – Displays a menu to manage the definition of CA-Datacom data. (E)

ESTIMATETBLDFN(*num_fields*) – Returns an estimate of the maximum CTABLESIZE and XTABLESIZE required for each table type. (E)

\$HTTPREQUEST – Issues an HTTP request and returns the response code and result. (F)

IDMS – Displays the main menu used to define a CA-IDMS database to TIBCO Object Service Broker. (E)

IMS – Displays the main menu used to define an IMS/DB database to TIBCO Object Service Broker. (E)

\$OTMA – Invoke IMS OTMA Callable Interface calls. (C)

@OTMA_MAP – Register and allocate storage for use with the @OTMA_MAP table. (TBL)

@PEERSERVERID – Directs remote TIBCO Object Service Broker table accesses to a particular peer server on a remote TIBCO Object Service Broker system. (TBL)

RESETXPARM(*component, entity, parm name, location*) – Resets overrides on server parameters or on default field values set in the Table Definer. (C)

@SERVERERROR(*RETURN_MESSAGE*) – Invokes special parsing and handling of the last message, which resulted from a request to the TIBCO Object Service Broker external DBMS server. (C)

SETXPARM(*component, entity, parm name, value, location*) – Overrides a server parameter or the Table Definer default value for a field at table access time. (C)

@STATICSQL – Defines and generates static SQL to be used to access DB2 data. (E)

External Memory and Routines

\$GETENVCOMMAREA(*segment#*) – Retrieves data passed into TIBCO Object Service Broker from a calling environment that is not TIBCO Object Service Broker. (F)

HLIPREPROCESSOR(*hostlang, imbedlang, infile, outfile, listfile, options*) – Invokes a language pre-processor to run against COBOL source programs that contain embedded TIBCO Object Service Broker access statements or SQL statements. (C)

@MAP – Registers and allocates storage for use with MAP tables. (TBL)

RETURN_CODE – Returns the return code from the last call of a TIBCO Object Service Broker external routine. (F)

S6BCALL – Invokes a TIBCO-supplied callable routine that requires a specialized environment. (C)

S6BFUNCTION – Invokes a TIBCO supplied function that requires a specialized environment. (F)

\$SETENVCOMMAREA(*value, segment#*) – Passes data from TIBCO Object Service Broker into a calling environment that is not TIBCO Object Service Broker. (F)

\$SETSESSIONEND(*action, value*) – Sets what execution is to take place when a TIBCO Object Service Broker session ends by returning data from the session to an external environment. (C)

Installation of Components

@INSTALL(*component [path]*) – Requests the installation of the specified component. (CE)

@UNINSTALL(*component*) – Requests that the specified component be uninstalled. (CE)

Load from/Unload to External Files

BATCHLOAD_CARDS – Defines input and output to the Batch Load utilities (S6BBRTBL/hrnbrtbl). (E)

BATCHUNLD_CARDS(*unload_source*) – Defines the control cards required by the Batch Unload utilities. (E)

EXPOCC_SIZE(*table*) – Returns the minimum record size required to hold the occurrences of the table being unloaded. (F)

LLOAD(*importfile, media*) – Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded from files with names in mixed case or lowercase. (CE)

LOAD(*importfile, media*) – Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded. (CE)

LOADER – Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded, with selection control. (CE)

UNLOAD – Unloads definitions of valid TIBCO Object Service Broker object types from a source system to a z/OS data set or a Windows or Solaris file. Data from table object types could also be unloaded. (E)

UNLOAD_DATA(*tablespec, selection, location*) – Unloads the data of a table to a z/OS data set or a Windows or Solaris file. (C)

UNLOAD_DEFN(*objecttype, objectname, library, location, presentationenv, parentonly*) – Unloads the definition of a TIBCO Object Service Broker object to a z/OS data set or a Windows or Solaris file. (C)

UNLOADLIBRARY(*library, location*) – Unloads all rules in the specified library at the specified location to a z/OS data set or a Windows or Solaris file. (C)

Mathematical Calculation

ABS(*value*) – Returns the absolute value of a number. (F)

MAX(*x, y*) – Returns the larger of two given values. (F)

MIN(*x, y*) – Returns the smaller of two given values. (F)

MOD(*dividend, divisor*) – Returns the modulus from dividing the dividend by the divisor. The function MOD handles negative dividends and divisors. (F)

NUM_CHK(*val*) – Determines if a given string satisfies the TIBCO Object Service Broker definition of a numeric literal. (F)

RANDOM(*rangelimit*) – Returns a random integer greater than or equal to 1 and less than or equal to the specified limit. (F)

RANDOMSEED(*seed*) – Sets the starting seed for the random number generator. (C)

REMAINDER(*dividend, divisor*) – Returns the remainder from dividing the dividend by the divisor. (F)

ROUND(*value*) – Returns the specified value rounded to the nearest integer. (F)

Menus

DEFINE_MENU(*menu*) – Creates and modifies menus and login screens used within TIBCO Object Service Broker user-defined applications. (E)

DISPLAY_MENU(*menuname*) – Calls a specific menu into an application. (C)

Messages and Message Logs

ENDMSG(*message*) – Sets the transaction completion message. (C)

GETENDMSG – Returns the current value of the end-of-transaction message. (F)

LOG_BROWSE – Displays the contents of the message log. (C)

MESSAGE(*utility, msg_num, tokenlist*) – Returns a customized message by taking a root message in the MESSAGES table and inserting customizing tokens. (F)

MESSAGE_LOG(*msglog, destin*) – Preserves the contents of the message log across transactions. (C)

MSGLOG(*string*) – Inserts the specified string as a line in the TIBCO Object Service Broker message log. (C)

RETURN_SYMSG – Returns the last \$SYSCALL system error message when an exception is raised. RETURN_SYMSG is a low-level tool that must be called immediately after an exception is trapped. (F)

S6BCALL(*'api_call', parameters*) – Invokes a TIBCO-supplied callable routine that requires a specialized environment. (C)

S6BNOTIFY(*msgnum, severity, action, source, subsource, correlation, msgtext*) – Sends a Notification message to TIBCO Hawk. (C)

Message Oriented Middleware

@MOMCLOSE(*connection, queue*) – Closes a Message Oriented Middleware (MOM) message queue. (F)

@MOMCOMMIT(*connection*) – Commits all changes to queues from a single Message Oriented Middleware (MOM) message manager. (C)

@MOMCONNECT(*name*) – Connects to a Message Oriented Middleware (MOM) message queue (MQ) manager. (F)

@MOMDISCONN(*connection*) – Disconnects from a Message Oriented Middleware (MOM) message manager. (F)

@MOMGET(*connection, queue, table*) – Reads a message from a Message Oriented Middleware (MOM) message queue. (C)

@MOMINIT(*buflen, mon_type*) – Identifies the type of Message Oriented Middleware (MOM) message manager, and initializes its environment (map and control structures) to enable subsequent @MOM calls. (C)

@MOMMAPLENGTH(*table_name*) – Returns the length of a MAP table. (F)

@MOMOPEN(*connection, name*) – Opens a Message Oriented Middleware (MOM) message queue. (F)

@MOMOPTION(*description*) – Queries the numeric equivalent of a Message Oriented Middleware (MOM) option. (F)

@MOMPUT(*connection, queue, table, len*) – Writes a message to a Message Oriented Middleware (MOM) message queue. (C)

@MOMROLLBACK(*connection*) – Backs out all database changes from a single Message Oriented Middleware (MOM) message manager since the start of the transaction or since the previous @MOMCOMMIT. (C)

@MOMSETOPT(*description*) – Sets a MOM option to a specified value. (C)

@MOMSPECIALCMD(*manager_name, queue_name, command*) – Sends a Message Oriented Middleware (MOM) command to a queue listener task. (C)

@MOMVALIDRC – Checks the return code of a previous command. (C)

@MQSMAP and **@MQSMAP_PORT** – Registers and allocates storage for use with the MQSMAP table. @MQSMAP is for use on z/OS and @MQSMAP_PORT on Open Systems. (TBL)

Printing and Output

\$BLANKPAGE(*titles_yn*) – Outputs a blank page. (C)

\$FLUSHPRINT – Releases output into the print spool. (C)

\$NEWPAGE – Positions subsequent output to the top of a new page. (C)

PRINT_DATA(*tablespec, select, sourceloc*) – Prints the data of a TIBCO Object Service Broker table. (C)

PRINT_DEFN(*object, instance, library, environment, srcloc, parentonly*) – Prints the definition of a TIBCO Object Service Broker object. (C)

\$PRINTFIELD(*string, pos, length, fill, just*) – Writes the specified string into the current printline. (C)

\$PRINTLINE(*text*) – Prints a string. (C)

PRINTTABLE(*tablespec, pagelength, pagewidth, media*) – Prints a table. (C)

PRT_VSCR(*vscr, page_length, page_width, page_start, media, mask*) – Prints the screen fields of a defined screen in a page format, with or without a mask. (C)

\$PUTLINE – Prints the current line constructed by \$PRINTFIELD. (C)

\$RESETPRINT(*length, width, page_number, media*) – Resets the output arguments. (C)

\$SETP#POS(*line_number, left_string, center_string, right_string*) – Defines the position and content of page number lines. (C)

\$SETPRINT(*length, width, page_number, media, clear_title_yn*) – Initializes the print attributes or, if they are already set, uses it to clear the titles for the output on the following pages. (C)

\$SETTITLE(*line_number, left_string, center_string, right_string*) – Sets a title or footer to be printed on subsequent pages of output. (C)

\$SKIPLINE(*count*) – Outputs zero or more blank lines. (C)

\$TOCPRINT(*fill_char*) – Prints the table of contents. (C)

\$TOCPUT(*section_name, spacing, numbering_yn*) – Puts a line in the table of contents. (C)

Promotions

ADMIN_RIGHTS – Obtains, releases, or transfers the promotion rights on objects. (E)

DBMAINTLVL – Displays the maintenance level of your TIBCO Object Service Broker database, including any database PTFs applied beyond the maintenance level. (E)

MANAGE_APPLY – Invokes the Promotion facility on the target system. (E)

MANAGE_REQUESTS – Invokes the Promotion facility on the source system. (E)

MANAGE_RIGHTS – Releases or transfers a user's promotion rights on rules, screens, reports, menus, object sets, and tables. (E)

@PROMBINDOBS – Restores the bind flag settings for the objects updated by @PROMUNBINDOBS. (E)

PROM_MAIN – Invokes directly the Promotion system. (E)

[@PROMUNBINDOBS](#) – Stores the current setting of the bind flag for a set of objects and resets the values to N in the metadata tables. (E)

[RMANAGE_REQUESTS](#) – Manages change requests for systems where the source system is remote to the target system. (E)

Read from/Write to External Files

[ALLOCDSN](#)(*ddname, dsname*) – Allocates a file to a z/OS DDNAME. (C)

[@CLOSEDSN](#) – Closes and frees the current file. (C)

[\\$LISTDSN](#)(*dsname_level, buffer_address*) – Lists the non-VSAM data sets and Generation Data Group (GDG) data sets of a certain level, using the z/OS Catalog Search Interface services. (C)

[\\$LISTPDS](#)(*pds_name, buffer_address, member_name*) – Lists the member names of a partitioned data set (PDS), or retrieves the statistics for a PDS member. (C)

[@OPENDSN](#)(*dsname*) – Specifies the name of the file that is subsequently used by @READDSN or @WRITEDSN. (C)

[@READDSN](#) – Returns the next record from the current file. (F)

[@WRITEDSN](#)(*string*) – Writes a record to the current file. (C)

[XMLPARSE](#)(*docname, validate, docsource, docdata*) – Initiates the parsing of an XML document. (C)

[XMLSTART](#)(*xmldocname, predicate, parm*) – Generates an XML document based on the passed data access arguments. (C)

[XMLSTARTDSN](#)(*outdsn, xmldocname, predicate, parm*) – Generates an XML document based on the passed data access arguments and places it in the specified file. (C)

[XMLSTARTSETDEST](#)(*tablespec, fieldspec*) – Sets up the output table and field for XMLSTART. (C)

[XMLSTARTTAB](#)(*tablename, format, predicate, parm*) – Returns the data of a table instance to the OIG client. (C)

Reports

[DEFINE_REPORT](#)(*reportname*) – Defines a new TIBCO Object Service Broker report or modifies an existing one. (E)

[GENERATE_REPORT](#)(*reportname*) – Defines a new report or modifies an existing report using the Report Generator. (E)

[\\$RPTIMMEDIATE](#)(*reportname, media*) – Sends the records to the output as they are read, without sorting. (C)

- \$RPTOCCLIMIT**(*reportname, occlimit*) – Limits the number of occurrences used to generate the report. (C)
- \$RPTOVERLAP**(*report, reporttable, reportfield, BLANKOVERLAP*) – Designates the report tables or report fields that are not to be printed on the overlapping page of a merged report. (C)
- \$RPTPARMS**(*reportname, length, width, eject, pagenumber*) – Controls explicitly the physical output of a report. (C)
- \$RPTPRINT**(*reportname, media*) – Prints a report to the medium specified. (C)
- \$RPTSKIPLINES**(*reportname, reporttable, element, linesbefore, linesafter*) – Controls explicitly the spacing of a report. (C)
- \$SETRPTATTRIBUTE**(*report, attribute, value*) – Sets the attributes of the report that is to be printed. (C)
- \$SETRPTMEDIUM**(*report, mediumtype, medium*) – Sets the medium to which a report is to be printed. (C)

Rules and Rules Libraries

- \$CALLRULE**(*rulecall*) – Invokes a procedural rule. (C)
- CHANGERULE** – Makes multiple text changes across multiple rules in a library. (CE)
- COPYLIB**(*source_lib, dest_lib*) – Copies all the rules from a source library to a destination library. (C)
- DEFINE_LIBRARY**(*libraryname*) – Defines a new library, displays a list of the rules in a library, or displays a list of the rules libraries. (E)
- EDITRULE**(*rulename*) – Defines a new TIBCO Object Service Broker rule or modifies an existing one. (E)
- \$EXCEPTION** – Returns the name of the last exception signalled within the current transaction by either a SIGNAL statement, a \$SIGNAL call, or the system (GETFAIL, ZERODIVIDE, and so on). (F)
- \$EXCEPTIONOBJECT** – Returns the name of the object (for example, a table) associated with the last exception signalled within the current transaction, if that exception is of the type that can be trapped with an ON `exception_name object_name` statement. (F)
- \$FUNCTION**(*rulecall*) – Invokes a functional rule. (F)
- INSTALLIB** – Returns the name of the currently designated installation library. (F)
- LIBID** – Returns the name of the currently designated local library. (F)

NOOP – Does nothing. (C)

\$RULE_EXISTS(*rule*) – Checks whether a rule with the given name would be a candidate for execution. The rule can be a rule in the current search path, a TIBCO Object Service Broker routine, or an external routine with an available and executable load module. (F)

\$RULENAME(*level, transactioncount*) – Retrieves the name of a rule from the current execution stack. (F)

RULEPRINTER(*rule*) – Prints a rule or prints an application structure using the root rule as the base. (E)

SEARCHLIB – Searches all rules or specified rules in a library for a given string. (CE)

\$SIGNAL(*exception, tablename*) – Raises the specified exception. (C)

SYSTEMLIB – Returns the name of the currently designated system library. (F)

\$TRXMODE – Retrieves the transaction mode of the current rule. (F)

\$TYPECAST – Converts a variable according to the arguments supplied. (F)

Screens

\$BEEP(*repetition*) – Issues the specified number of beeps from the terminal. (C)

CONFIRMACTION(*screen, confirmmsg, key, defaultmsg, table, commandfield*) – Issues a confirmation message for a PF key action or for a specified command. (C)

CURSOR_FLDCOL(*screen*) – Returns the relative column number within the field containing the cursor. (F)

CURSORFIELD(*screen*) – Returns the name of the field where the cursor is located. (F)

CURSOROCC#(*screen*) – Returns the occurrence number within the screen table where the cursor is positioned. (F)

CURSOROCC_VALUE(*screen, scrtbl, scrfld*) – Returns the value of a particular screen field that is selected by the cursor. (F)

CURSORTABLE(*screen*) – Returns the name of the screen table where the cursor is positioned. (F)

DELETESCREENDATA(*screen*) – Deletes all the occurrences from all the screen tables of a screen. (C)

DRAW(*screenname*) – Defines a new TIBCO Object Service Broker screen or modifies an existing one. (E)

ENTERKEY(*screen*) – Returns the name of last key used when the specified screen appeared. (F)

EVENTSCREEN – Returns the name of the screen that activated the current screen validation rule. (F)

EXIT_DISPLAY – Signals the standard exception EXIT_DISPLAY. (C)

FCNKEY_MSG(*screen*) – Creates a string containing the function keys defined for a screen. (F)

\$GETATTRIBUTE(*screen, table, field, attribute*) – Queries the current attributes for the field of the screen table, in the specified screen. (F)

\$GETCOLOUR(*screen, table, field, color_type*) – Queries the current color of a screen field. (F)

@PRESENTATIONENV – Returns the name of the presentation environment for the current session. (F)

PROCESS_FCNKEY(*screen*) – Processes the function keys while a screen is being displayed. (C)

SCREENCOL – Returns the number of columns on the user's physical screen. (F)

SCREENMSG(*name, msg*) – Displays the given message in the message area of the specified screen. (C)

SCREENROW – Returns the number of rows on the user's physical screen. (F)

\$SETATTRIBUTE(*screen, table, field, attribute, flag*) – Sets attributes for the field of the screen table, in the specified screen. (C)

\$SETCOLOUR(*screen, table, field, color_type, color*) – Sets the color of a screen field. (C)

SETCURSOR(*screen, table, field*) – Positions the cursor in the field of the screen table, in the specified screen. (C)

SETCURSOR_POS(*screen, table, field, occurrence_number, column_offset*) – Positions the cursor in the column of the field of the occurrence, in the screen table of the screen. (C)

Searches for Objects

CROSSREFSEARCH(*querystring, querykind, library*) – Searches the cross reference index of the specified library to answer a query. (C)

KEYWORDMGR – Ensures that the TIBCO Object Service Broker keyword system conforms to the established formatting standards and that the keyword index table is up-to-date. (E)

KEYWORDSEARCH(*querystring, object_type*) – Searches the keyword index of a default library to answer a query. (C)

REFMAKER(*library*) – Rebuilds the global cross reference index. (E)

SEARCH – Searches the keyword or cross reference indexes to answer a query. (E)

SEARCHLIB – Searches all rules or specified rules in a library for a given string. (CE)

Secondary Indexes

SIXBUILD(*table, secondary_key*) – Creates a secondary index online for a TDS table. (C)

SIXBUILD_CARDS – Defines the control cards required by the Batch Secondary Index Build utilities. (E)

SIXDELETE(*table, secondary_key*) – Deletes an existing secondary index. (CE)

Security

AUDITLOG – Invokes the Query Audit Log tool. (CE)

BATCH_ENABLE(*wipe_existing*) – Enables all the object sets previously processed using @MAKEMEMBERS. (C)

CREATEUSERS(*input_table, modeluser*) – Creates a list of new user IDs and adds them to the TIBCO Object Service Broker system. (E)

@MAKEMEMBERS(*objectset*) – Creates the member list for an object set to be enabled through the BATCH_ENABLE utility. (CE)

@MNG_USERS – Modifies your user security profile. (E)

PURGELOG_BATCH(*fromdate, todate, file*) – Purges the audit log data from the TIBCO Object Service Broker audit log table and archives it to an external file. (CE)

PURGELOG_SCREEN – Specifies the archive file for the audit log data and purges the audit log data after writing it to the specified archive file. (CE)

SEC_REBIND(*object, parmcats, name*) – Rebinds all security data previously bound in the Execution Environment storage. (C)

SECURITY – Invokes the TIBCO Object Service Broker Security Manager main menu. (E)

Selection Lists

DEFINE_OBJLIST(*table*) – Defines, for a table, an object list to appear using the Object Manager or modifies an existing object list definition. (E)

OBJECT_MGMT(*tablespec*) – Displays the contents of a table and enables a predefined set of commands that are unique to the table to operate on the display. (E)

OBJECTMGR(*tablespec*) – Displays the contents of a table and enables a predefined set of commands that are unique to the table to operate on the display. (C)

OPTIONLISTER(*optionlistname*) – Displays options in columns and returns the ones selected (C).

SELECT_OBJ(*name, type, unit, author, library, location, children, subtype*) – Provides a screen that can be used to list and select objects that meet specified criteria. (C)

Session Options and Parameters

\$GETOPT(*option_name*) – Returns the value of a session parameter or option. (F)

\$GETTRANSACTION(*name*) – Gets a transaction name set by \$SETTRANSACTION. (F)

REMOTELOCATION – Returns the current value of the default remote location. (F)

@SESSION – Alters session-related items maintained by this table. (TBL)

@SESSIONCOUNTS – Obtains information on events occurring within the Execution Environment during a session. (TBL)

SESSMGR – Displays the login interface to the Session Manager (workbench).

\$SETOPT(*parameter, value*) – Sets the value of a session parameter or option. (C)

\$SETTRANSACTION(*field, value*) – Returns the current name of a TIBCO Object Service Broker transaction and sets a new name. (F)

SETREMOTELOC(*remoteloc*) – Sets the default remote location for distributed data processing. (C)

USERID – Returns a string containing the user ID. (F)

Strings and Text

FROM_UNICODE(*unistring, externalcodepage*) – Converts a Unicode string to Raw Data encoded in an external Code page.(F)

GEN_TED(*tablespec, screenname, screentablename*) – Presents a screen where text can be entered and edited under the control of the text editor. (C)

GENBIN(*value, length*) – Returns a syntax V string containing the same internal binary value as the input numeric value, right-justified. (F)

GENFLOAT(*value, length*) – Returns a syntax V string containing the same internal float representation as the input float value, left-justified. (F)

GENPACK(*value, length, decimal*) – Returns a syntax V string containing the same internal packed decimal value as the input syntax P value, right-justified. (F)

\$GET_DECIMALS(*value*) – Retrieves the number of decimal places for an expression. (F)

\$GET_MAXSIZE(*value*) – Retrieves the dictionary size of an expression. (F)

\$GET_SIZE(*value*) – Retrieves the size of an expression. (F)

\$GET_SYNTAX(*value*) – Retrieves the syntax of an expression. (F)

\$GET_TYPE(*value*) – Retrieves the semantic type of an expression. (F)

\$GETBINARY(*string, offset, length*) – Stores character data in binary format. (F)

GETCHAR(*string*) – Returns the first character from the specified string, removing it from the string. (F)

\$GETFLOAT(*string, offset, length*) – Stores character data in floating format. (F)

\$GETPACKED(*string, offset, length*) – Stores character data in packed decimal format. (F)

HEADSTRING(*string, length*) – Returns the head portion of the specified string. (F)

LENGTH(*string*) – Returns the length of the specified string. (F)

LIT_TO_VAL(*string*) – Converts a string to a typeless value as described in the string. (F)

LOWER_EBCDIC(*string*) – Converts a string to lowercase EBCDIC characters. (F)

LOWER_UNICODE(*string*) – Converts a string to lowercase Unicode characters. (F)

LOWERCASE(*string*) – Converts all uppercase characters in a string to lowercase characters. (F)

MATCH(*string, pattern*) – Returns the starting position, in characters, of the specified pattern in the specified string, relative to the start of the string. (F)

PAD(*string, length, padcharacter, just*) – Returns a string padded to a specified length using a pad character, positioning the string to the left, right or center of the padding. (F)

PARSE(*grammar_usage, string*) – Breaks up an input string into tokens and applies grammar rules to the tokens. (C)

PATTERN_MATCH(*string, pattern*) – Determines whether a string matches a given pattern. (F)

PEEL(*peelchars, string*) – Returns the result of removing the specified leading and trailing characters from the specified string. (F)

PEEL_HEAD(*char, string*) – Removes the specified leading characters from a given string. (F)

PEEL_TAIL(*char, string*) – Removes the specified trailing characters from a given string. (F)

\$PIC(*value, mask*) – Returns a number in a format specified by a mask. (F)

QUOTE(*string*) – Returns a string with single quotation marks around it and doubles any single quotation marks in the string. (F)

SCRIPT(*source, dest*) – Uses commands to format text from a table and store the formatted text in another table. (C)

SEARCH_REPLACE(*input_string, replace_this, with_this, else_with_this*) – Replaces all occurrences of a pattern with specified characters. (F)

SUBSTRING(*string, start, length*) – Returns a selected portion of a string. (F)

TAILSTRING(*string, length*) – Returns the tail portion of the string. (F)

TED(*text_input*) – Displays a table for text editing. (E)

TEXTSETUP(*setupname*) – Defines a setup for formatting a text document. (E)

TOKEN(*string*) – Parses an input string and returns the first token and the string with the token removed. (F)

TO_UNICODE(*rdstring, externalcodepage*) – Converts a raw data string encoded in an external code page to Unicode. (F)

\$TYPECAST(*type, syntax, size, decimals, value*) – Converts a variable according to the arguments supplied. (F)

\$UNPIC(*picVal, mask*) – Determines the original value submitted given a masked value produced by \$PIC and the display mask that produced it. (F)

UNQUOTE(*string*) – Returns a string with the single quotation marks removed. (F)

UPPER_EBCDIC(*string*) – Converts a string to uppercase EBCDIC characters. (F)

UPPER_UNICODE(*string*) – Converts a string to uppercase Unicode characters. (F)

UPPERCASE(*string*) – Converts all lowercase characters in a string to uppercase characters. (F)

VAL_TO_LIT(*string*) – Converts a value to a string containing a token describing its value. (F)

VALID_NAME(*name*) – Determines if a given string satisfies the TIBCO Object Service Broker definition of an identifier. (F)

Table Definitions and Data

BROWSER(*tablespec*) – Displays the contents of a TIBCO Object Service Broker table for viewing. (CE)

CLEARTABLE_APPL(*table, select*) – Deletes occurrences from a table or table instance. (E)

\$CLRTAB(*tablename, parm1, parm2, parm3, parm4*) – Deletes (clears) the data rows from a table or table instance without reading the data rows. (C)

COPY_DATA(*srctabspec, select, desttabspec, srclocation, destlocation, overwrite*) – Copies data from one table or table instance to another table or table instance. (C)

COPYTABLE_APPL – Copies selected occurrences from a source table to a destination table. (E)

COUNTOCCURRENCES(*table, selection*) – Returns the number of occurrences that meet a selection criteria. (F)

DEFINE_TABLE(*tbl_name*) – Defines a new TIBCO Object Service Broker table or modifies an existing one. (E)

DELETE_DATA(*tablespec, select, location*) – Deletes the data from a table or table instance. (C)

DIFF_DATA(*table1, field1, location1, selection1, table2, field2, location2, selection2, printresult*) – Compares the data of two tables or table instances and lists the differences. (F)

EVENTFIELD – Returns the name of the field of the subview table that activated the current derived field rule, trigger rule, or validation rule. (F).

EVENTSUBVIEW – Returns the name of the subview table that activated the current derived field rule. (F)

EVENTTABLE – Returns the name of the table that activated the current derived field rule, trigger rule, or validation rule. (F)

FLDMGR(*fieldname*) – Adds fields to the global field dictionary. (E)

FORALLA(*table, parm, selection, ordering*) – Returns the first table occurrence that satisfies the selection criteria. Use if the value of every table parameter and every selection criterion is 99 or fewer characters long. (C)

@FORALLA(*table, parm, selection, ordering*) – Returns the first table occurrence that satisfies the selection criteria. Use if the value of any table parameter or of any selection criterion is 100 or more characters long. (C)

FORALLB(*table*) – Returns the next table occurrence that satisfies the selection criteria following a call to FORALLA. (C)

FORALLE(*table*) – Releases internally the resources used by FORALLA on a table. (C)

MOV TAB(*tablename, segmentID*) – Changes the segment number of a table. (CE)

INDEXCHK – Estimates the maximum number of data rows a table can contain before reaching the maximum index levels. (E)

NLS – Enables the database administrator to set code page values in translation tables. (E)

OBJECTMGR(*tablespec*) – Displays the contents of a table and enables a predefined set of commands that are unique to the table to operate on the display. (C)

PARMVALUE(*parmname*) – Returns the value of the parameter from the table that was accessed when the trigger or validation rule was activated. (F)

PARSE_TAM(*string*) – Breaks up an input string into a table specification, and optionally, the WHERE clause and the ORDERED clause of the corresponding table access statement. (C)

PROCESS_TABLE(*tablespec, selection, ordering, processrule*) – Provides specific processing for every occurrence in a table that is selected, ordered, or both selected and ordered. (C)

RETURN_MESSAGE – Returns the system error message whenever an exception is raised. RETURN_MESSAGE is a low-level tool that must be called immediately after an exception is trapped. (F)

SETNLSBIT(*table, flag*) – Sets the NLS bit for the specified table, in the RESERVED field of the TABLES table. (C)

SIMPLESELECT(*selection*) – Processes a selection string into a format that can be used by the FORALLA tool. (C)

SOE(*tablespec*) – Edits a single occurrence in a table. (CE)

STE(*tablename*) – Invokes the Table Editor. (CE)

STEBROWSE(*input*) – Views the contents of a TIBCO Object Service Broker table. (E)

TABLEPRINT(*tablespec*) – Prints the contents of a table or of a set of joined tables.
(E)

Trigger or Validation Rules

EVENTSCREEN – Returns the name of the screen that activated the current screen validation rule. (F)

EVENTTABLE – Returns the name of the table that activated the current derived field rule, trigger rule, or validation rule. (F)

PARMVALUE(*parmname*) – Returns the value of the parameter from the table that was accessed when the trigger or validation rule was activated. (F)

Chapter 2 **Using User Exits in Workbench Tools**

This chapter describes how to use user exits in workbench tools.

Topics

- [Overview, page 26](#)
- [Description of the User Exits, page 27](#)
- [How to Use the Exits, page 29](#)

Overview

Purpose of the User Exits

User exits are provided to customize the access and activity for a subset of the workbench tools provided with TIBCO Object Service Broker. You can implement these exits in the following places when executing one of these tools:

- The entry to the tool, where the exit can be used for some kind of validation, such as verifying user access defined to meet a standard.
- Immediately before the definition of an object is committed to the database. This can be used for validation, and possibly manipulation, of the object being saved.
- After the definition is committed into the database. At this point the object can still be manipulated and logging can be implemented.

Tools Supporting the User Exits

The following tools, supplied with the default developer workbench, support the use of the user exit rules:

- Rule Editor
- Rule Debugger
- Table Browser
- Table Editor
- Single Occurrence Editor
- Screen Definer
- Table Definer
- Report Definer
- Library Definer
- Screen Table Definer
- Report Table Definer

Description of the User Exits

There are three user exit rules. All are functions and all have the same arguments. The purpose of each of these is described below.

@ENTRY_VALIDATE(*caller, type, name, library, new*)

This rule is called before the tool makes its first display. If it returns “Y”, the tool continues. If it returns any other string, the tool terminates with no display, and the returned string appears as a message on the workbench. Tools where you can create a new object by changing the name or type of the object call this rule again for the new definition.

@PRE_SAVE_OBJECT(*caller, type, name, library, new*)

This rule is called after PF3 is pressed and before changes are committed into the database. If it returns any string except “Y”, the returned string appears as a message by the tool and the tool does not save the changes. At this point, a user can press PF12 to exit, or possibly change the object so it can be saved.

@SAVED_OBJECT(*caller, type, name, library, new*)

This rule is called after PF3 is pressed and the changes are committed into the database. If it returns any string except “Y”, the returned string appears as a message on the workbench.

Arguments

Use the arguments as follows:

<i>caller</i>	<p>The name of the entry rule of the tool calling the exit. The rule name for each of the tools are:</p> <ul style="list-style-type: none">• Rule Editor – EDITRULE• Rule Debugger – DEBUG• Table Browser – STEBROWSE• Table Editor – STE• Single Occurrence Editor – SOE• Screen Definer – DRAW• Table Definer – DEFINE_TABLE• Report Definer – DEFINE_REPORT• Library Definer – DEFINE_LIBRARY• Screen Table Definer – STABLE_PAINT• Report Table Definer – PAINT_RPT
<i>type</i>	<p>The type of object being defined. This is the type used by the object model, such as TEM_TABLE. For other possible values, refer to the @OBJECTTYPES table.</p>
<i>name</i>	<p>The name of the object.</p>
<i>library</i>	<p>The name of the library for a rule, otherwise an empty string.</p>
<i>new</i>	<p>“Y” if the object is new, “N” otherwise. The Table Editor and Browser pass “Y” for an empty table, and the Single Occurrence Editor passes “Y” for a new occurrence.</p>

How to Use the Exits

Location of the Rules

The exit rules are stored in the system (COMMON) library. They are provided as stub rules and always return the string "Y". This causes the affected tools to function as they did in the past.

Modifying an Exit

Using the appropriate sample stub rule as a base, modify the exit rule. The modified rule must return a string to the tool. Save the modified rule in the appropriate library, usually the installation (SITE) library.

Set the Search Path

You must ensure that the correct library search path for the modified tool is set in the workbench menu definition. Refer to *TIBCO Object Service Broker Defining Screens and Menus* for details about using the DEFINE_MENU tool to define workbenches and about using DEFINE_OBJLIST for object manager prompts.

Examples

The following examples show how you can:

- Force users to observe a naming convention on rules
- Restrict a certain set of users to define only TEM or SES tables

The rules @ENTRY_VALIDATE and @PRE_SAVE_OBJECT are changed to test the CALLER argument.

RULE EDITOR ==>		SCROLL: P	
@ENTRY_VALIDATE(CALLER, TYPE, NAME, LIBRARY, NEW);			

CALLER = 'EDITRULE';		Y	N N
CALLER = 'DEFINE_TABLE';			Y N

RETURN(VALIDRULENAME(NAME, LIBRARY));		1	
RETURN(VALIDTABLETYPE(TYPE, NEW));			1
RETURN('Y');			1

If @ENTRY_VALIDATE is called by the Rule Editor, it calls VALIDRULENAME. If it is called by the Table Definer, it calls VALIDTABLETYPE.

RULE EDITOR ==>		SCROLL: P	
@PRE_SAVE_OBJECT(CALLER, TYPE, NAME, LIBRARY, NEW);			

CALLER = 'EDITRULE';		Y	N N
CALLER = 'DEFINE_TABLE';			Y N
-----		+	
RETURN(VALIDRULENAME(NAME, LIBRARY));		1	
RETURN(VALIDTABLETYPE2(TYPE));			1
RETURN('Y');			1

If @PRE_SAVE_OBJECT is called by the Rule Editor, it calls VALIDRULENAME.
If it is called by the Table Definer, it calls VALIDTABLETYPE2.

RULE EDITOR ==>		SCROLL: P	
VALIDRULENAME(NAME, LIBRARY);			

LIBRARY -= 'ORDERED';		Y	N N
HEADSTRING(NAME, 3) -= 'ED_';			Y N
-----		+	
RETURN('Y');		1	1
RETURN('You can only edit rules starting with "ED_" in '			1
'this library');			

VALIDRULENAME ensures that if a rule is to be edited or saved in this library, it must start with ED_.

RULE EDITOR ==>		SCROLL: P			
VALIDTABLETYPE(TYPE, NEW);					

NEW;		Y	N	N	N
HEADSTRING(USERID, 2) -= 'WE';			Y	N	N
TYPE = 'TEM_TABLE';				Y	N
TYPE = 'SES_TABLE';					Y

RETURN('Y');		1	1	1	1
RETURN('Your user group can only define TEM and SES tables');					1

VALIDTABLETYPE and VALIDTABLETYPE2 ensure that users whose user ID starts with WE can define only a SES_TABLE or a TEM_TABLE.
VALIDTABLETYPE first checks the NEW flag because when a new definition is created, it starts out as a TDS_TABLE. Without this extra check, a user whose user ID starts with WE would be unable to create a new table.

```

      RULE EDITOR ==>
VALIDTABLETYPE2(TYPE);
      SCROLL: P
-
- -----
- HEADSTRING(USERID, 2) ¬= 'WE';           | Y N N N
- TYPE = 'TEM_TABLE';                     |   Y N N
- TYPE = 'SES_TABLE';                     |     Y N
- -----+-----
- RETURN('Y');                           | 1 1 1
- RETURN(                                |      1
-   'Your user group can only define TEM and SES tables'); |
- -----
```


Chapter 3 **Tools**

This chapter describes all the shareable tools. The tools are listed in alphabetical order ignoring the \$ and @ that are in front of some tools' names. For example, the tools `CLEARTABLE_APPL`, `@CLOSEDSN`, and `$CLRTAB` follow each other in the list.

ABS

Returns the absolute value of a number. (F)

Invocation `number = ABS(value)`

<i>number</i>	On return, contains the absolute value
<i>value</i>	The integer, packed decimal, or float number to be evaluated

Example The following rule determines the absolute value of -1 and prints it to the workbench or screen:

```
RULE EDITOR ==>                                SCROLL: P
ABS_1;
_ LOCAL NUMBER, VALUE;
_ -----
_                                     +-----
_ VALUE = - 1;                        | 1
_ NUMBER = ABS(VALUE);                | 2
_ CALL ENDMSG('THE ABSOLUTE VALUE OF ' || VALUE || ' IS ' || | 3
_   NUMBER || '.');                    |
_ -----
```

The following end message appears after the rule is executed:
THE ABSOLUTE VALUE OF -1 IS 1.

\$ADD_DATE

Adds or subtracts a component (such as a day, week, month, or year) to or from a date and returns a new date value. (F)

Invocation `new_date = $ADD_DATE(date, component, amount)`

<i>new_date</i>	On return, contains the new date value.
<i>date</i>	A specific date.
<i>component</i>	<p>Must be one of the following date components:</p> <ul style="list-style-type: none">• D (day)• W (week)• M (month)• Y (year) <p>Its syntax is C (fixed-length character string) with a length of one.</p>
<i>amount</i>	<p>An integer specifying the amount to be added or subtracted.</p> <p>Its syntax is B (binary) with a length of four.</p>

- Usage Notes**
- The result appears in the default date format for your system installation.
 - If the component being added or subtracted is a month, \$ADD_DATE returns the last possible date in the resultant month if the resultant day does not exist in the resultant month. For example, one month subtracted from MARCH 31, 2000 returns a value of FEBRUARY 29, 2000.
 - The default century for two-digit year values depends on the setting of YYCENTURYRANGE.
 - Treat with caution values returned for dates prior to the adoption of the Gregorian calendar in 1582 or for dates in the very far future that could be subject to calendar adjustments (for example, it is not yet clear if the year 4000 is a leap year). \$ADD_DATE accurately returns values for dates 200 years before or after the present date.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for information on YYCENTURYRANGE in z/OS. See *TIBCO Object Service Broker Parameters* for information on YYCENTURYRANGE in Open Systems.

Exceptions

CONVERSION	Signaled if <i>date</i> is not a valid date or if the value given for <i>amount</i> is not a number.
OVERFLOW	Raised if the resulting date cannot be expressed as a semantic type date, binary 4.
RANGERROR	Signaled if the value given for <i>component</i> is not one of D, W, M, or Y.

Example The following rule returns the employee names and the dates when they reach their five-year anniversaries, and displays the result in the message log:

```
ADD_DATE;
_ LOCAL ANNIV_DATE;
_ -----
_                                     +-----+
_ CALL MSGLOG('THE ANNIVERSARY DATES FOR THE EMPLOYEES ARE') | 1
_ ;                                                         |
_ FORALL EMPLOYEE_DATE WHERE HIREDATE >= $CREATE_DATE(      | 2
_   'YYYY/MM/DD', '1997/04/01') :
_   ANNIV_DATE = $ADD_DATE(EMPLOYEE_DATE.HIREDATE, 'Y', 5);
_   CALL MSGLOG(PAD(EMPLOYEE_DATE.LNAME, 22, ' ', 'L')
_     || ANNIV_DATE);
_ END;
_ -----
```

Pressing PF2 after executing the rule displays the following screen:

```
----- INFORMATION LOG -----
COMMAND ==>>>                                SCROLL: P

THE ANNIVERSARY DATES FOR THE EMPLOYEES ARE
SMYTHE                2003-09-18
CHANG                 2003-04-05
GARZA                 2002-06-30
TOWNSEND              2002-05-13
.
```

ADMIN_RIGHTS

Obtains, releases, or transfers the promotion rights on objects. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And ...
Administrator's workbench	PA Promotion Rights Admin option	Press Enter
Developer's workbench	EX Execute Rule option	Type ADMIN_RIGHTS <Enter>
	COMMAND prompt	Type EX ADMIN_RIGHTS <Enter>

See Also *TIBCO Object Service Broker Managing Deployment* for information about using ADMIN_RIGHTS.

ALLOCDSN

Allocates a file to a z/OS DDNAME. (C)

Invocation `CALL ALLOCDSN(ddname, dsname)`

<i>ddname</i>	A character string specifying the DDNAME. Its syntax is C (fixed-length character string) with length 8.
<i>dsname</i>	A character string specifying the filename. Its syntax is C with length 44.

- Usage Notes**
- If *ddname* is already in use, the old file is freed before the new file is allocated. Otherwise, a data set allocated by ALLOCDSN is not freed automatically by TIBCO Object Service Broker.
 - The file referred to must be a z/OS sequential file. A member of a partitioned data set, such as PRODUCT.MAIN.SOURCE(DOB), is not allowed.
 - ALLOCDSN is used only with the z/OS version of TIBCO Object Service Broker.
 - ALLOCDSN is not meant to be used with the @CLOSEDSN, @OPENDSN, @READDSN, and @WRITEDSN tools.

Exceptions

ROUTINEFAIL	Raised if the allocation cannot be successfully completed.
--------------------	--

Example The following rule allocates the example file to a z/OS DDNAME:

```
ALLOCDSN_1;
-----
- -----+-----
- CALL ALLOCDSN('EXAMPLES', TSOID || ' .EXAMPLES.DATA');      | 1
- -----
```

@ARCH_ACCESSLOGI

Enables analysis of archived audit log data.

The following technique may be used to analyze archived data on z/OS or Open Systems.

Two tables are provided:

1. @ARCH_ACCESSLOGI is an IMP table.

As supplied, it uses DD name ARCHIVE rather than a file name. If required, a level-7 user should be used to modify the definition of this table to use either a file name or DD name of your choice. The file or DD name should be specified according to instructions in "TIBCO Object Service Broker Managing External Data".

2. @ARCH_ACCESSLOG is a subview of the IMP table.

Browse this table to review the archived data.

For example:

- TSO ALLOC F(ARCHIVE) DA('S6B.ARCHIVE.DATA(D111007)') SHR REUSE
- Log on as level-7 user
- BR @ARCH_ACCESSLOG

BROWSING TABLE : @ARCH_ACCESSLOG

COMMAND ==> SEL TIME > '06:08:39' & TIME < '06:08:42'

IDKEY	DATE	TIME	MSG
-------	------	------	-----

4	111007	06:08:40	Insert access to TABLE @OBJECTSETS(SYSADMIN) by SYSADMIN
5	111007	06:08:40	Insert access to TABLE @COMPONENTS(TESTCXEC) by SYSADMIN

AUDITLOG

Invokes the Query Audit Log tool. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type AUDITLOG <Enter>
	COMMAND prompt	Type EX AUDITLOG <Enter>
From a rule		Type CALL AUDITLOG

Prerequisites The @AUDITLOG object set must be enabled before you can use this utility.

Usage Notes This tool enables users to do the following:

- Query the audit log using a default filter or a user-defined filter
- Create a new filter and update the existing user-defined filter
- Initiate a reporting session

See Also *TIBCO Object Service Broker Managing Security* for information on the audit log.

BATCH

Submits a batch job to a particular queue, views the status of the batch jobs, and views the queues that are available. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And ...
Developer's workbench	EX Execute Rule option	Type BATCH <Enter>
	COMMAND prompt	Type EX BATCH <Enter>

Either method displays the screen shown in [Batch Submission Main Menu Illustrated](#) below.

- Usage Notes**
- `$BATCHOPT` is the version of this tool called from within a rule.
 - If you want to submit a batch job to a queue as part of a rule or application, use the SCHEDULE statement with the TO clause.
 - The queue must be defined first before you can submit a job to it.

See Also *TIBCO Object Service Broker Programming in Rules* for information about the SCHEDULE statement. See *TIBCO Object Service Broker for z/OS Installing and Operating* for information about defining queues.

Batch Submission Main Menu Illustrated

The Batch Submission Main menu screen shown here appears when you execute BATCH:

Batch Submission Facility	
Submit Batch Request	_
Build Batch Commands	_
View Batch Status	_
Queue Definition	_

PFKEYS: 2=LOGS 3=EXIT 12=EXIT

Main Menu Options

From the main menu, you can choose to:

- Submit a batch request to a specific queue
- Update existing batch commands or create new batch commands for the batch job
- View all the requests you submitted
- Look at the definitions of available queues

To select an option from the menu, place the cursor next to the option and press Enter.

Submit Batch Request Option

If you choose the Submit Batch Request option, the Submit Batch Request Screen appears.

Submit Batch Request Screen

Command ==>

Batch Submission Facility

Rule Settings:

Queue Name	==>	Requestor ID	==> USR40
Rule Name	==>	Batch Command Name	==> *DEFAULT*
Search Path	==> L (S/I/L)	Security Group	==> DEVELOPERS
Local Library	==> USR40	Browse Mode	==> N (Y/N)
Parameter Values	==>		

Output Settings:

Print Destination	==>	Print Form	==>
Print Class	==>	External Writer	==>
Number of Copies	==>	FCB	==>
		UCS	==>

PFKEYS: 1=HELP 3=SAVE 12=EXIT 6=SUBMIT 4=PREV REQUEST 2=OPTIONS

The screen is divided into two parts:

- Rule Settings

Most of these fields are filled in for you with default values. You can change these values, and you must provide values for the **Queue Name** and **Rule Name** fields.
- Output Settings

Optionally, you can fill in these fields.

Rule Settings

The following fields describe the attributes for the rule that you want to execute in batch (refer to the Rule Settings in [Batch Submission Main Menu Illustrated on page 41](#)).

Queue Name

You must provide the name of the queue that executes your TIBCO Object Service Broker rule in batch mode. The queue name is validated to ensure that a queue with that name exists. To view valid queue names, place the cursor in this field and press PF2. A screen similar to the following appears:

Valid Queue Names Screen

LIST OF VALID QUEUE NAMES	
COMMAND ==>	SCROLL: P
NAME	DESCRIPTION
-----	-----
_ AFTERNOON	Job executed between 12 noon and 5 p.m.
_ IMMEDIATE	Job executed immediately
_ SUNDAY	Job executed on Sunday
_ OVERNIGHT	Job executed between 8 p.m and 4 a.m.
< Place "S" beside the items you wish to have Selected on PF3 >	
PFKEYS: 1=HELP 3=SELECT 13=PRINT 12=EXIT	

You can select a queue from the screen by typing an S next to the queue name and pressing PF3.

To see the definition of a queue, use the fourth option (Queue Definition) on the main menu shown in [Batch Submission Main Menu Illustrated on page 41](#).

Rule Name	Provide the name of the rule to be executed in batch mode. The rule name is not validated.
Search Path	Enter the order of libraries to search for the rule: S - System library only; I - Installation library, then system library; L - Local library, installation library, then system library.
Local Library	Enter the name of the local library to access as the local library when the requested rule is executed.

Parameter Values	If the requested rule has arguments, enter the values here, separated by commas. Single quotation marks are not necessary.
Requestor Id	Currently, you can submit only your own requests for batch jobs; therefore, this field defaults to your user ID and you cannot change it.
Batch Command Name	Enter the name of the batch command that you created to run your batch request. You can create or modify batch commands with the Build Batch Commands option on the main menu shown in Batch Submission Main Menu Illustrated on page 41 . To view a list of all the batch commands you created, place the cursor on the BATCH COMMAND NAME field and press PF2 to display the LIST OF JCL NAMES screen:

List of JCL Names Screen

LIST OF JCL NAMES	
COMMAND ==>	SCROLL: P
_ *DEFAULT*	
_ FML_EXPORT	
_ MY_JCL	
_ RULE1	
< Place "S" beside the item(s) you wish to have Selected on PF3 >	
PFKEYS: 1=HELP 3=SELECT 13=PRINT 12=EXIT	

You can select a batch command by typing an **S** next to the batch command name and pressing Enter.

Security Group	Specify the security group that is in effect when the rule is run in batch mode. You can specify any of your valid groups.
Browse Mode	Specify whether the rule runs in browse mode or update mode. Y indicates browse mode and N indicates update mode.

Output Settings

The following fields describe the print file options (Output Settings in [Batch Submission Main Menu Illustrated on page 41](#)) for the output generated by the rule run in batch. All these print options are optional. If you do not specify anything for the output settings, the batch server uses the default options specified in the queue definition.

Print Destination	Enter the symbolic name of the printer. If you want to include the node name of the printer, specify the print destination in the form: <ul style="list-style-type: none">• <i>node</i> - The node name• <i>char</i> - A non-alphanumeric character that separates the node name from the printer name• <i>printer</i> - The printer name
Print Class	Enter the output class.
Number Of Copies	You can specify a number of copies of output from 1 to 255.
Print Form	If the output data should be printed or punched on a special output form, enter Y. This flag takes effect in z/OS only.
External Writer	If you want to direct system output to an unsupported device, you can specify an external routine to direct the output. You must leave this field blank if you specify a node name in the Print Destination field.
FCB	Enter the FCB (Forms Control Block) value for the type of output format.

UCS	Enter the UCS (Universal Character Set) value that should be used for printing the output data set.
-----	---

Instead of entering information in all the fields for a batch job request, you can recall the specifications for a previous request by pressing PF4. A screen similar to the following appears:

Previous Requests Screen

List of Previous Requests			USERID: USR40		
COMMAND ==>			Scroll P		
RULE	QUEUE	SEARCH_PATH	LOCAL_LIB	REQUESTOR_ID	JCL_*
-----	-----	-----	-----	-----	-----
_ AFDASF	IMMEDIATE	L	USR40	USR40	PRODU
_ BACKUP	SUNDAY	L	USR40	USR40	*DEFA
_ RESTORE	OVERNIGHT	S	USR40	USR40	*DEFA
D-Delete S-Select					
PFKEYS: 12=EXIT 13=PRINT 3=END 5=FIND NEXT 9=RECALL					

Use PF11 to scroll to the right to view more of the attributes for previous requests. To select a request, use the **S** line command; to delete a request, use the **D** line command.

After displaying the selected request on your Submit Batch Request screen, you can modify it to suit your present needs. After entering the information, use one of the following PF keys:

- PF3 - Save the request.
- PF6 - Submit the batch request.
- PF12 - Exit the batch request screen without saving the request.

Build Batch Commands Option

If you choose the Build Batch Commands option from the Batch Submission Main menu, you see a screen similar to the following:

Build Batch Command Screen

```
COMMAND ==>
                Batch Submission Facility
        Batch Command list for Userid:  USR40

    < Create New Batch Command >  _____

        *DEFAULT*      _
            MY_JCL      _
            BATCHEXP     _

PFKEYS:  1=HELP  3=EXIT 12=EXIT ENTER=SELECT
```

On this screen, you can do one of the following:

- You can select one of the batch command names that you previously defined by placing the cursor next to the name and pressing Enter. Each person has a private copy of the *DEFAULT* batch command and these can be individually customized.
- You can create new batch commands by typing a new name in the **Create New Batch Command** field and pressing Enter to display a screen similar to the following:

New Batch Command Screen

```
                Batch Submission Facility

Batch Command Name  : BATCHIMP
Command ==>

                Batch Command
```

 -

PFKEYS: 12=QUIT 22=DELETE 3=SAVE 9=REPEAT CMND 1=HELP

In this environment, you can enter batch commands and edit them with the TED editing facility (refer to [TED](#)). When you build your batch command, you can use the following symbols, which are filled in for you when the job is submitted:

- {USERID} - This is replaced with the requestor user ID as specified in the Requestor Id field of the Submit Batch Request Screen (refer to [Submit Batch Request Option on page 43](#)).
- {QUEUE} - This is replaced with the queue name.
- {TDS} - This is replaced with the communications identifier (for example, VTAM name) that identifies the Data Object Broker that is connected to the batch server.
- {CLASS} - This is replaced with the print class specified for the batch queue.

All the TED commands are available. A particularly helpful command for copying existing batch commands is the **COPY** command. Specify the text table that is your source of data and the data is copied to the table that you are currently editing with TED. For example, to copy the default batch command (named *DEFAULT*) of user USR50, type the following on the command line:

```
COPY @BATCH_JCL(USR50, '*DEFAULT*')
```

@BATCH_JCL is a text table that requires a user ID and the name of the batch command as parameters. For more information about @BATCH_JCL refer to the *TIBCO Object Service Broker for z/OS Installing and Operating* manual.

The copied batch command can look similar to the batch command shown here:

Copied Batch Command Screen

Batch Submission Facility

Batch Command Name : BATCHIMP
Command ==>

```
Batch Command
-----
{USERID}A JOB ('{USERID}'),
      'JOB CARD',
      NOTIFY={USERID},
      MSGCLASS={CLASS},
      REGION=4096K,
      MSGLEVEL=(1,1),
      TIME=10
```

PFKEYS: 12=QUIT 22=DELETE 3=SAVE 9=REPEAT CMND 1=HELP

You can add more steps to the batch command provided in the default. Do not, however, add a step to execute the requested rule because this step is appended to the end of the batch command when the batch request is submitted.

View Batch Status Option

If you choose the View Batch Status option from the Batch Submission Main menu, you see a screen similar to the following:

View Batch Status Screen

COMMAND ==>

Batch Submission Facility				
Job Status for Userid: AKS40				
Rule Name	Queue Name	Status	Submit	
			Date	Time
AFDASF	IMMEDIATE	DONE	01/14/98	10:35:03
BACKUP	SUNDAY	PENDING		
RESTORE	OVERNIGHT	SUBMITTED		

D - Delete
 PFKEYS: 1=HELP ENTER=REFRESH 3=EXIT 12=EXIT

This screen shows all the requests that you submitted. The screen is refreshed every time you press Enter. The following information is provided:

Rule Name	The name of the rule that you submitted for execution in batch
Queue Name	The name of the queue to which you submitted your batch job
Status	<p>The present state of a batch job. The status can be:</p> <ul style="list-style-type: none"> • SUBMITTED - The request is submitted to the operating system. • PENDING - The request is waiting in the queue. • EXECUTING - The request is currently being executed in batch mode. • DONE - The execution of the request is finished. <p>FAILED - The execution of the request failed. You can use the D line command to remove requests from the queue, and if the status is SUBMITTED or EXECUTING, you must remove the job from the operating system.</p> <p>NOTE: View Batch Status cannot replace your normal methods of monitoring batch jobs because TIBCO Object Service Broker has no knowledge of jobs that fail due to batch command errors or cancellation by an operator.</p>
Submit Date	The date that you submitted your request to the queue, not the date when the actual job was submitted to the operating system
Submit Time	The time that you submitted your request to the queue, not the time when the actual job was submitted to the operating system

Queue Definition Option

If you choose the Queue Definition option from the Batch Submission Main menu, you see a screen similar to the following one. For more information about defining new queues, refer to the *TIBCO Object Service Broker for z/OS Installing and Operating* manual.

Queue Definition

COMMAND ==>>

Batch Submission Facility
Queue Definitions

< Define New Queue > _____

AFTERNOON _
IMMEDIATE _
OVERNIGHT _
SUNDAY _

PFKEYS: 1=HELP 3=EXIT 12=EXIT

The example screen shows four existing queues: AFTERNOON, IMMEDIATE, OVERNIGHT, and SUNDAY. To view the definition of one of these queues, position your cursor on the field beside the name of that queue and press Enter. The definition of the queue appears as shown here:

Queue Definition Screen

COMMAND ==>>

Batch Submission Facility
Queue Definitions for: SUNDAY

Wait Duration ==> 15 (# of seconds to wait when queue empty)
Wait Limit ==> 12 (# of waits before queue shut down)

Default Output Settings:
Print Destination ==> PRINTER2 Print Form ==>
Print Class ==> Y External Writer ==>

Number of Copies ==> 1

FCB
UCS

==>
==>

Description:

PFKEYS: 1=HELP 3=SAVE 12=CANCEL 22=DELETE

The fields on this screen are defined as follows:

Wait Duration	The time in seconds that the batch server waits when the queue becomes empty
Wait Limit	The number of times the batch server goes into a wait state before shutting itself down
Print Destination	The name of the printer where you want the batch job output sent. If you want to include the node name of the printer, specify the print destination in the form: node- The node name; char - A non-alphanumeric character that separates the node name from the printer name; printer - The printer name
Print Class	The class of the output
Number Of Copies	The number of output copies desired
Print Form	Whether the output data should be printed or punched on a special output form
External Writer	Specifies the external routine that directs the output, if the system output is to be directed to an unsupported device
FCB	(Forms Control Block) Specifies the type of output format
UCS	(Universal Character Set) Describes the character set that should be used for printing the output data set
Description	A brief description explaining the purpose of this queue



The default print option values defined on the Queue Definition screen are used when the user submitting a batch request does not supply those values on the Submit Batch Request screen or through a call to \$BATCHOPT.

BATCH_ENABLE

Enables all the object sets previously processed using [@MAKEMEMBERS](#). (C)

Invocation `CALL BATCH_ENABLE(wipe_existing)`

<i>wipe_existing</i>	This argument saves or deletes permissions for existing members. Valid values are Y (delete existing memberships) or N (save existing memberships).
----------------------	---

Prerequisites All users must be suspended from the system while BATCH_ENABLE is running.

Usage Notes Run this tool after the members of an object list have been set up using [@MAKEMEMBERS](#). Refer to [@MAKEMEMBERS](#) for further information about this associated tool.

Example An installation has a large number of enabled object sets used by a large user community. Updating member lists of an object set by changing the Enable Lists of that object set involves a lot of processing and can be subject to lock failure due to ongoing use of the objects by users. To make such updates easier, it is preferable to suspend the system and run a batch job. New enable lists for object sets are created ahead of time (using [@MAKEMEMBERS](#)), so as not to affect current use of objects within these object sets. The system is suspended during off-hours and the job is scheduled:

```
SCHEDULE BATCH_ENABLE("Y");
```

After this is done, only those members on the enable list specified via [@MAKEMEMBERS](#) have the object set enabled for them. Users previously on the enable list who are not specified in [@MAKEMEMBERS](#) are no longer on the enable list and therefore no longer are able to access the objects through that object set.

BATCHLOAD_CARDS

Defines input and output to the Batch Load utilities (S6BBRTBL/hrnbrtbl). (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type BATCHLOAD_CARDS <Enter>
	COMMAND prompt	Type EX BATCHLOAD_CARDS <Enter>

Preparation of File and Table Definitions

Before using BATCHLOAD_CARDS, complete the following steps:

- Depending on your platform:
 - On z/OS systems, allocate a file to hold the control cards. This fixed block file must have a record length of 80 bytes.
 - On non-z/OS systems, set the DSBIFTYPE Execution Environment parameter to TEXT.



This setting could affect the behavior of other tools.

- Define the TDS table that is to receive the data.

If the table is non-parameterized, it must be empty. If the table is parameterized, the table instance you plan to load must not already exist.
- [Optional] Define a table that reflects the format and layout of the input file to be loaded.

This step is required if you do not have a table that reflects the layout of the input file.

BATCHLOAD_CARDS references the table definition to establish the format and layout of the input file. If the definition of the table you are going to load closely resembles the layout of the input file, BATCHLOAD_CARDS references this table definition instead.

See the *TIBCO Object Service Broker Parameters* manual for information on the DSBIFTYPE Execution Environment parameter.

Defining Control Cards with BATCHLOAD_CARDS

To define your control cards with BATCHLOAD_CARDS, you describe your input and output data in a series of screens. BATCHLOAD_CARDS then writes the necessary control cards in a file. Complete the following tasks:

- 1. [Execute BATCHLOAD_CARDS from a TIBCO Object Service Broker workbench](#)
- 2. [Supply an input file definition](#)
- 3. [Match input file parameters and fields with data table parameters and fields](#)
- 4. [Specify constant values or ignore fields](#)
- 5. [View the fields with secondary indexes](#)

These tasks are detailed in the following sections.



Consider the effect of the following procedures on your continuous backup.

Task A Execute BATCHLOAD_CARDS from a TIBCO Object Service Broker workbench

Execute BATCHLOAD_CARDS from the workbench as follows:

```
EX execute rule ==> BATCHLOAD_CARDS
```

The Control Card Definition screen appears.

Control Card Definition Screen

Control Card Definition for BATCH LOAD			
Control Card File	:	USR01.BATCH.CNTLCARD(FEB2000)	
Table	:	EMP_MSTR_TDS	Character Set : ENGL
PAGE FILL LEVELS	:	(Default = 75 % if blank)	
		Group Index :	Primary Index :
		Secondary Index :	Data Pages :
# of Input Records	:	(Default = 100000 if blank)	
Total # of Fields	:	(Default = 50 if blank)	
Dynamic Unit Name	:	(Default = SYSDA if blank)	
Dynamic Block Size	:	(Default = 4096 if blank)	

PFKEYS: 1=HELP 3=EXIT 4=CONTINUE 12=CANCEL

Fields You must supply values for the following fields:

Control Card File	The file (and member name if the file is a z/OS partitioned data set) to contain the BATCHLOAD_CARDS output. You can change the filename by typing over it.
Table	The name of the table that is to be loaded by S6BBRTBL or hrnbrtbl.
Character Set	The character set ID for the language of the data. If the data being loaded is in a language other than the one shown, specify the appropriate character set in this field. Values are listed in Character Set Values below.

Character Set Values

Supported character set values are listed here.

Char Set ID	Language	Territory/Country
CDNB	Canadian Bilingual	Canada
DANS	Danish	Denmark
DEUT	German	Austria, Germany
ENGB	English	Great Britain
ENGL	English	USA
ESPA	Spanish	Latin America, Spain
FRAN	French	France
ITAL	Italian	Italy
NORS	Norwegian	Norway
PORT	Portuguese	Portugal

Char Set ID	Language	Territory/Country
SCHW	French, German	Switzerland
SUOM	Finnish	Finland
SVEN	Swedish	Sweden

Specify Control Card Field Values

The lower portion of the Control Card Definition screen specifies the values to be used to create the S (specification) type control card. To override the defaults, type a new value in the space provided.

PAGE FILL LEVELS	The percentage of usable page space that is filled during the load process. For more information about adjusting Page fill levels for TDS tables, refer to <i>TIBCO Object Service Broker for z/OS Utilities</i> .
# of Input Records	The number of records in the input file.
Dynamic Block Size	The block size used for table instances and for temporary work files required to process secondary indexes. The small default block size can be detrimental to good performance and could require optimizing if secondary indexes are to be built for large tables, that is, tables with large numbers of rows.

Use PF4 to move to the next screen.

Task B Supply an input file definition

Input File Definition Screen

To define your input file, use the Input File Definition screen as illustrated here:

Control Card Definition for BATCH LOAD

INPUT FILE DEFINITION:

Was input data unloaded from an archive: N
 Input Data File Length Format (Fixed of Variable): F

Define the input file by either specifying the name of the table that

matches the input file layout or specifying the fields explicitly in the space provided.

Table: EMP_MSTR_TDS

	Name	Syntax	Length	Decimal	Offset(optional)
	-----	----	----	--	----
—	PK	C	9	0	
—	F1	C	15	0	
—	F2	C	15	0	
—	F3	C	25	0	
—	F4	C	25	0	
—	F5	P	6	2	
—	F6	C	8	0	
—	F7	B	2	0	

PFKEYS: 1=HELP 3=EXIT 4=CONTINUE 12=CANCEL

Specify Whether the File Was Loaded From an Archive

To specify whether the input file was created using S6BBRULA or hrnbrula, type Y or N at the prompt:

Was input data unloaded from an archive: N

If you type Y, you must specify V (variable) as the file length format in the Input Data File Length Format field that follows.

Specify File Length

To specify whether the input file is fixed or variable length, type F or V at the prompt:

Input Data File Length Format (Fixed or Variable): F

The name of the table you specified in the Table field on the previous screen appears in the Table field by default, because the table you are loading could resemble the layout of the input file.

Specify File Format

To define the format of the input file, do one of the following on this screen:

Specify a Table whose Definition Reflects the Input File Layout

- 1. Keep the default Table value or change it to a table that reflects the format of the input file. If you defined a table for your input file, enter the name of that table in this field.

- 2. To see the fields in the Table, press Enter. All the field definitions for that table appear on the screen.
- 3. To edit the field definitions, remove the table name from the Table field, and press Enter again.

Specify the Layout of the Input File Explicitly

- 1. Delete the Table value.
- 2. Type the names and definitions of the fields in the input file.
- 3. Press Enter.

Input File Fields

The lower portion of the Input File Definition Screen displays field definitions for the input file named on the Table Field. If, for example, you define a table named INPUT_FILE and change the Table value in the Input File Definition Screen to INPUT_FILE, the screen displays the fields of the INPUT_FILE table, as shown here:

Control Card Definition for BATCH LOAD				
INPUT FILE DEFINITION:				
Was input data unloaded from an archive: N				
Input Data File Length Format (Fixed of Variable): F				
Define the input file by either specifying the name of the table that matches the input file layout or specifying the fields explicitly in the space provided.				
Table:	INPUT_FILE			
Name	Syntax	Length	Decimal	Offset(optional)
-----	----	----	--	----
_ SOCIAL_SECURITY	C	9	0	
_ LAST_NAME	V	15	0	
_ FIRST_NAME	V	15	0	
_ SALARY	P	6	2	
_ ADDRESS_LINE1	V	25	0	
_ ADDRESS_LINE2	UN	25	0	
_ AGE	N	2	0	
_ SEX	C	1	0	
_ DEPARTMENT_#	RD	5	0	
PFKEYS: 1=HELP 3=EXIT 4=CONTINUE 12=CANCEL				

Editing Field Definitions

To edit the field definitions displayed, clear the INPUT_FILE name from the Table field and press Enter. Now you can make changes to the other fields on the screen:

Name	The input file field name (maximum of 35 characters).
Syntax	The syntax of the input field.
Length	The maximum field length of the input field.
Decimal	The number of post decimal positions for a numeric field. If omitted, the decimal number defaults to 0 and appears on the control cards as blank. The number of decimals is required to accurately convert the value to TIBCO Object Service Broker syntax.
Offset	The location of the field within the input record. The offset is optional. If specified, it must start at 0 for position 1.

After defining all the input fields, press PF4 to move to the next screen.

Task C Match input file parameters and fields with data table parameters and fields

Field Matching Screen

To match an input field with a receiving field, type the number of the input field in the blank area before the receiving field name. For example, in the screen that follows, SOCIAL_SECURITY (field 1) maps to PK, LAST NAME maps to F1, and so on:

Control Card Definition BATCH LOAD	
Input File: INPUT_FILE	Table: EMP_MSTR_TDS
1 SOCIAL_SECURITY	___ PRM1
2 LAST_NAME	9_ PRM2
3 FIRST_NAME	1_ PK
4 SALARY	2_ F1
5 ADDRESS_LINE1	3_ F2
6 ADDRESS_LINE2	5_ F3
7 AGE	6_ F4
8 SEX	4_ F5
9 DEPARTMENT_#	___ F6
	___ F7

PFKEYS: 1=HELP 3=EXIT 4=CONTINUE 12=CANCEL

Screen Layout

The names of the input file/import table and the TDS table appear at the top of the screen. If an input file is described by a table, the table name appears in the Input File field; otherwise, it is left blank. The Table is the name of the TDS table that is loaded by S6BBRTBL or hrnbrtbl (the Batch Load utilities).

Input fields not mapped are not loaded into the receiving table.

Task D Specify constant values or ignore fields

Procedure

If some fields are not mapped to input fields, you can load a constant value into each occurrence of a field or ignore fields of the table. If you want to specify a value for the field displayed, provide one in the VALUE field (to a maximum of 250 bytes). To ignore the displayed field, leave the VALUE field blank.

Sample Screen

Fields that are not mapped to an input file field appear on the screen below.

In the following example, a common city and FAX number are added to each employee record loaded into the TDS table. Field F7 is ignored. If more fields than the ones displayed have not been mapped, use PF8 to scroll down.

Control Card Definition BATCH LOAD

The table fields that have not been matched to the input file fields appear below. To specify a value for the field, type the value in the given area next to the field.

Table: EMP_MSTR_TDS

FIELD	VALUE
-----	-----
PRM1	TORONTO

555-1234

F7

PFKEYS: 1=HELP 3=SAVE 4=CONTINUE 12=CANCEL



- Primary key fields must be either cross-referenced to an input field or system generated (that is, an IDgen key). They must not appear on value cards.
- Fields of syntax RD (raw data) and UN (Unicode) must not appear on value cards.

Press PF4 to continue.

Task E View the fields with secondary indexes

The following screen displays a list of TDS fields with the currently assigned primary and secondary keys. After viewing the secondary index fields, press PF4 to complete the control cards definition and write them in your file.

Secondary Index View Screen

Control Card Definition BATCH LOAD

These are the fields and their key types for the selected table.

Table: EMP_MSTR_TDS

Primary	Secondary	Field	Name
Y	\bar{Y}	PK	
		F1	
		F2	
		F3	
		F4	
		F5	
		F6	

PFKEYS: 1=HELP 3=EXIT 4=CREATE CTL CARDS 12=CANCEL

An example of an output file containing the control cards is shown here:

An example of an output file containing the control cards is shown here:

Code	Entity	Field	Value	Code	Entity	Field	Value
S	75	75	000000000100000	050	SYSDA		
I	001	F	SOCIAL_SECURITY			C 009	PK
I	002	F	LAST_NAME			V 015	F1
I	003	F	FIRST_NAME			V 015	F2
I	004	F	SALARY			P 006 02	F5
I	005	F	ADDRESS_LINE_1			V 025	F3
I	006	F	ADDRESS_LINE_2			V 025	F4
I	007	F	AGE			B 002	
I	008	F	SEX			C 001	
I	009	F	DEPARTMENT_#			C 004	PRM2
H	001	R	ENGL				EMP_MSTR_TDS
H	002	P	PRM1			S C 015	
H	003	P	PRM2			S C 004	
H	004	F	PK			I C 009	P
H	005	F	F1			S C 015	S
H	006	F	F2			S C 015	
H	007	F	F3			S C 025	
H	008	F	F4			S C 025	
H	009	F	F5			Q P 006 02	
H	010	F	F6			S C 008	
H	011	F	F7			C B 002	
V	PRM1		TORONTO				
V	F6		555-1234				



Notice the P and S designations for the PK and F1 fields. These letters mean that PK is a primary key and F1 is a secondary index. If a field is both a primary key and a secondary index, the designation is Q.

BATCHLOAD_CARDS Record Layout

The following tables relate control fields on the screens with their equivalent fields in the control records:

Table Attributes

Table Attribute	Screen	Record ID/Type	Columns
Table Name	1	H/R	64 – 79
Character Set	1	H/R	9 – 12
Page Fill Levels:			
Group Index	1	S	3 – 4
Primary Index	1	S	6 – 7
Secondary Index	1	S	9 – 10
Data Pages	1	S	12 – 13
# of Input Records	1	S	15 – 29
Total # of Fields	1	S	31 – 33
Dynamic Unit Name	1	S	35 – 42
Dynamic Block Size	1	S	44 – 48
# Volumes for work file	n/a	S	50 – 51

Input File Definitions

Input File Definition	Screen	Record ID/Type	Columns
Name	n/a	I/F	9 – 44
Syntax ¹	n/a	I/F	45 – 48
Length	n/a	I/F	49 – 52
Decimal	n/a	I/F	54 – 55
Offset	n/a	I/F	60 – 62
Variable Length Indicator	2	I/R	46
Cross Reference	2	I/F	64 – 79

1. Syntax is left-justified if it is a raw data or Unicode syntax; otherwise, it is right-justified.

Field Definitions

Field Definition	Screen	Record ID/Type	Columns
Name	n/a	H/F or P	9 – 44
For fields other than raw data (RD) or Unicode (UN):			
Semantic Type	n/a	H/F or P	46
Syntax	n/a	H/F or P	48
For fields of syntax RD or UN:			
Syntax	n/a	H/F or P	45 – 48
Continue for all fields:			
Length	n/a	H/F or P	49 – 52
Decimal	n/a	H/F or P	54 – 55
Key Type	n/a	H/F	57
Null-Equivalent Value	n/a	H/F	64 – 79
Static Values	4		
Target table field	4	V	3 – 18
Continuation sequence #	4	V	20
Table field static value	4	V	22 – 71



I and H records contain a sequence number in columns 3 – 5, which must be consecutive and must start with 001. The types within these records must be in the order: R, if any, followed by P, if any, followed by F.

\$BATCHOPT

Sets the batch options associated with a SCHEDULE TO statement's batch request, which sends the batch job to a queue. (C)

Invocation `CALL $BATCHOPT(batch_option,option_value)`

<i>batch_option</i>	The name of the option you want to set. It can be one of the following:
search_path	Where to start searching for the rule. Valid values: S – system library, I – installation library, L – local library
local_lib	The name of the library to access as the local library when the requested rule is executed.
jcl_name	The name of the JCL that you created to run your batch request.
security_group	The security group that is in effect when the rule is run in batch mode. You can specify any of your valid groups.
print_class	The output class.
print_dest	<p>The symbolic name of the printer. If you want to include the node name of the printer, specify the print destination in the form:</p> <p>node char printer</p> <ul style="list-style-type: none"> • node is the name of the node. • char is a non-alphanumeric character that separates the node name from the printer name. • printer is the name of the printer.
print_copies	The number of copies (1 to 255) to output.
print_form	The form name, if the output data should be printed on a special output form.

print_xwtr	An external routine to direct the output, if you want to direct system output to an unsupported device (z/OS and Solaris only). Leave this field blank if you specify a node name in the PRINT_DESTINATION field.
print_fcb	The FCB (Forms Control Block) to use, if you want to use an output format other than the default
print_ucs	The UCS (Universal Character Set) to use, if you want to use a character set other than the default
option_value	Specify the value that you want for the selected option.

Usage Notes

- **BATCH** is the interactive version of this tool used on the developer workbench.
- Use \$BATCHOPT before using a SCHEDULE TO statement (refer to *TIBCO Object Service Broker Programming in Rules* for a description of the SCHEDULE statement with the TO clause). Because \$BATCHOPT applies only in conjunction with scheduling to a queue, it does not affect a SCHEDULE statement without a TO clause.
- When an option is set, it remains the same throughout the session until it is reset again.
- If you do not set batch options with \$BATCHOPT, the default for JCL_NAME is *DEFAULT*, and the values for the rest of the options are determined by the default values for the current session.

Example

The rule shown here sets the local library and the JCL name before scheduling the rule, BACKUP, in browse mode to a queue called OVERNIGHT.

SCHED_RULE ;	

CALL \$BATCHOPT('LOCAL_LIB', 'TEST') ;	1
CALL \$BATCHOPT('JCL_NAME', 'MY_JCL') ;	2
SCHEDULE IN BROWSE TO 'OVERNIGHT' BACKUP ;	3

When the rule called BACKUP is executed in batch mode, the JCL that is used is MY_JCL, and the local library TEST runs in browse mode.

BATCHUNLD_CARDS

Defines the control cards required by the Batch Unload utilities. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type BATCHUNLD_CARDS (<i>unload_source</i>) <Enter>
	COMMAND prompt	Type EX BATCHUNLD_CARDS (<i>unload_source</i>) <Enter>

Where:

<i>unload_source</i>	One of the following:
	<ul style="list-style-type: none"> • DATABASE - If you are preparing control cards for the S6BBRULH/hrnbrulh or S6BBRULB/hrnbrulb utilities. • ARCHIVE - If you are preparing control cards for the S6BBRULA/hrnbrula utility.

Usage Notes BATCHUNLD_CARDS helps you define the control cards required by the Batch Unload utilities. There are three Batch Unload utilities:

- S6BBRULH/hrnbrulh (Batch Unload while TIBCO Object Service Broker is running and the segment is online)
- S6BBRULB/hrnbrulb (Batch Unload while TIBCO Object Service Broker is shut down or the segment is offline)
- S6BBRULA/hrnbrula (Table Unload From Archive)

BATCHUNLD_CARDS displays a screen for you to specify the table or tables to unload, and if the table is parameterized, you can choose table instances. The utility then defines control cards and selection criteria in two separate files.

If you are using FTP to transfer unloaded data between z/OS and Windows, Solaris, or UNIX, it is no longer mandatory to use the S6BBRFRU z/OS utility before the data can be used by TIBCO Object Service Broker. Refer to *TIBCO Object Service Broker for z/OS Utilities* for more information about S6BBRFRU.

See *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities* for information on the method used by the unload utilities to preserve null values in the unload file. The unload utilities use specific field values to represent nulls in the unload file.

Preparation **Allocate File**

On z/OS systems, before you can use BATCHUNLD_CARDS, you must allocate a file for the control cards, and if you are going to specify table instances, also allocate a file for the selection criteria. The file attributes are as follows:

- **CNTRL** - The control card file specifies the definition of the table to be unloaded. The CNTRL file should be defined as fixed block, record length 80.
- **SELECT** - [Optional] The selection criteria file contains criteria for selecting table instances from a parameterized table set. If a SELECT file is required, it should be defined as fixed block, record length 530.

Specify the File Type

On non-z/OS systems, set the DSBIFTYPE Execution Environment parameter to TEXT.



This setting can affect the behavior of other tools.

See *TIBCO Object Service Broker Parameters* for information on the DSBIFTYPE Execution Environment parameter.

Procedure to Invoke the Control Card Preparation Facility

To invoke the control card preparation facility, complete the following tasks:

1. [Run BATCHUNLD_CARDS from the workbench](#)
2. [Supply field values](#)
3. [Select table instances for a parameterized table](#)

These tasks are described here.

Task A Run BATCHUNLD_CARDS from the workbench

```
EX execute rule ==> BATCHUNLD_CARDS(unload_source)
```

where unload_source is one of the following:

- **DATABASE** if you intend to use S6BBRULH/hrnbrulh or S6BBRULB/hrnbrulb

- ARCHIVE if you intend to use S6BBRULA/hrnbrula

The screen displayed is illustrated with sample input as follows:

```
CONTROL CARD SPECIFICATION FOR BATCH UNLOAD FROM DATABASE
Control Card File
USR01.BATCH.CNTLCARD(JAN2000)

Selection Criteria File
USR01.BATCH.SELECT(JAN2000)

      Tables      Selection of table instances for parameterized table set
-----
_ EMPLOYEE      REGION = 'MIDWEST'

_ EMPLOYEE      REGION = 'SOUTHWEST'

_

PFKEYS: 1=HELP 3=SAVE 12=CANCEL
```

Task B Supply field values

Supply values for the following fields:

Control Card File	The file (and its member name if the file is a z/OS partitioned data set) to contain the BATCHUNLD_CARDS output. You can change the filename by typing over it.
Selection Criteria File	The file (and its member name if the file is a z/OS partitioned data set) to contain the selection criteria, if you are selecting table instances of a parameterized table. Otherwise, ensure that this field is blank.
Tables	The TDS table to be unloaded. You can specify only one table for S6BBRULH/hrnbrulh or S6BBRULB/hrnbrulb and up to 999 for S6BBRULA/hrnbrula.

Task C Select table instances for a parameterized table

If you want to select table instances of a parameterized table for *inclusion* in the unload file, specify selection criteria for each data parameter. The selection criteria are subject to the following constraints:

- You must make a selection for each data parameter and join the parameters with the AND (&) logical operator. For example, if the EMPLOYEE table had REGION and CITY data parameters, you must specify both REGION and CITY:

```
REGION = 'MIDWEST' & CITY = 'CHICAGO'
```

- You must use the same operator for each parameter specification in one selection. Operators you can use are:

=	Equal to
≠	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

For example, a correct use of operator for two parameters is:

```
REGION = 'MIDWEST' & CITY = 'CHICAGO'
```

- You can make more than one selection, but you can use only one of the following operators on a table:

>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

For example, valid selection criteria are:

```
REGION = 'MIDWEST' & CITY = 'CHICAGO'
```

```
REGION = 'SOUTHWEST' & CITY = 'DALLAS'
```

After specifying the table and any table instances to be unloaded, press PF3 to write the control cards and selection file to your files. The control cards definition is then complete.

Review the Output File

A sample output file containing the control cards is shown here:

H 000 T	EMPLOYEE		001	014
H 000 P	REGION	I C	016	000
H 000 F	EMPNO	I P	003	000
H 000 F	LNAME	S C	022	000
H 000 F	POSITION	S C	014	000
H 000 F	MGR#	I P	003	000
H 000 F	DEPTNO	I B	002	000
H 000 F	SALARY	Q P	004	002
H 000 F	HIREDATE	S C	009	000
H 000 F	ADDRESS	S V	038	000
H 000 F	CITY	S C	020	000
H 000 F	STATE_PROV	S C	004	000
H 000 F	ZP_CODE	S C	007	000
H 000 F	BIRTHDATE	S C	010	000
H 000 F	SEC_GROUP	I C	016	000
H 000 F	ACCESSTYPE	S C	001	000
H 000 R				

An example of a selection criteria file is shown here:

*****	0002
EMPLOYEE	I=MIDWEST
EMPLOYEE	I=SOUTHWEST

\$BEEP

Issues the specified number of beeps from the terminal. (C)

Invocation `CALL $BEEP(repetition)`

<i>repetition</i>	A positive number that indicates to the terminal how many short beeps to issue.
-------------------	---

- Usage Notes**
- This command is available across all platforms.
 - In an IMS TM environment, only one beep is issued, regardless of the number of beeps specified for *repetition*.

Example This rule, which uses an employee’s last name as input, gets an employee name and then calls a second rule for further processing. If the employee does not exist, the terminal emits two beeps and displays the end message.

<pre>RULE EDITOR ==> GET_EMPLOYEE(LAST_NAME); - - ----- - GET EMPLOYEES WHERE EMPLOYEE = LAST_NAME; - CALL PROCESS_EMPLOYEE; - ----- - ON GETFAIL : - CALL \$BEEP(2); - CALL ENDMSG('EMPLOYEE NOT FOUND');</pre>	<pre> SCROLL: P +-----+ 1 2 +-----+</pre>
---	---

\$BLANKPAGE

Outputs a blank page. (C)

Invocation `CALL $BLANKPAGE(titles_yn)`

<i>titles_yn</i>	One of the following: Y – Specifies to print a title on the blank page. N or " (NULL) – Specifies not to print a title. Its syntax is C (fixed-length character string), with length 1.
------------------	--

- Prerequisites**
- The print arguments must be previously set with a call to either [\\$SETPRINT](#) or [\\$RESETPRINT](#).
 - The titles are previously specified with [\\$SETP#POS](#) or [\\$SETTITLE](#).

Exceptions

LOGLIMIT	Too much output is sent to the message log.
RANGERROR	<i>titles_yn</i> is neither Y nor N (or " (NULL), which is treated as N.)
ROUTINEFAIL	<code>\$BLANKPAGE</code> is not preceded by a call to <code>\$RESETPRINT</code> or <code>\$SETPRINT</code> .
STRINGSIZE	This exception is raised if the left, center, or right titles overlap or if the combined length of the character strings exceeds the <i>width</i> (where <i>width</i> is the page width set by <code>\$RESETPRINT</code> or <code>\$SETPRINT</code>) or 132, whichever is less.

Example The following rule prints three blank pages to the message log:

```
BLANKPAGE_1;
```

```

—
— -----
— -----+
— CALL $SETPRINT(10, 70, 1, 'SCR', 'N');          | 1
— CALL $BLANKPAGE('Y');                          | 2
— CALL $BLANKPAGE('N');                          | 3
— CALL $BLANKPAGE('Y');                          | 4
— -----
—
```

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P

----- NEW PAGE -----
Page 1

----- NEW PAGE -----

----- NEW PAGE -----
Page 3
```

The first and third pages are printed with titles, the second page without titles.

\$BRCONTAINER

Lists the 16-character container names and displays the count of the containers associated with the channel. (C)

This tool combines the functions of three CICS API commands, as follows:

- STARTBROWSE CONTAINER
- GETNEXT CONTAINER
- ENDBROWSE CONTAINER

Invocation `CALL $BRCONTAINER(channel, container-list, length, count)`

<i>channel</i>	The name (1-16 characters) of the channel whose containers are to be browsed. <i>channel</i> must be the name of either the current channel or a channel created by the session that issues \$BRCONTAINER.
<i>container-list</i>	The pointer to an area large enough to contain all the 16-character container names returned.
<i>length</i>	The length of the container-list area in question.
<i>count</i>	The total number of containers associated with the channel. <i>container-list</i> might not list all the names if it is not large enough for <i>count</i> .

Usage Notes The channel must be within the scope of the program. For details, see Chapter 7, “Using the TIBCO Service Gateway for CICS,” in the *TIBCO Service Broker for z/OS External Environments* manual.

Example Following is a sample rule.

<pre> RULE EDITOR ==> C_\$BRCONTAINER(CHANNEL); _ LOCAL AREA, CONTLIST, LISTSIZE, COUNT, I; _ ----- _ ----- _ @MAP.ADDRESS = 0; _ @MAP.SIZE = 80; _ INSERT @MAP('ENVIRONMENT'); _ CONTLIST = @MAP.ADDRESS; _ LISTSIZE = @MAP.SIZE; _ CALL \$BRCONTAINER(CHANNEL, CONTLIST, LISTSIZE, COUNT); _ CALL MSGLOG('\$BRCONTAINER(CHANNEL, CONTLIST, LISTSIZE, COUNT)'); _ CALL MSGLOG('Channel name is: ==> ' CHANNEL); </pre>	<pre> SCROLL: P 1 2 3 4 5 6 7 8 </pre>
--	--

```
_ CALL MSGLOG('Count of container is: ' || COUNT);
_ CALL MSGLOG('Containers are: ');
_ UNTIL EQ :
_   GET CCN_CONTLIST(CONTLIST);
_   CALL MSGLOG('      ' || CCN_CONTLIST.CONTAINER1);
_   I = I + 1;
_   CALL @EQ(I, COUNT);
_   CONTLIST = CONTLIST + 16;
_   END;
_ -----
```

Following is the MAP table CCN_CONTLIST:

COMMAND==>TABLE DEFINITION														
Table: CCN_CONTLIST						Type: MAP	Unit: HZS80			IDgen: Y				
Parameter Name	Typ	Syn	Len	Dc	Cls	Reference			Event	Rule	Typ	Acc		
-----							-	-	-----					-
ADDRESS		B	4	0	A									
LOCATION	I	C	16	0	L									
							-	-						-
Field Name							----- EXTERNAL ----- ----- Metadata Definition -----							
	Xsyn	Xlen	Xdec	Offset	Key	Typ	Syn	Len	Dec	Rqd	Default			
-----							-	-	-----					-
KEY	B		4	0	0	P	I	B	4	0				
CONTAINER1	C		16	0	0			C	16	0				
CONTAINER2	C		16	0	16			C	16	0				
CONTAINER3	C		16	0	32			C	16	0				

BROWSER

Displays the contents of a TIBCO Object Service Broker table for viewing. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type BROWSER(<i>tablespec</i>) <Enter>
	COMMAND prompt	Type EX BROWSER(<i>tablespec</i>) <Enter>
From a rule		Type CALL BROWSER(<i>tablespec</i>)

Where:

<i>tablespec</i>	The table name and the parameter, if necessary.
------------------	---

You should set the BROWSE flag to Y to ensure the data is not locked.

- Usage Notes**
- If you are using BROWSER to browse a table, you cannot edit an occurrence. You can use all the other commands supported within the Browser facility.
 - [STEBROWSE](#) is the interactive version of this tool used on the developer workbench.

See Also

TIBCO Object Service Broker Managing Data for additional information on the Table Browser.

Example

In the following screen, the user can browse a table containing data on the employees of a selected region:

Date: 2000-01-11

Employees by Region

- _ East
- _ Mideast
- _ Central
- _ Midwest
- _ West

FCNKEYS: ENTER=GET EMPLOYEES 12=EXIT

Screen Table Definition

The fields of the screen table containing the regions are named to match the parameters of the EMPLOYEES table, which is also parameterized by region:

```
SCREEN PAINTER COMMAND ==>                                Scroll: P
...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
-A East
-A Mideast
-A Central
-A Midwest
-A West
```

Table: REGIONS			Unit: BZDD0											
ROW	COL	FIELD NAME	Type	Syn	Len	Dec	Just	Fill	Prot	Show	Rqd	Hi	Skip	Null
1	2	EAST		C	1	0	L	—	N	Y	N	N	Y	Y
2	2	MIDEAST		C	1	0	L	—	N	Y	N	N	Y	Y
3	2	CENTRAL		C	1	0	L	—	N	Y	N	N	Y	Y
4	2	MIDWEST		C	1	0	L	—	N	Y	N	N	Y	Y
5	2	WEST		C	1	0	L	—	N	Y	N	N	Y	Y

PFKEYS: 6=+FLD 18=-FLD 4=+LINE 5=CUT 19=FLD_HELP 17=PASTE 16=-LINE 13=PRINT

GETREGION Rule

When you press Enter, the following rule returns the name of the field where the cursor is placed and calls BROWSER to display the contents of the table:

```
RULE EDITOR ==>                                SCROLL: P
GETREGION;
- LOCAL REG;
- -----
- -----+-----
- REG = CURSORFIELD('EMPLOYEE_SCR_C');           | 1
- CALL BROWSER('EMPLOYEES(' || REG || ')');       | 2
- -----
```

Resulting Output

When you place the cursor beside Midwest and press Enter, the following data appears for browsing:

BROWSING TABLE : EMPLOYEES(MIDWEST)
COMMAND ==>

SCROLL: P

EMPNO	LNAME	POSITION	MGR#	DEPTNO	SALARY
22001	DRABEK	CUST SUPPORT	56112	30	900.00
22007	ROEDER	CUST SUPPORT	56112	30	900.00
30058	HOEGSON	PRE-SALES	37219	20	675.00
34111	TERAMURA	PRE-SALES	37219	20	710.00
34121	LEES	CUST SUPPORT	56112	30	700.00
36162	MORANG	JR OPERATOR	44798	80	575.00
41001	CROFTON	TECH WRITER	80002	70	675.00
41007	STEVENSON	EDUCATOR	80002	60	700.00
41009	SMITH	TESTER	79912	50	600.00
44385	SOUZA	SALES	37219	10	719.00
44622	SAUNDERS	ACCOUNTANT	98895	40	800.00
51111	HRODEK	ANALYST	79912	50	710.00
51121	CANNON	ANALYST	79912	50	700.00
51162	KIMURA	JR PROGRAMMER	79912	50	575.00
61219	WONG	SENIOR ANALYST	79912	50	820.00
61385	DHILLON	EDUCATOR	80002	60	685.00
61622	SCHULTZ	SENIOR ANALYST	79912	50	800.00

PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND

\$CALLRULE

Invokes a procedural rule. (C)

Invocation `CALL $CALLRULE(rulecall)`

<i>rulecall</i>	A character string of syntax V (variable-length character string) or UN (Unicode) containing the name of the rule to be invoked, and, if the rule takes one or more arguments, a left parenthesis, a list of the comma-separated arguments, and a right parenthesis.
-----------------	--

Usage Notes

- You can pass a NULL value to the rule by omitting an actual value from the list of parameters in the position of the parameter to be passed the NULL. For example, in the sample rule below, to call ADDC without a value for B, you would use:

`CALL $CALLRULE('ADDC(' || A || ',')');`
- If the rule to be invoked does not exist in a library in the current search order of the transaction, an untrappable error occurs. You can use the \$RULE_EXISTS shareable tool to test for the existence of a rule before trying to invoke it.

Exceptions

EXECUTEFAIL	Raised if <i>rulecall</i> is not a string conforming to the format described above.
--------------------	---

Example

This rule receives a request to process two numbers by adding them together or subtracting them from each other. It then calls the appropriate user-written rule to do that processing. If one of the rules does not exist, the terminal displays an end message.

<pre> RULE EDITOR ==> ARITHC(A, OPERATION, B); LOCAL C; ----- \$RULE_EXISTS('ADDC') = 'N'; \$RULE_EXISTS('SUBTRACTC') = 'N'; OPERATION = '+'; OPERATION = '-'; ----- SIGNAL NORULE; CALL \$CALLRULE('ADDC(' A ', ' B ')'); CALL \$CALLRULE('SUBTRACTC(' A ', ' B ')');</pre>	<pre> SCROLL: P ----- Y N N N N Y N N N Y N N Y N ----- 1 1 1 1</pre>
---	---

```
_ CALL ENDMSG('THE RESULT OF ' || A || OPERATION || B ||      |      2 2
_ ' IS ' || C);                                           |
_ -----
_ ON NORULE :
_   CALL ENDMSG('RULE NOT FOUND');
_ ON EXECUTEFAIL :
_   CALL ENDMSG('CHECK SYNTAX IN RULE');
```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

This is one of the rules ARITHC calls:

```
      RULE EDITOR ==>                                SCROLL: P
ADDC(A, B);
_
_ -----
_ -----+-----
_ C = A + B;                                | 1
_ -----
```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

The output message from ARITHC is similar to:
THE RESULT OF 7+2 IS 9

CHANGE_SERVERID

Updates the server ID of any external TIBCO Object Service Broker data types. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type CHANGE_SERVERID (<i>table_name</i> , <i>old_serverid</i> , <i>new_serverid</i>) <Enter>
	COMMAND prompt	Type CHANGE_SERVERID (<i>table_name</i> , <i>old_serverid</i> , <i>new_serverid</i>) <Enter>

Where:

<i>table_name</i>	The TIBCO Object Service Broker name of the external table. Note: You do not have to specify the table type.
<i>old_serverid</i>	The current value of the server ID
<i>new_serverid</i>	The new value for the server ID

Usage Notes

- This tool is especially useful when a dedicated server is required to obtain trace information.
- CHANGE_SERVERID provides an alternate way to update the server ID. Normally, one would update the server ID of an external table type through the Table Definer of that type.
- The server for the external table does not have to be running.
- For DB2, the new server ID must be predefined by the system administrator who manages DB2 table definition. Refer to *TIBCO Service Gateway for DB2 Installing and Operating*.
- If a match of tablename and server ID is not found, the tool returns with OK, does not update the server ID of the table specified, and writes the details to the MSGLOG.



There is no check in place to ensure that the new server ID is valid. These checks are built in under the Table Definer.

Example The server ID for DB2_EMPLOYEE is to be changed from DB2SVR1 to dedicated server DB2SVR2 to improve access performance. From the workbench, an authorized user goes to the EX Execute rule entry on the menu and types:

```
CHANGE_SERVERID(DB2_EMPLOYEE, DB2SVR1, DB2SVR2)
```

Accesses to the DB2_EMPLOYEE table are then serviced by server DB2SVR2.

CHANGERULE

Makes multiple text changes across multiple rules in a library. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type CHANGERULE <Enter>
	COMMAND prompt	Type EX CHANGERULE <Enter>
From a rule		Type CALL CHANGERULE

Usage Notes

With CHANGERULE, you can make multiple text changes to all or some of the rules in a library. It changes any occurrences of the string that appear as the following:

- LOCAL definition
- LOCAL reference
- Table name
- Argument
- Rule name
- Total string



To guard against unexpected changes, consider applying changes to a copy of the rules library.

To use CHANGERULE, complete the following steps:

1. Enter the name of the library you are making the changes in.
2. To restrict the changes to a specific rule or set of rules, enter the rule name with the appropriate wildcard characters.

You can use an exact string or use the wildcard characters "?" (single character) or "*" (multiple characters) in your search string.

3. To restrict the changes to a specific unit or set of units, enter the unit name with the appropriate wildcard characters.

You can use an exact string or use the wildcard characters "?" (single character) or "*" (multiple characters) in your search string.

- 4. Enter the tokens or strings you want to change and the values to which you want to change them.

Changing is done on a token basis, so you cannot change an entire string like *TABLENAME.FIELDNAME*, although you can enter separate changes for *TABLENAME* and *FIELDNAME*.
- 5. Press PF5 to make the changes.

Example This screen makes the following changes to all rules in library DOCMSG2 with names like CUSTINFO*:

- The tables \$CUST and CUST are changed to \$CUSTOMER and CUSTOMER.
- The fields NUM and LNAME are changed to NUMBER and LASTNAME.

Global Change Rules Utility

Library: DOCMSG2 Rules Like: custinfo* Unit Like: *

From Value	To Value
-----	-----
\$cust_____	\$customer_____
cust_____	customer_____
num_____	number_____
lname_____	lastname_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

PFKEYS: 5=CHANGE 2=LOG 12=EXIT

Message Log

When you press PF5, the following text appears in the message log:

Preparing to change from \$CUST to \$CUSTOMER
Preparing to change from CUST to CUSTOMER
Preparing to change from NUM to NUMBER
Preparing to change from LNAME to LASTNAME

Start scan at 1:36:32

```
Checking rule CUSTINF01
Token found
Token found
Token found
Rule CUSTINF01 - Changed $CUST to $CUSTOMER
Rule CUSTINF01 - Changed CUST to CUSTOMER
Rule CUSTINF01 - Changed NUM to NUMBER
```

```
Checking rule CUSTINF02
Token found
Token found
Token found
Rule CUSTINF02 - Changed $CUST to $CUSTOMER
Rule CUSTINF02 - Changed CUST to CUSTOMER
Rule CUSTINF02 - Changed LNAME to LASTNAME
```

```
End scan at 1:36:33
```

CLEARTABLE_APPL

Deletes occurrences from a table or table instance. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	CL Clear Table Option	Press <Enter>. The prompt screen appears.
	CL Clear Table Option	Type <i>table</i> [, <i>select</i>] <Enter>
	EX Execute Rule option	Type CLEARTABLE_APPL <Enter>. The prompt screen appears.
	COMMAND prompt	Type EX CLEARTABLE_APPL <Enter>. The prompt screen appears.

Where:

<i>table</i>	Name of the table and, if applicable, the parameter
<i>select</i>	[Optional] Selection string consisting of a field name, a relational operator, and a value

Usage Notes CLEARTABLE_APPL deletes occurrences from a single table or table instance. To clear occurrences from a set of table instances, use the [\\$CLR TAB](#) rule.

Using CLEARTABLE_APPL From the Workbench

- To delete all occurrences in a non-parameterized table or a table instance, type the name of the table, the parameter if applicable, and a comma. For example:

```
CL Clear Table ==> EMPLOYEES(MIDWEST),
```

- To delete selected occurrences, type the name of the table, the parameter if applicable, a comma, and a selection string. For example:

```
CL Clear Table ==> EMPLOYEES(MIDWEST),NUM>2
```



When invoking `CLEARTABLE_APPL` from the workbench, the selection string must be based on a field that contains numeric rather than alphabetic data. To select on alphabetic data, execute `CLEARTABLE_APPL` from the prompt screen and refer to the following section.

Using `CLEARTABLE_APPL` from the Prompt Screen

- To delete all occurrences within a non-parameterized table or a table instance, type the name of the table and the parameter if applicable in the Table Name field and press Enter. For example:

```
Table Name ==> EMPLOYEES(MIDWEST)
```

- To delete selected occurrences, type the name of the table and the parameter if applicable in the Table Name field, and then type a selection string in the Select Occurrences Where field. For example, to delete all employees named Smith from `EMPLOYEES(MIDWEST)`, type the following:

```
Rule Execute Utility CLEAR TABLE
```

```
Table Name ==> EMPLOYEES(MIDWEST)
```

```
Select Occurrences Where  
==> LNAME='Smith'
```

Example

The following example uses the CL Clear Table option to remove all occurrences in `EMPLOYEE_DEPT(10)`:

```
CL Clear Table ==> EMPLOYEE_DEPT(10),
```

Using the prompt screen, the following entries remove all occurrences in the non-parameterized table `EMPLOYEE` that have the field `DEPTNO` equal to or greater than 20:

```
Rule Execute Utility CLEAR TABLE
```

```
Table Name ==> EMPLOYEE
```

```
Select Occurrences Where  
==> deptno>=20
```


@CLOSEDSN

Closes and frees the current file. (C)

Invocation CALL @CLOSEDSN

- Usage Notes**
- [@WRITEDSN](#) and [@READDSN](#) can be used to write to and read from the file.
 - If [@CLOSEDSN](#) is not used after [@READDSN](#) or [@WRITEDSN](#), the file is closed automatically at the end of the transaction.
 - If no file is open, [@CLOSEDSN](#) is treated as [NOOP](#).
 - [@CLOSEDSN](#) accesses a z/OS file using the data set name. There is no provision for using a DDNAME with this tool instead of a data set name.

Example The following rule specifies the file to use, writes data from the example table to it, closes the file, re-specifies it, reads back the first record from it, and prints that record to the message log:

```
CLOSEDSN_1;
_ LOCAL RECORD;
_ -----
_ -----+-----
_ CALL @OPENDSN(TSOID || ' .EXAMPLES.DATA '); | 1
_ FORALL EMPLOYEE : | 2
_     CALL @WRITEDSN(EMPLOYEE.LNAME); |
_     END; |
_ CALL @CLOSEDSN; | 3
_ CALL @OPENDSN(TSOID || ' .EXAMPLES.DATA '); | 4
_ RECORD = @READDSN; | 5
_ CALL MSGLOG(RECORD); | 6
_ CALL @CLOSEDSN; | 7
_ -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following output:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
SMYTHE
```

\$CLRTAB

Deletes (clears) the data rows from a table or table instance without reading the data rows. (C)

Invocation `CALL $CLRTAB(tablename, parm1, parm2, parm3, parm4)`

<i>tablename</i>	The name of the table to be cleared. Its syntax is C with length 16.
<i>parm1</i>	The value for the first data parameter of <i>tablename</i> .
<i>parm2</i>	The value for the second data parameter of <i>tablename</i> .
<i>parm3</i>	The value for the third data parameter of <i>tablename</i> .
<i>parm4</i>	The value for the fourth data parameter of <i>tablename</i> .

Usage Notes

- \$CLRTAB is a replacement for the CLRTAB tool. CLRTAB exists for backward compatibility only.
- Because \$CLRTAB does not take part in the TIBCO Object Service Broker two-phase commit protocol, it does not clear the intent list. It affects only the information in the specified table. Therefore, if you use \$CLRTAB in a transaction that also updates data within the cleared table, you can get the following, possibly undesired, result: occurrences previously in the intent list are saved to the table at the next commit after the clearing of the table.

To prevent this, commit insertions and deletions before calling \$CLRTAB.

- \$CLRTAB does not clear the current-occurrence buffer. It affects only the information in the specified table. For example, if you do the following:
 - Assign values to the fields of a table
 - Run \$CLRTAB
 - Update one field
 - INSERT the occurrence

the other fields of the inserted occurrence still contain the values you assigned. If you intend that the table be empty except for your newly assigned value, run \$CLRTAB as a separate transaction followed by the update.

- The table must be of type TDS, TEM, SES, EES, SCR, or DB2.
- If called from within a transaction in BROWSE mode, \$CLRTAB clears TEM, SES, EES, and SCR tables; it does not clear TDS or DB2 tables.

- You must have delete (DEL) access to the table to use \$CLRTAB successfully.
- If the table is parameterized, the number of data parameter values supplied must equal the number of data parameters in the table definition. For example, if the table has fewer than four data parameters, type two single quotation marks (") for each argument that is not required. If the quotation marks are not included for the excess arguments, the transaction fails with a ROUTINEFAIL exception.
- \$CLRTAB ignores location parameters. In case of minimal table definition, \$CLRTAB does nothing.
- In the case of an SCR table, there must be exactly one parameter: the screen name.
- \$CLRTAB does not trigger any event rules during its execution.
- The parameter values supplied must be convertible to the syntax of the parameter definitions.
- To delete a specific table instance, specify its parameter values in the arguments *parm1*, *parm2*, *parm3*, and *parm4*.
- To delete all the data in a parameterized table, specify a set of single quotation marks (") for each of the arguments *parm1*, *parm2*, *parm3*, and *parm4*.
- [CLEARTABLE_APPL](#), which you can use from the CL clear table option on the standard workbench, can also be used to clear table data. This interactive tool reads and deletes each table row.



If \$CLRTAB fails, you must run it again before doing another database update to prevent damage to the database. If \$CLRTAB fails again, contact your database administrator or TIBCO Support immediately.

Exceptions

DEFINITIONFAIL	Raised if the definition of the table does not exist or is inconsistent.
SECURITYFAIL	Raised if the user is not authorized to delete from the table.
ROUTINEFAIL	<p>Raised if any argument fails validation, that is, for conversion errors or incorrect table type</p> <p>It is also raised if, when clearing an instance of a table that has multiple data parameters, some of the data parameters for the instance are missing as arguments.</p>

Example The following example deletes a table instance from the table EMPLOYEE_DEPT:

```
$CLRTAB_1;  
--  
-- -----  
-- -----+-----  
-- CALL $CLRTAB('EMPLOYEE_DEPT','30','','',''); | 1  
-- -----
```

The following example deletes all the data from all table instances of the table EMPLOYEE_DEPT:

```
$CLRTAB_2;  
--  
-- -----  
-- -----+-----  
-- CALL $CLRTAB('EMPLOYEE_DEPT','','','',''); | 1  
-- -----
```

@CONFIGURESERVER

Sets and modifies the server configuration parameters for a particular server ID.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type @CONFIGURESERVER(<i>type</i>) <Enter>
	COMMAND prompt	Type EX @CONFIGURESERVER(<i>type</i>) <Enter>

Where:

<i>type</i>	The type of server for which you want to set or modify configuration parameters. Valid values are ADA, IMS, ORS, and SLK.
-------------	---

Usage Notes

If for type you specify...	Do this
ADA, IMS	Only the server IDs for the specified type appear.
ORS, SLK or nothing	All server IDs appear.

See Also The appropriate TIBCO Service Gateway manual for information on installing servers and on available server configuration parameters for each server type.

Example The following example illustrates how to add a new server ID and set its configuration parameters.

1. Type @CONFIGURESERVER at the EX execute rule option on the workbench.
2. Press Enter. This displays existing server IDs for the specified *type* (in this example, IMS is specified for *type*). The following screen appears:

Command ==>	Scroll P
NUMBER	SERVERTYPE SERVERID

```

-----
-          5 IMS      AZI11
-          6 IMS      NJSIMS
-          7 IMS      NJSIMS1
-          8 IMS      NJS30
-          9 IMS      NJS32IMS

```

3. Type an A beside one of the SERVERID entries.
4. Press Enter. The following screen appears:

```

To complete this command:
      NUMBER      SERVERTYPE  SERVERID
-----
A          5 IMS      AZI11
Enter parameter(s):

SERVERTYPE      ==>
SERVERID        ==>

```

5. Enter the appropriate values for SERVERTYPE and SERVERID.
6. Press Enter.

The following screen appears, enabling you to set or modify the configuration parameters for the server ID you specified:

```

External Server Configuration Utility
COMMAND ==>
Server Type: IMS      Server ID: IMS1

Name      Value      Recommended/
-----
DEBUGLEVEL 0          '0...9'
DUMP        N          Y, N
DUMPLIMIT  512         '0...2147483647'
EXTIOBUFFERSIZE 32768  32768
KEEPLOG     N          Y, N
LOGMEDIA    SCR        * TBL, SCR, PRT
TRACE       N          Y, N

```

7. After setting or modifying the parameters, press PF3 to save the settings and return to the workbench.

CONFIRMATION

Issues a confirmation message for a PF key action or for a specified command. (C)

Invocation	CALL CONFIRMATION(<i>screen</i> , <i>confirmmsg</i> , <i>key</i> , <i>defaultmsg</i> , <i>table</i> , <i>commandfield</i>)	
	<i>screen</i>	The name of an existing screen
	<i>confirmmsg</i>	The confirmation message
	<i>key</i>	Enter the name of the PF key that is to be used to confirm the action
	<i>defaultmsg</i>	A string containing the default message that is to appear when the action is completed
	<i>table</i>	The name of the screen table used for primary commands
	<i>commandfield</i>	The name of the screen field used for primary commands

Usage Notes Text is concatenated to *confirmmsg* telling the user how to confirm the action.

Exceptions	ACTION_CANCELLED	Raised when the PF key used for confirmation is not the key specified in <i>key</i>
------------	-------------------------	---

Example The rules in this example do the following:

1. Display a screen for data entry.
2. Get an occurrence to be deleted.
3. Ask for confirmation of the deletion.
4. Accept or cancel the deletion depending on the answer.

DELETE_EMPLOYEE Rule

DELETE_EMPLOYEE displays the screen DELETE_EMPLOYEE and initializes the function keys:

RULE EDITOR ==>> DELETE_EMPLOYEE; —	SCROLL: P
---	-----------

```

-----
FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('DELETE_EMPLOYEE');      | 1
INSERT FCNKEY_SPECS('DELETE_EMPLOYEE');                     | 2
UNTIL EXIT_DISPLAY DISPLAY DELETE_EMPLOYEE :                 | 3
    CALL PROCESS_FCNKEY('DELETE_EMPLOYEE');                   |
END;                                                           |
-----

```

DEL_EMP_1 Rule

DEL_EMP_1 is the rule initiated when you press PF22. It gets the requested occurrence, prompts for confirmation with CONFIRMATION and deletes the occurrence if confirmation is given:

```

RULE EDITOR ==>                                           SCROLL: P
DEL_EMP_1;

-----
GET EMPLOYEE_INFO('DELETE_EMPLOYEE');                      | 1
GET EMPLOYEES(EMPLOYEE_INFO.REGION) WHERE EMPNO =          | 2
    EMPLOYEE_INFO.EMPNO;                                    |
CALL CONFIRMATION('DELETE_EMPLOYEE',                        | 3
    'ABOUT TO DELETE EMPLOYEE ' || EMPLOYEE_INFO.EMPNO,   |
    'OK', '', '', '');                                     |
DELETE EMPLOYEE(EMPLOYEE_INFO.REGION);                      | 4
CALL SCREENMSG('DELETE_EMPLOYEE',                           | 5
    'Deleted employee with no.: ' || EMPLOYEE_INFO.EMPNO ||
    '.');                                                    |
-----
ON ACTION_CANCELLED :
    CALL SCREENMSG('DELETE_EMPLOYEE', 'Deletion cancelled');
ON GETFAIL :
    CALL SCREENMSG('DELETE_EMPLOYEE',
        'Employee does not exist, re-enter a valid employee number');

```

COPY_DATA

Copies data from one table or table instance to another table or table instance. (C)

Invocation `CALL COPY_DATA(srctabspec, select, desttabspec, srclocation, destlocation, overwrite)`

<i>srctabspec</i>	The name of the source table or table instance
<i>select</i>	The selection criteria to be used, if selection is required
<i>desttabspec</i>	The name of the destination table or table instance
<i>srclocation</i>	The name of the node where the source data is located
<i>destlocation</i>	The name of the node where the destination data is to be located
<i>overwrite</i>	Specifies whether to overwrite existing occurrences, if the table exists. Valid values are: Y – Overwrite existing occurrences. N – Do not overwrite existing occurrences.

- Usage Notes**
- If the tables specified in *srctabspec* and *desttabspec* are parameterized, specify only data parameters, not location parameters.
 - If you specify an empty string for *select*, all occurrences are copied from the table.
 - When copying IDgen tables, the IDgen key values are regenerated on the target table. IDgen key values do not change if batch utilities are used.
 - The syntax for *select* is *field_name relational_operator value*.
 - Specify only values for *srclocation* and *destlocation* if the data are located on different nodes.

Exceptions

NO_DATA_FOUND	Raised if the source table/instance is empty
COPYFAILED	Raised if anything goes wrong with the copy operation

Both exceptions should be handled by the calling rule. More information describing the circumstances of the failure is in the @OBJECTMSG.MSG field.

Example COPY_DATA_1 Rule

The following rule copies selected occurrences from the table instance EMPLOYEES_REMOTE(CANADA) on node NODE3, to the table EMPLOYEE_CENTRAL on node NODE3B:

RULE EDITOR ==>		SCROLL: P
COPY_DATA_1;		

CALL COPY_DATA('EMPLOYEES_REMOTE(CANADA)', 'MGR#=56112',		1
'EMPLOYEE_CENTRAL', 'NODE3', 'NODE3B', '');		

COPY_DATA_2 Rule

The following rule copies all instances from table EMPLOYEES_REM to EMPLOYEES. Both tables are on the same node.

RULE EDITOR ==>		SCROLL: P
COPY_DATA_2;		
LOCAL A, B;		

FORALL \$EMPLOYEES_REM :		1
A = 'EMPLOYEES_REM(' \$EMPLOYEES_REM.REGION ')';		
B = 'EMPLOYEES(' \$EMPLOYEES_REM.REGION ')';		
CALL COPY_DATA(A, '', B, '', '', '');		
END;		

COPY_DEFN

Copies the definition of one or more objects from a source location to a destination location. (C)

Invocation `CALL COPY_DEFN(objecttype, instancename, library, environment, srclocation, destlocation, parentonly)`

<i>objecttype</i>	The TIBCO Object Service Broker object type of the object that is to be copied. Valid object types are: <ul style="list-style-type: none"> • GLOBALFIELDLIBRARY • MENU • OBJECTSET • REPORT • RULE • SCREEN • TABLE • WEBSERVICEPROD
<i>instancename</i>	Specifies the name of the object that is to be copied.
<i>library</i>	Specifies the name of the rules library where the rule is stored if the object is a rule.
<i>environment</i>	This argument, although not currently used, must be supplied. You can enter a null (") value.
<i>srclocation</i>	Specifies the name of the node where the source object is located.
<i>destlocation</i>	Specifies the name of the node where the destination object is located.
<i>parentonly</i>	Specifies if all the objects or only the parent object should be copied. Valid values are: <p>Y – Copy only the parent.</p> <p>N – Copy the parent and child objects.</p>

Populating the @DEFNS Table

Before invoking COPY_DEFN, you must populate the @DEFNS table.

If you do not populate the table, it is populated automatically with **newname=name**, **newlibrary=library**, and **overwrite=N**.

Use rules statements to populate the @DEFNS table, before calling COPY_DEFN.

Parameters of @DEFNS

The parameters of the @DEFNS table are as follows:

OBJECTTYPE	The type of object being copied
INSTANCENAME	The name of the object being copied

Fields of @DEFNS

The fields of the @DEFNS table are as follows:

NAME	The name of the object to be copied
NEWNAME	The name of the new object
TYPE	<p>The type of the object (for example, rule or screen). If the object is a table, use the subtype name, such as TDS_TABLE or SCR_TABLE, or execute the OBTAINSUBTYPE(<i>objtype</i>, <i>objname</i>, <i>lib</i>, <i>env</i>, <i>srcloc</i>) rule to determine the subtype name, where the following are the arguments:</p> <ul style="list-style-type: none"> • <i>objtype</i> – The general type of the object (for example, TABLE). • <i>objname</i> – The name of the object. • <i>lib</i> – The library, if the object is a rule. • <i>env</i> – The environment (default is 3270). • <i>srcloc</i> – The node where the object exists.
LIBRARY	The name of the library where the rule resides, if the object is a rule
NEWLIBRARY	The name of the library where the rule is copied

VERSIONED	Y if the object is a rule; otherwise, N
OVERWRITE	Y if an object that already exists should be overwritten

Usage Notes

- [COPYDEFN](#) is an interactive version of this tool used on the workbench.
- You must specify values for *srclocation* and *destlocation*.
- The definition is copied to segment 0 on the destination location.
- Specify a value for *parentonly* if the object is composed of one or more other objects, for example, a report is composed of report tables.
- The definition must not exist on the destination location.
- References to event rules and reference tables are copied, not the rules and tables themselves.
- Help text is copied for a screen since it is part of the definition of the screen.
- The original statistics for the object or object set (for example, CREATED, AUTHOR, UNIT, and SEGMENT) are also copied to the new library.
- If you are copying to a remote node, your user ID must be identical on both nodes or security complications could occur.

Exceptions

COPYFAILED	Raised if anything goes wrong with the copy operation. The exception should be handled by the calling rule. Further information describing the circumstances of the failure is in the @OBJECTMSG.MSG field.
------------	--

Example

The following rule copies the definition of the screen NEW_EMPLOYEES and its child objects from NODE1 to NODE2:

RULE EDITOR ==>		SCROLL: P
COPY_DEFN_1;		

@DEFNS.NAME = 'NEW_EMPLOYEES';		1
@DEFNS.NEWNAME = 'NEW_EMPLOYEES';		2
@DEFNS.TYPE = 'SCREEN';		3
@DEFNS.LIBRARY = '';		4
@DEFNS.NEWLIBRARY = '';		5
@DEFNS.VERSIONED = 'N';		6
@DEFNS.OVERWRITE = 'Y';		7
INSERT @DEFNS('SCREEN', 'NEW_EMPLOYEES');		8

```
_ CALL COPY_DEFN('SCREEN', 'NEW_EMPLOYEES', '', '', 'NODE1', | 9
_ 'NODE2', 'N'); |
_ -----
```

COPYDEFN

Copies the definition of one or more TIBCO Object Service Broker objects or object sets. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	CD Copy Defn option	Press <Enter>
	EX Execute Rule option	Type CD <Enter>
	COMMAND prompt	Type EX CD <Enter>

Usage Notes

- `COPY_DEFN` is the callable version of this tool.
- `COPYDEFN` cannot be run in browse mode.
- Help text is copied for a screen since it is part of the definition of a screen. The definition is copied to segment 0 on the destination location. The original statistics for the object or object set (for example, `CREATED`, `AUTHOR`, `UNIT`, and `SEGMENT`) are also copied. References to event rules and reference tables are copied, not the rules and tables themselves.
- PF keys and their associated rules are not copied since they are not part of the definition of a screen.
- Invoking `COPYDEFN` displays the following screen:

```

COPY DEFINITION
COMMAND ==>
Scroll: P

Source
-----
Location      :
Default Library:
Environment    :

Destination
-----
Location:
Library :

Name      Type      Library  x      Name      xwrite  Over Parent
-----  -----  -----  x      -----  x      -      Only
-
-
-
-
-
-
-
-
-
-

```

PFKEYS: 2=LOG 4=COPY 7=UP 8=DOWN 3=EXIT 12=EXIT 5=SELECT OBJECTS

To view additional fields from the screen, position your cursor in the appropriate section and use PF11 to scroll right. You can type data directly into the fields displayed or you can use PF5 from within the Copy Definition screen to display a screen that you can use to select the objects that you require.

Copy Definition Screen

The Copy Definition screen is composed of five sections. Each section is described below.

Optional base settings

This section identifies any outside nodes and rules libraries for the copy process. Information entered here applies to all source and destination objects listed in subsequent sections, unless specifically overridden on a line-by-line basis. Entries in this section are optional, as the cursor immediately homes to the next section—Source Object Attributes—when executing this tool.

Source Location	If the objects are to be copied to a remote location, type the name of the node where the source objects are located.
Source Default Library	Using this field, you can specify a default library that contains the rules to be copied. Type in the name directly, or press PF1 for a list of available libraries. The default library can be overridden on a line-by-line basis by individual entries in the LIBRARY field.
Source Environment	This field is currently not in use.
Destination Location	If the objects are to be copied from a source location, type the name of the node where the objects are being copied.
Destination Library	If the object is a rule, enter the name of the library where the rule is copied. Use PF1 for a list of valid values.

Source Object Attributes

Each line in this section names a source object and gives additional information about it. You can enter in this information a line at a time or pull it all across from the section screen.

Name	The name of the object to be copied. Use PF5 to display a screen for selection. Refer to Select Objects Screen on page 112 for more information about selecting objects.
Type	The name of the object type. Use PF1 a list of valid values.
Library	If the object is a rule, type the name of the rules library where the rule to be copied is located. Use PF1 for a list of valid values. Entries in this field override any value displayed in the Source Default Library field.

Destination Object Attributes

This section is used to enter information about the destination objects.

Name	The name that the copied object is to be called. If the object is to be copied to a remote location, you do not have to specify a new name. If you do not specify a value, the name of the source object is used.
------	---

Copy Attributes

This section contains further information relating to the copy objects.

Overwrite	If the object exists, specify whether to overwrite the existing definition. Valid values are: Y – Overwrite existing definition. N – Do not overwrite existing definition.
-----------	--

Parent Only	<p>If the object is composed of child objects (for example, an object set is composed of objects), specify if the definitions of the child objects should also be copied. Valid values are:</p> <p>Y – Copy only the definition of the parent.</p> <p>N – Copy the definitions of the parent and children without specifying the child objects as individual items.</p>
--------------------	---

Object's Parent Attributes

This section contains further information relating to the object's parent attributes.

Name	<p>If you are copying a parent object and a child object and you are renaming the child object in the destination location, type the name of the source parent object on the line where you are specifying the child object information.</p> <p>Use this only if you still want the child object to be a part of the parent definition. References to the child object in the definition of the parent object are then changed to refer to the child by its new name.</p>
Type	Specifies the object type of the object specified in the Name field.

PF Keys Available

You can use the following PF keys from within this screen:

PF2	Displays the message log.
PF3	Exits the screen and return to the workbench.
PF4	Copies the definition as defined.
PF5	Displays a screen to be used to select objects.
PF7	Scrolls up.
PF8	Scrolls down.
PF10	Scrolls left.
PF11	Scrolls right.

PF12 Exits the screen and return to the workbench.

Select Objects Screen

Pressing PF5 from the Copy Definition screen displays the following screen:

Object Selection

COMMAND ==>

Location:

Library (for RULES):

Presentation Environment:

Select All: N

List Children: N

Attr

Op

Value

NAME

TYPE

UNIT

AUTHOR

=

AND

AND

AND

unspecified

attributes will

be ignored

Scroll:

Name

Type

Library

Environment

Unit

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

Specify the following information in the fields:

Location	Specify the name of the node where the selection criteria are applied.
Library	If the selection list is to contain rules, type the name of the rules library that is searched. Press PF1 for a list of valid values.
Presentation Environment	This field is currently not in use.

Select All	<p>Specify whether all the items displayed, based on the selection criteria, should be copied into the Copy Definition screen.</p> <p>Y – Copy all items displayed.</p> <p>N – Do not copy all items displayed.</p>
List Children	<p>Specify if you want to list all the child objects that an object is composed of. Valid values are:</p> <p>Y – List all child objects.</p> <p>N – Do not list child objects.</p> <p>The middle section of the screen can be used to select the items to be copied or to narrow down the selection list. You can use more than one type of selection criteria for each object type and you can specify multiple object types within one session. For a list of valid values for each of these fields, press PF1.</p> <p>When you use Enter after specifying the selection criteria, the selected items appear in the bottom portion of the screen. You can select the objects displayed in this section by typing an S in the line command field beside the objects.</p>
NAME	<p>If you know the name of the item, in the Op field type the logical operator to be used. In the Value field, type the name of the object.</p>
TYPE	<p>The name of the object type. Use PF1 for a list of valid values. If you do not supply an object type, you must specify a value in at least one of the other selection fields.</p> <p>If you specify an object type and no further selection values, a listing of the items for the object type defined in your TIBCO Object Service Broker database appears for further selection.</p>
UNIT	<p>In the Op field, type the logical operator to be used. In the Value field, type the name of the Unit associated with the object.</p>
AUTHOR	<p>In the Op field, type the logical operator to be used. In the Value field, type the name of the author of the object.</p>

PF Keys Available

You can use the following PF keys from within this screen:

Enter	Updates the screen.
PF3	Saves the selection and returns to the Copy Definition screen.
PF12	Exits without selecting objects and returns to the Copy Definition screen.

Example This example copies the EMPLOYEES_3DPARM table to the EMPTABLE_3NEW table.

COPY DEFINITION

COMMAND ==>Scroll: P

Source

Location :
Default Library:
Environment :

Destination

Location:
Library :

Name	Type	Library	Name	Over write	Parent Only
-----	-----	-----	-----	-	-
_ EMPLOYEES_3DPARM	TABLE		EMPTABLE_3NEW	N	N
-					
-					
-					
-					
-					
-					

PFKEYS: 2=LOG 4=COPY 7=UP 8=DOWN 3=EXIT 12=EXIT 5=SELECT OBJECTS

Message Log

Press PF4 to copy the definition. Pressing PF2 displays a log similar to the following listing the results of the copy.

----- INFORMATIONAL MESSAGE LOG -----

COMMAND ==>SCROLL ==> P

**** Begin to copy object "EMPLOYEES_3DPARM" ****

Object 'EMPLOYEES_3DPARM' is copied to 'EMPTABLE_3NEW'

**** Copy process ended ****

COPYLIB

Copies all the rules from a source library to a destination library. (C)

Invocation `CALL COPYLIB(source_lib, dest_lib)`

<i>source_lib</i>	The name of the source library
<i>dest_lib</i>	The name of a predefined destination library

- Usage Notes**
- The local variable *MSG* must be declared by the calling rule. It contains a message indicating how many rules were copied, if the copy is successful, or a message indicating that the copy failed.
 - If a rule in the copied library exists in the destination library it is overwritten.

Exceptions

LIBCOPYFAILED	Returns a message if the copy fails.
----------------------	--------------------------------------



The exception must be handled by the calling rule.

Example The following rule copies the rules from the USR40 library to the DOCUMENT library and returns the status of the copy to the ENDMSG:

```

      RULE EDITOR ==>
COPYLIB_EX;
_ LOCAL MSG;
_ -----
_ -----+-----
_ CALL COPYLIB('USR40', 'DOCUMENT');          | 1
_ CALL ENDMSG(MSG);                          | 2
_ -----
_ ON LIBCOPYFAILED:
_ CALL ENDMSG(MSG);
```

It returns the following ENDMSG:
Library copy succeeded, 75 rules copied.

COPYTABLE_APPL

Copies selected occurrences from a source table to a destination table. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	CT Copy Table option	Press <Enter>
	EX Execute Rule option	Type CT <Enter>
	COMMAND prompt	Type EX CT <Enter>

The following screen appears when COPYTABLE_APPL is executed:

```
Rule Execute Utility COPY TABLE

Source Table Name ===>
Destination Table Name ===>

Select Occurrences Where
===>

PFKEYS: ENTER=COPYTABLE 1=HELP 3=EXIT 12=EXIT
```

- Usage Notes**
- Only those fields with the same name in both the source and destination tables are copied. If the occurrence already exists in the destination table, the occurrence is replaced.
 - COPYTABLE_APPL fails if source data is too large for a destination field.
 - When copying IDgen tables, the IDgen key values are regenerated on the target table. IDgen key values are not changed if batch utilities are used.

- An empty string specified for Select Occurrences results in the selection of all occurrences.
- The syntax for selection is *<field name> <relational operator> <value>*.

Example The following entries to the prompt screen copy all occurrences from the table EMPLOYEE_DEPT(10) to the table EMPLOYEE_DEPT(30):

Rule Execute Utility COPY TABLE

Source Table Name ==> EMPLOYEE_DEPT(10)

Destination Table Name ==> EMPLOYEE_DEPT(30)

Select Occurrences Where
==>

PFKEYS: ENTER=COPYTABLE 1=HELP 3=EXIT 12=EXIT

COUNTOCCURRENCES

Returns the number of occurrences that meet a selection criteria. (F)

Invocation `count = COUNTOCCURRENCES(table, selection)`

<i>count</i>	The number of occurrences meeting the selection criteria.
<i>table</i>	The table name.
<i>selection</i>	The selection criteria. Its syntax can be C (fixed-length character string), V (variable-length character string), or UN (Unicode).

- Usage Notes**
- COUNTOCCURRENCES can be used for TDS, parameter (PRM), temporary (TEM), calculation (CLC), session (SES) tables, and Execution Environment (EES) tables.
 - COUNTOCCURRENCES cannot be used for remote tables.
 - If the table is parameterized, specify the parameter values in the selection criteria.
 - You can base the selection criteria on any field of the table. For optimum performance, base the selection on a secondary index field.

Example The following rule returns the number of employees who work for a regional company. The EMPLOYEE_REGION table is parameterized by region and MGR# is a secondary index field.

```

COUNT_STAFF(AREA, MANAGER);
_ LOCAL SELECTION;
_ -----
_ -----+-----
_ SELECTION = 'REGION = ' || AREA || ' ' & MGR# = ' ' ||      | 1
_ MANAGER || ' ' ;
_ RETURN(COUNTOCCURRENCES('EMPLOYEE_REGION', SELECTION));    | 2
_ -----

```

If you executed this rule from the workbench and passed in the values of MIDWEST for the argument AREA and 56112 for the MGR#, the following result is returned at the bottom of your screen:

8:41:03 Result of rule COUNT_STAFF is 3

\$CREATE_DATE

Converts a string with a specified format to a value of semantic type date. (F)

Invocation `date = $CREATE_DATE(pic_string, date_string)`

<i>date</i>	On return, contains the date. Its syntax is B (binary).
<i>pic_string</i>	The date format to be used. Its syntax is C (fixed-length character string), UN (Unicode), or V (variable-length character string).
<i>date_string</i>	The string that is to be converted. Its format must be the same as the format given in <i>pic_string</i> . Its syntax is C, UN, or V.

Usage Notes • The following lists valid date formats:

Format Code	Meaning	Example
W	One- or two-digit week # (of year), with no leading 0.	1 or 25
WW	Two-digit week # (of year).	01
WWW	Abbreviated weekday.	Thur
WWWW	Full weekday.	Thursday
M	Numeric month, with no leading 0 (1 or 2 digits).	3 or 10
MM	Numeric month (2 digits).	02
MMM	Abbreviated month.	Mar
MMMM	Full month.	March
D	Day in month, with no leading 0 (1 or 2 digits).	5 or 14
DD	Day in month (2 digits).	02
DDD	Day in year (3 digits).	074
YY	Last two digits in a year.	98
YYYY	Full year.	2000

Format Code	Meaning	Example
QQ	Two-character quarter.	2Q
JD	Julian date.	98.074
CC	Two-digit century.	19

- A separator character can be any one of the following:
/ \ ; : , . * - blank
- If no date format is specified, the default installation date format is used.
- A week is defined to begin on a Monday and end on the following Sunday. However, January 1st always begins week one, regardless of where it falls in the week, and week two starts on the following Monday.



You can specify just a portion of a date field within your mask (for example, entering only MMMM displays the month). Partial date occurrences cannot be accessed using a GET or FORALL statement, as the data cannot be interpreted as a complete date. At least the year portion of a date must be present in the mask to make it accessible to these statements.

Exceptions

CONVERSION	Signaled if <i>date_string</i> does not match <i>pic_string</i> .
OVERFLOW	Raised if the resulting date cannot be expressed as a semantic type date, binary 4.
RANGERROR	Signaled if <i>pic_string</i> contains invalid characters, or if it is not in a valid date format.

Example

- The following rule:
- Inserts the occurrences from the table EMPLOYEE into the table EMPLOYEE_DATE
 - Creates values with the date semantic type, for the field HIRE_DATE

```
COPY_EMPLOYEE ;
-
- -----
- -----+-----
- FORALL EMPLOYEE :                               | 1
-     EMPLOYEE_DATE.* = EMPLOYEE.* ;              |
```

```

-      EMPLOYEE_DATE.HIRE_DATE = $CREATE_DATE( 'YYYY/DD/MM' ,      |
-      EMPLOYEE.HIREDATE);                                     |
-      INSERT EMPLOYEE_DATE;                                     |
-      COMMIT;                                                  |
-      END;                                                      |
-      -----
```

CREATEUSERS

Creates a list of new user IDs and adds them to the TIBCO Object Service Broker system. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type CREATEUSERS (<i>input_table</i> , <i>modeluser</i>) <Enter>
	COMMAND prompt	Type EX CREATEUSERS (<i>input_table</i> , <i>modeluser</i>) <Enter>

Where

<i>input_table</i>	The name of a table containing the user IDs to be added.
<i>modeluser</i>	The name of a model user profile on which the new user IDs are to be based

Prerequisites To use CREATEUSER, a user must satisfy all the following:

- The user must have CREATE_USER capability, that is, be either a level-7 user or a SecAdmin with CREATE_USER capability.
- The table that stores the list of user IDs to be created must have a field called **HURON_USERID** with syntax C and length 8.
- The user should have sufficient security clearance to access the profile of the MODELUSERID.

TIBCO Object Service Broker checks for the above conditions and generates error messages accordingly if they are not fulfilled.

- Usage Notes**
- You can extract the user IDs from elsewhere and put them into the **HURON_USERID** field of the input table (that is, the table can be an import (IMP) table).
 - Create the model user profile using the MANAGE USERS area of the SECURITY MANAGEMENT MAIN MENU, which you access through the SE Security Administration option of the workbench. This enables the login,

print, and other parameters specified for the model user profile to be inherited by the new user IDs. Individual users can further customize their profiles when they log in to the system.

Example Input Table

The following shows a sample input table. It contains a list of the user IDs to be added to the system:

EDITING TABLE	:	NEWUSERS	
COMMAND ==>			SCROLL: P
HURON_USERID			
— -----			
— BJ000			
— BJ002			
— BJ003			
— BJ004			
— BJ005			
— BJ006			
— BJ008			
— BJ009			
— BJ014			
— BJ019			
— BJ020			
— BJ021			
— BJ024			
— BJ033			
PFKEYS: 4=INSERT 16=DELETE 5=FIND NEXT 6=CHG NEXT 18=EXCLUDE 3=SAVE 12=CANCEL			

User Profile

The following shows a sample user profile. All the new user IDs in the sample input table take on the characteristics of this model profile:

Command ==>	

MANAGE USERID: EAG00	(Clearance = 1)

Full Name: Gardiner, Earl	Phone: 305-842-3309 Timezone: 0
CURRENT GROUP: DEVELOPERS	SecAdmin: DBA
Change Password	Logon Parameters
-----	-----
Password:	Logon Restricted from 0 to 0
Verify Password:	Session Menu: DEVELOPER
	Security Group: DEVELOPERS
	Library:

<pre> Print Parameters ----- Destination: HRNPRT9 Form: Class: Y FCB: UCS: Number of Copies: 1 External Writer: File: PFKEYS: 1=HELP ENTER=CHANGE GROUP 5=MEMBERSHIPS 3=SAVE 12=CANCEL 22=DELETE </pre>	<pre> Startup Rule: Action: Browse: Search: Application Parameters ----- Character Set: Fold PFkeys: N Borrower: Y Default Unit: EAG00 TDS Segment: 0 HDS Segment: 1 </pre>
--	--

Message Log

CREATEUSER was run giving the sample table and model user ID from above as input parameters. Here is the confirmation of a successful execution as it appears in the message log:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
CREATING USERIDS:

Userid BJ000 has been created
Userid BJ002 has been created
Userid BJ003 has been created
Userid BJ004 has been created
Userid BJ005 has been created
Userid BJ006 has been created
Userid BJ008 has been created
Userid BJ009 has been created
Userid BJ014 has been created
Userid BJ019 has been created
Userid BJ020 has been created
Userid BJ021 has been created
Userid BJ024 has been created
Userid BJ033 has been created

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT 12=EXIT 13=PRINT

```

CROSSREFSEARCH

Searches the cross reference index of the specified library to answer a query. (C)

Invocation `CALL CROSSREFSEARCH(querystring, querykind, library)`

<i>querystring</i>	<p>A string of one or more names or keywords (the wild card characters asterisk (*) and question mark (?) can be used if they are enclosed in single quotation marks), the AND (&) or OR () logical operators, the NOT operator or the not sign (¬), and the open and closed parentheses symbols</p>
<i>querykind</i>	<p>A string specifying the type of search is to be carried out on the global cross reference. The valid search types are:</p> <p>RULE – Searches for objects that refer to the named rules.</p> <p>TABLE – Searches for objects that refer to the named tables. The search can be based on a whole table, on the field of a table, or on just a field. Enter one of the following for <i>querystring</i>:</p> <ul style="list-style-type: none"> — A table name (for example, EMPLOYEE) — A table name and a field name (for example, EMPLOYEE.DEPTNO) — A field name (for example, *.DEPTNO) <p>SCREEN – Searches for objects that refer to the screens named in the query.</p> <p>REPORT – Searches for objects that refer to the reports named in the query.</p> <p>GLOBALFIELD – Searches for objects that refer to the global fields named in the query.</p> <p>OBJECTSET – Searches for objects that refer to the Object Sets named in the query.</p> <p>LIBRARY – Searches for objects that refer to the libraries named in the query.</p> <p>MENU – Searches for objects that refer to the menus named in the query.</p> <p>SIGNAL – Searches for rules that raise the exceptions named in the query.</p>

<i>library</i>	A string consisting of the name of the TIBCO Object Service Broker library to search
----------------	--

Usage Notes

- [SEARCH](#) is the interactive version of this tool used on the developer's workbench. It also searches the cross reference index.
- Before using CROSSREFSEARCH, the cross reference index must first be built for the library. Refer to [REFMAKER](#) for more information about building the index.
- The local variable MSG must be declared by the calling rule.
- The results of the search are sent to the temporary table @RESULTLIST.

Exceptions

SYNTAX_ERROR	Raised when there is a syntax error in the <i>querystring</i> , the value for <i>querystring</i> is invalid, the value for <i>querykind</i> is invalid, or the <i>library</i> does not exist. In each of these cases, a message is placed in MSG.
---------------------	---

Example

The following rule searches the SYSADMIN library for the FCNKEY table and sends the results to the message log:

```

CROSSREFSEARCH_1;
  LOCAL MSG;
  -----
  -----+-----
  CALL CROSSREFSEARCH('FCNKEYS', 'TABLE', 'SYSADMIN');      | 1
  CALL $RESETPRINT(60, 80, 1 'SCR');                          | 2
  FORALL @RESULTLIST :                                       | 3
    CALL $PRINTLINE(PAD(@RESULTLIST.INDEX, 4, ' ', 'R') ||   |
      PAD(@RESULTLIST.NAME, 18, ' ', 'R') || ' ' ||         |
      @RESULTLIST.TYPE);                                     |
  END;
  -----
  -----

```

Resulting Output

Pressing PF2 displays the following output:

```

-----INFORMATIONAL MESSAGE LOG-----
COMMAND ==>                                                                    SCROLL: P
-----NEW PAGE-----
1          FCNKEY_MSG      RULE

```

2	PROCESS_FCNKEY	RULE
3	RESTORE_ENTRY	RULE
4	SCRDEF_INIT	RULE
5	WRITE_FCNKEYS	RULE

CURSOR_FLDCOL

Returns the relative column number within the field containing the cursor. (F)

Invocation `column = CURSOR_FLDCOL(screen)`

<i>column</i>	On return, contains the column number. Its syntax is B (binary) with length 2.
<i>screen</i>	A character string specifying the name of the screen. Its syntax is C (fixed-length character string) with length 16.

- Usage Notes**
- Relative column numbering begins with 1.
 - A zero (0) is returned if the cursor is not in a valid field.

Example The following rule fills the example screen with data from the example table, displays it, determines the column where the cursor is positioned, and prints it to the message log:

```
CURSOR_FLDCOL_1;
_ LOCAL COLUMN_NUM;
_ -----
_
_ FORALL EMPLOYEE :                               | 1
_     EMPLOYEE_DATA.* = EMPLOYEE.*;               |
_     INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');       |
_     END;                                          |
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_SCR'); | 2
_ INSERT FCNKEY_SPECS('EMPLOYEE_SCR');           | 3
_ UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_SCR:       | 4
_     CALL PROCESS_FCNKEY('EMPLOYEE_SCR');        |
_     END;                                          |
_ COLUMN_NUM = CURSOR_FLDCOL('EMPLOYEE_SCR');     | 5
_ CALL MSGLOG(                                     | 6
_     'THE CURSOR IS POSITIONED ON COLUMN NUMBER ' ||
_     COLUMN_NUM);
_ -----
```

CURSORFIELD

Returns the name of the field where the cursor is located. (F)

Invocation `field_name = CURSORFIELD(screen)`

<code>field_name</code>	On return, contains the name of the field. Its syntax is C (fixed-length character string) with length 16.
-------------------------	--

<code>screen</code>	The name of the screen. Its syntax is C with length 16.
---------------------	---

Usage Notes An empty string is returned if the cursor is not positioned on a field on the specified screen.

Example This rule fills the example screen with data from the example table, displays it, determines the name of the current field, and prints it to the message log:

```

CURSORFIELD_1;
_ LOCAL FIELD_NAME;
_ -----
_ -----+-----
_ FORALL EMPLOYEE :                               | 1
_     EMPLOYEE_DATA.* = EMPLOYEE.*;
_     INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');
_     END;
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_SCR'); | 2
_ INSERT FCNKEY_SPECS('EMPLOYEE_SCR');              | 3
_ UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_SCR:          | 4
_     CALL PROCESS_FCNKEY('EMPLOYEE_SCR');
_     END;
_ FIELD_NAME = CURSORFIELD('EMPLOYEE_SCR');          | 5
_ CALL MSGLOG(                                       | 6
_     'THE CURSOR IS POSITIONED ON THE SCREEN FIELD ' ||
_     FIELD_NAME);
_ -----

```

Sample Output 1

Executing this rule displays the following output:

Employee Name	Employee#
-----	-----
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033

On your screen, a single highlighted character identifies the cursor position.

Sample Output 2

After the rule ends, press PF2 to display the following output.

-----	INFORMATIONAL MESSAGE LOG	-----
COMMAND ===>		SCROLL ===> P
THE CURSOR IS POSITIONED ON THE SCREEN FIELD LNAME		

CUSOROCC#

Returns the occurrence number within the screen table where the cursor is positioned. (F)

Invocation `occ_num = CUSOROCC#(screen)`

<code>occ_num</code>	On return, contains the occurrence number. Its syntax is B (binary) with length 2.
----------------------	--

<code>screen</code>	A character string and specifies the name of the screen. Its syntax is C (fixed-length character string), with length 16.
---------------------	---

Usage Notes

- A zero (0) is returned if the cursor is not on an occurrence on the screen.
- If the screen table is populated with occurrences and the cursor is positioned on a title row, CUSOROCC# returns an occurrence number of 1. If the screen table is not populated with occurrences and the cursor is positioned on a title row, CUSOROCC# returns an occurrence number of 0.



There are differences between where z/OS and Open Systems applications allow the cursor to be positioned.

Example

The following rule fills the example screen with data from the example table, displays it, determines the number of the occurrence within the table where the cursor is positioned, and prints it to the message log:

```

CUSOROCC#_1;
_  LOCAL OCCURRENCE_NUM;
_
_-----+-----
_  FORALL EMPLOYEE :                               | 1
_      EMPLOYEE_DATA.* = EMPLOYEE.*;
_      INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');
_      END;
_  FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_SCR'); | 2
_  INSERT FCNKEY_SPECS('EMPLOYEE_SCR');              | 3
_  UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_SCR:          | 4
_      CALL PROCESS_FCNKEY('EMPLOYEE_SCR');
_      END;
_  OCCURRENCE_NUM = CUSOROCC#('EMPLOYEE_SCR');      | 5
_  CALL MSGLOG(                                       | 6
_      'THE CURSOR IS POSITIONED ON OCCURRENCE NUMBER ' ||
_      OCCURRENCE_NUM);
_-----+-----

```

Sample Output

Executing this rule displays the following output:

Employee Name	Employee#
-----	-----
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033

The cursor position is indicated by a bold letter.

Message Log

After the rule ends, press PF2 to display the following output:

----- INFORMATIONAL MESSAGE LOG -----	
COMMAND ==>	SCROLL ==> P
THE CURSOR IS POSITIONED ON OCCURRENCE NUMBER 1	

CUSOROCC_VALUE

Returns the value of a particular screen field that is selected by the cursor. (F)

Invocation `selected_value = CUSOROCC_VALUE(screen, scrtbl, scrfld)`

<code>selected_value</code>	On return, contains the value of the field.
<code>screen</code>	A character string specifying the screen name.
<code>scrtbl</code>	A character string specifying the screen table.
<code>scrfld</code>	A character string specifying the screen field.

Example The rules below do the following:

1. Display the EMPLOYEE_SCR screen.
2. Display the EMPLOYEE_INFORM screen when a specific function key is used from the EMPLOYEE_SCR screen.

EMPLOYEE_CURSOR Rule

```

EMPLOYEE_CURSOR;
- -----
- -----+-----
- FORALL EMPLOYEE :                               | 1
-     EMPLOYEE_DATA.* = EMPLOYEE.*;               |
-     INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');       |
-     END;                                          |
- FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_SCR'); | 2
- INSERT FCNKEY_SPECS('EMPLOYEE_SCR');           | 3
- UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_SCR:        | 4
-     CALL PROCESS_FCNKEY('EMPLOYEE_SCR');         |
-     END;                                          |
- -----

```

Sample Output 1

Executing this rule displays the following screen:

Employee Name	Employee#
-----	-----
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033
NELSON	81000
CAREY	81001
CHIU	81003
LYNGBAEK	81014
KINGSON	81019

FCNKEYS: 3=END 14=EXPAND

CURSOR_EXPAND Rule

Positioning the cursor anywhere on the line containing the employee number and pressing PF14 (Expand) invokes the CURSOR_EXPAND rule. This rule uses CURSOROCC_VALUE:

```

CURSOR_EXPAND;
  _ LOCAL SELECTED_VALUE;
  _ -----
  _ -----+-----
  _ SELECTED_VALUE = CURSOROCC_VALUE('EMPLOYEE_SCR',      | 1
  _   'EMPLOYEE_DATA', 'EMPNO');
  _ GET EMPLOYEE WHERE EMPNO = SELECTED_VALUE;              | 2
  _ CALL DELETESCREENDATA('EMPLOYEE_INFORM');              | 3
  _ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_INFORM');  | 4
  _ INSERT FCNKEY_SPECS('EMPLOYEE_INFORM');                | 5
  _ EMPLOYEE_INFO.* = EMPLOYEE.*;                          | 6
  _ INSERT EMPLOYEE_INFO('EMPLOYEE_INFORM');               | 7
  _ UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_INFORM:            | 8
  _   CALL PROCESS_FCNKEY('EMPLOYEE_INFORM');
  _   END;
  _ -----

```

Sample Output 2

Positioning the cursor on the line of EMPNO 80003 displays
EMPLOYEE_INFORM as follows:

Employee Information	
EMPNO:	80003
LNAME:	CHANG
POSITION:	Assoc Analyst
MGR# :	83020
DEPTNO:	10
SALARY:	589.00

CURSORTABLE

Returns the name of the screen table where the cursor is positioned. (F)

Invocation `table_name = CURSORTABLE(screen)`

<code>table_name</code>	On return, contains the name of the screen table. Its syntax is C (fixed-length character string) with length 16.
-------------------------	---

<code>screen</code>	A character string specifying the name of the screen. Its syntax is C with length 16.
---------------------	---

Usage Notes An empty string is returned if the cursor is not positioned in a screen table.



There are differences between where z/OS and Open Systems applications allow the cursor to be positioned.

Example The following rule fills the example screen with data from the example table, displays it, determines the name of the screen table where the cursor is positioned, and prints it to the message log:

```

CURSORTABLE_1;
_ LOCAL TABLE_NAME;
_ -----
_
_ FORALL EMPLOYEE :                               | 1
_     EMPLOYEE_DATA.* = EMPLOYEE.*;
_     INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');
_     END;
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('EMPLOYEE_SCR'); | 2
_ INSERT FCNKEY_SPECS('EMPLOYEE_SCR');              | 3
_ UNTIL EXIT_DISPLAY DISPLAY EMPLOYEE_SCR:          | 4
_     CALL PROCESS_FCNKEY('EMPLOYEE_SCR');
_     END;
_ TABLE_NAME = CURSORTABLE('EMPLOYEE_SCR');        | 5
_ CALL MSGLOG(                                       | 6
_     'THE CURSOR IS POSITIONED ON THE SCREEN TABLE ' ||
_     TABLE_NAME);
_ -----

```

Sample Output

Executing this rule displays the following output:

Employee Name	Employee#
-----	-----
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024

Message Log

The cursor position is indicated by a bold letter. After the rule ends, press PF2 to display the following output:

-----	INFORMATIONAL MESSAGE LOG	-----
COMMAND ==>		SCROLL ==> P
THE CURSOR IS POSITIONED ON THE SCREEN TABLE EMPLOYEE_DATA		

DASTATS

Returns statistical data collected by the Data Object Broker for an individual segment. (F)

Invocation `string = DASTATS(segment)`

<code>string</code>	On return contains the statistical data.
<code><i>segment</i></code>	A valid segment number.

Usage Notes A sample rule S6BSEGSTATS is supplied in library SAMPLE. This rule retrieves the data from the Data Object Broker and uses MAP and TEM table functionality to display the data using the Table Browser. The associated tables are S6BSEGSTAT_MAP, S6BSEGSTAT_TEM and S6BSEGSTATN_MAP.

DATACOM

Displays a menu to manage the definition of CA-Datacom data. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type DATACOM <Enter>
	COMMAND prompt	Type EX DATACOM <Enter>

Executing DATACOM displays the following screen:

CA-Datacom Table Management Facility

- _ Generate metadata tables from CA-Datacom definitions
- _ Display table/serverid information
- _ Extract CA-Datacom information
- _ Generate DB2 CREATE TABLE from CA-Datacom definitions

PFKEYS: 2=LOGS 3=EXIT 12=EXIT

\$DATE_DEFAULT

Returns the default date format used by the installation. (F)

Invocation `string = $DATE_DEFAULT`

<code>string</code>	On return, contains the default date format.
---------------------	--

Usage Notes The valid date formats are listed here:

Format Code	Meaning	Example
W	One- or two-digit week # (of year), with no leading 0.	1 or 25
WW	Two-digit week # (of year).	01
WWW	Abbreviated weekday.	Thur
WWWW	Full weekday.	Thursday
M	Numeric month, with no leading 0 (1 or 2 digits).	3 or 10
MM	Numeric month (2 digits).	02
MMM	Abbreviated month.	Mar
MMMM	Full month.	March
D	Day in month, with no leading 0 (1 or 2 digits).	5 or 14
DD	Day in month (2 digits).	02
DDD	Day in year (3 digits).	074
YY	Last two digits in a year.	98
YYYY	Full year.	2000
QQ	Two-character quarter.	2Q
JD	Julian date.	98.074
CC	Two-digit century.	19

- A separator character can be any one of the following:
/ \ ; : , . * - blank
- If no date format is specified, the installation default date format is used.
- A week is defined to begin on a Monday and end on the following Sunday. However, January 1st always begins week one, regardless of where it falls in the week, and week two starts on the following Monday.
- The default century for two-digit year values depends on the setting of YYCENTURYRANGE.

See *TIBCO Object Service Broker for z/OS Installing and Operating* for information on YYCENTURYRANGE. See *TIBCO Object Service Broker Parameters* for information on YYCENTURYRANGE in Open Systems.



You can specify just a portion of a date field within your mask (for example, entering only MMMM displays the month). Partial date occurrences cannot be accessed using a GET or FORALL statement, as the data cannot be interpreted as a complete date. At least the year portion of a date must be present in the mask to make it accessible to these statements.

Example The following rule returns the date default format for the installation to the message line on the workbench:

```
DATE_DEFAULT;  
-----  
-----+-----  
CALL  ENDMSG($DATE_DEFAULT);                | 1  
-----
```

It returns: YYYY-MM-DD

\$DATE_LENGTH

Returns the maximum string length of a given date format. (F)

Invocation `length = $DATE_LENGTH(pic_string)`

<code>length</code>	On return, contains the maximum length, in characters, of the date format.
---------------------	--

<code><i>pic_string</i></code>	The given date format. Its syntax is C (fixed-length character string), UN (Unicode), or V (variable-length character string).
--------------------------------	--

Usage Notes The valid display masks are listed here:

Format Code	Meaning	Example
W	One- or two-digit week # (of year), with no leading 0.	1 or 25
WW	Two-digit week # (of year).	01
WWW	Abbreviated weekday.	Thur
WWWW	Full weekday.	Thursday
M	Numeric month, with no leading 0 (1 or 2 digits).	3 or 10
MM	Numeric month (2 digits).	02
MMM	Abbreviated month.	Mar
MMMM	Full month.	March
D	Day in month, with no leading 0 (1 or 2 digits).	5 or 14
DD	Day in month (2 digits).	02
DDD	Day in year (3 digits).	074
YY	Last two digits in a year.	98
YYYY	Full year.	2000
QQ	Two-character quarter.	2Q

Format Code	Meaning	Example
JD	Julian date.	98.074
CC	Two-digit century.	19

- A separator character can be any one of the following:
/ \ ; : , . * - blank
- If no date format is specified, the installation default date format is used.
- A week is defined to begin on a Monday and end on the following Sunday. However, January 1st always begins week one, regardless of where it falls in the week, and week two starts on the following Monday.



You can specify just a portion of a date field within your mask (for example, entering only MMMM displays the month). Partial date occurrences cannot be accessed using a GET or FORALL statement, as the data cannot be interpreted as a complete date. At least the year portion of a date must be present in the mask to make it accessible to these statements.

Exceptions

RANGERROR	Signalled if <i>pic_string</i> is not a valid date format.
-----------	--

Example

The following rule returns the maximum length of the date format, which is given as an argument, and displays the result on the message line of the workbench:

```
DATE_LENGTH;  
-----  
+-----  
CALL ENDMMSG('THE LENGTH OF THE DATE FORMAT IS ' ||           | 1  
$DATE_LENGTH('MMMM,D,YYYY')) ;                               |  
-----
```

It returns: THE LENGTH OF THE DATE FORMAT IS 17

Basis for the Value Returned

This length is based on the following:

- The longest value for MMMM is 9 (for example, September).
- The longest value for D is 2.

- The longest value for YYYY is 4.
- The separators have a length of 2.

These values total up to 17.

\$DATE_PIC

Converts a value of semantic type date to a semantic type string. (F)

Invocation `date_string = $DATE_PIC(pic_string, date)`

<i>date_string</i>	Contains the resulting string. Its syntax is V (variable-length character string).\
<i>pic_string</i>	The format of the resulting string. Its syntax is C (fixed-length character string), UN (Unicode), or V.
<i>date</i>	The date value that is to be converted to a string. Its syntax is B (binary).

- Usage Notes**
- Only the date components explicitly mentioned in *pic_string* are converted.
 - If *pic_string* is null the installation default is used.
 - The following display mask components are valid for semantic type date fields:

Format Code	Meaning	Example
W	One- or two-digit week # (of year), with no leading 0.	1 or 25
WW	Two-digit week # (of year).	01
WWW	Abbreviated weekday.	Thur
WWWW	Full weekday.	Thursday
M	Numeric month, with no leading 0 (1 or 2 digits).	3 or 10
MM	Numeric month (2 digits).	02
MMM	Abbreviated month.	Mar
MMMM	Full month.	March
D	Day in month, with no leading 0 (1 or 2 digits).	5 or 14
DD	Day in month (2 digits).	02

Format Code	Meaning	Example
DDD	Day in year (3 digits).	074
YY	Last two digits in a year.	98
YYYY	Full year.	2000
QQ	Two-character quarter.	2Q
JD	Julian date.	98.074
CC	Two-digit century.	19

- A separator character can be any one of the following:
/ \ ; : , . * - blank
- If no date format is specified, the installation default date format is used.
- A week is defined to begin on a Monday and end on the following Sunday. However, January 1 always begins week one, regardless of where it falls in the week, and week two starts on the following Monday.



You can specify just a portion of a date field within your mask (for example, entering only MMMM displays the month). Partial date occurrences cannot be accessed using a GET or FORALL statement, as the data cannot be interpreted as a complete date. At least the year portion of a date must be present in the mask to make it accessible to these statements.

Exceptions

CONVERSION	Signalled if <i>pic_string</i> is not a string, or if date is not a date.
RANGERROR	Signalled if <i>pic_string</i> contains invalid characters, or if it is not in a valid date format.

Example

- The following rule:
1. Creates values with the string semantic type, for the field BIRTHDATE, in the format YY/MM/DD.
 2. Replaces the occurrences in the EMPLOYEE table.

```
INSERT_BIRTHDATE;  
-----
```

```
-----+-----
-  FORALL EMPLOYEE_DATE :                               | 1
-    GET EMPLOYEE WHERE EMPNO = EMPLOYEE_DATE.EMPNO;    |
-    EMPLOYEE.BIRTHDATE = $DATE_PIC('YY/MM/DD',         |
-      EMPLOYEE_DATE.BIRTHDATE);                        |
-    REPLACE EMPLOYEE;                                  |
-    COMMIT;                                             |
-    END;                                                |
-  -----
```

The format of the BIRTHDATE field in the table EMPLOYEE_DATE is illustrated here. It appears in the default date format for the installation.

EMPNO	BIRTHDATE
80000	1960-06-13
80002	1972-09-11
80003	1962-12-30
80004	1964-04-22
80014	1970-09-21

It is converted to the format illustrated below, in the EMPLOYEE table:

EMPNO	BIRTHDATE
80000	60/06/13
80002	72/09/11
80003	62/12/30
80004	64/04/22
80014	70/09/21

\$DATE_REF

Adds or subtracts a given number of days to or from a reference date, and converts the number of days returned to units of a day, a week, a month, or a year. (F)

Invocation value = \$DATE_REF(*component*, *duration*, *date*, *round*)

value	On return, contains the number of units.
component	<div>The type of component to be extracted. It is one of the following:<ul style="list-style-type: none">D - DayW - WeekM - MonthY - YearIts syntax is C (fixed-length character string), with length 1.</div>
duration	An integer specifying the number of days to be added or subtracted. Its syntax is B (binary) with length 4.
date	The date used as the reference point
round	<div>One of the following:<ul style="list-style-type: none">Y - Round the answer.N - Truncate the answer.</div>

Usage Notes • The valid date formats are listed here:

Format Code	Meaning	Example
W	One- or two-digit week # (of year), with no leading 0.	1 or 25
WW	Two-digit week # (of year).	01
WWW	Abbreviated weekday.	Thur
WWWW	Full weekday.	Thursday

Format Code	Meaning	Example
M	Numeric month, with no leading 0 (1 or 2 digits).	3 or 10
MM	Numeric month (2 digits).	02
MMM	Abbreviated month.	Mar
MMMM	Full month.	March
D	Day in month, with no leading 0 (1 or 2 digits).	5 or 14
DD	Day in month (2 digits).	02
DDD	Day in year (3 digits).	074
YY	Last two digits in a year.	98
YYYY	Full year.	2000
QQ	Two-character quarter.	2Q
JD	Julian date.	98.074
CC	Two-digit century.	19

- A separator character can be any one of the following:
/ \ ; : , . * - blank
- If no date format is specified, the installation default date format is used.
- A week is defined to begin on a Monday and end on the following Sunday. However, January 1st always begins week one, regardless of where it falls in the week, and week two starts on the following Monday.
- Treat with caution values returned for dates prior to the adoption of the Gregorian calendar in 1582 or for dates in the very far future that could be subject to calendar adjustments (for example, it is not yet clear if the year 4000 is a leap year). \$DATE_REF accurately returns values for dates 200 years before or after the present date.
- The default century for two-digit year values depends on the setting of YYCENTURYRANGE.

See *TIBCO Object Service Broker for z/OS Installing and Operating* for information on YYCENTURYRANGE. See *TIBCO Object Service Broker Parameters* for information on YYCENTURYRANGE in Open Systems.



You can specify just a portion of a date field within your mask (for example, entering only MMMM displays the month). Partial date occurrences cannot be accessed using a GET or FORALL statement, as the data cannot be interpreted as a complete date. At least the year portion of a date must be present in the mask to make it accessible to these statements.

Exceptions

CONVERSION	Signalled if <i>component</i> is not a number or if date is not a date.
OVERFLOW	Raised if the value returned is too large for the length of the syntax.
RANGERROR	Signalled if the value given for <i>component</i> is not one of D, W, M, or Y, or if the value given for <i>duration</i> is not valid for <i>component</i> .

Example

The following rule references the date 98/05/03, subtracts 100 days from it, and returns the number of months, truncated to the nearest month. It displays the result on the message line of the workbench.

```
DATE_REF;
-
- -----
- -----+-----
- CALL ENDMMSG('-100 DAYS IS ' || $DATE_REF('M', - 100,      | 1
-   $CREATE_DATE('YY/MM/DD', '98/05/03'), 'N') ||          |
-   ' MONTHS FROM THE REFERENCE DATE');                    |
- -----
```

It returns:

-100 DAYS is -3 MONTHS FROM THE REFERENCE DATE

DBMAINTLVL

Displays the maintenance level of your TIBCO Object Service Broker database, including any database PTFs applied beyond the maintenance level. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type DBMAINTLVL <Enter>
	COMMAND prompt	Type EX DBMAINTLVL <Enter>

Usage Notes The database maintenance level appears on the message line at the bottom of the workbench screen. Press PF2 to see a list of the PTFs applied beyond that level, if any.

Exceptions None.

Example The following example shows the workbench screen after execution of DBMAINTLVL:

```

H U R O N          USR40      TEST: N BROWSE: N  12:48 AM  THURSDAY OCT 25 2001

ER edit rule       ==>
EX execute rule    ==> DBMAINTLVL
DB debug rule      ==>
BR browse table    ==>
ED edit table      ==>

OS object set      ==>
DS define screen   ==>
DR define report   ==>
DT define table    ==>
DL define library  ==>
GR generate rpt    ==>

COMMAND ==>  _
execute rule:  EX DBMAINTLVL

PFKEYS: 2=LOGS 3=EXIT 12=EXIT

3:45:32 Database maintenance level 3.2.00 - PF2 for additional PTFs applied

```


On pressing PF2, you see information similar to the following:

```
----- INFORMATION LOG -----  
COMMAND ==>                                SCROLL ==> P  
Database maintenance level 3.2.00  
  8946      3.2.01  
  8947      3.2.01  
  8948      3.2.01  
  8949      3.2.01  
  8950      3.2.01  
  8951      3.2.01  
  8952      3.2.01  
  9036      3.2.01  
  9037      3.2.01  
  9038      3.2.01  
  9039      3.2.01  
  9093      3.2.01
```

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT FIND 12=EXIT 13=PRINT 9=RECALL

DEBUG

Invokes the interactive Rule Debugger. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DB debug rule option	Type <i>rulename</i> <Enter>
	EX execute rule option	Type EDEBUG (<i>rulename</i>) <Enter>
	COMMAND prompt	Type DB <i>rulename</i> <Enter>
From a rule		Type CALL DEBUG (<i>rulename</i>)
In the Rule Editor or in Execute Rule	Object manager screen	G line command

Where:

<i>rulename</i>	A string containing the name of the rule to debug.
-----------------	--

Usage

If you supply for rulename...	When using...	Pressing Enter displays:
A value	All methods	A screen that you use to debug the rule
No value	DB debug rule or Command line	A listing of rules defined in your local library. You can select the rule that you require from this list.
	EX execute rule	A screen prompting for a rule name

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Rule Debugger.

TIBCO Object Service Broker Programming in Rules for information on the Rule Debugger.

DEFINE_LIBRARY

Defines a new library, displays a list of the rules in a library, or displays a list of the rules libraries. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DL define library option	Type <i>libraryname</i> <Enter>
	EX Execute Rule option	Type DEFINE_LIBRARY (<i>libraryname</i>) <Enter>
	COMMAND prompt	Type EX DEFINE_LIBRARY (<i>libraryname</i>) <Enter>

Where:

<i>libraryname</i>	The name of a rules library
--------------------	-----------------------------

Usage

If you supply this for libraryname...	Pressing Enter displays...
A new value	A screen that you use to define a library
An existing value	A listing of the rules in the specified library
No value	A list of libraries defined in your TIBCO Object Service Broker database You can select the library name that you require from this list.

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Library Definer.

TIBCO Object Service Broker Programming in Rules for additional information on the use of DEFINE_LIBRARY and defining libraries.

DEFINE_MENU

Creates and modifies menus and login screens used within TIBCO Object Service Broker user-defined applications. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type DEFINE_MENU(<i>menu</i>) <Enter>
	COMMAND prompt	Type EX DEFINE_MENU <Enter>

Where:

<i>menu</i>	The name of the menu.
-------------	-----------------------

See Also *TIBCO Object Service Broker Defining Screens and Menus* for information on DEFINE_MENU.

DEFINE_OBJECTSET

Defines a set of objects or modifies an existing set. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	OS object set option	Type <i>objsetname</i> <Enter>
	EX Execute Rule option	Type DEFINE_OBJECTSET (<i>objsetname</i>) <Enter>
	COMMAND prompt	Type OS <i>objsetname</i> <Enter>

Where:

objsetname The name of the object set to be defined.

Usage Notes You can specify security later only for those objects that are identified explicitly as components of the object set.

Steps for Defining Object Sets

Overview

To define an object set, complete the following tasks:

Task	Description	Optional
A	Access the Object Set Definition tool.	N
B	Specify a valid object type.	N
C	Specify a value for the Name field.	N
D	Specify additional object types.	Y
E	Specify access permissions.	Y
F	View the full object definition.	Y

Task	Description	Optional
G	Print the object set definition.	Y
H	Save the definition.	N

The following sections provide a detailed description for defining object sets.

Task A Access the Object Set Definition tool

To access the Object Set Definition tool, use one of the following methods. The *objsetname* is the name of the new or existing object set you are defining.

- Type `define_objectset(objsetname)` at the EX execute rule option on the workbench and press Enter.
- Type an *objsetname* at the OS object set option on the workbench and press Enter.
- Type `os objsetname` at the command prompt and press Enter.

Example

The following is an example of the Object Definition screen for the object set SAMPLAPP:

DEFINE OBJECTSET: SAMPLAPP				UNIT: SAMPLAPP		
COMMAND ==>				Scroll: P		
OBJECT TYPE:						
Name	Type	Unit	Author	Created	Modifier	Modified
-----	-----	-----	-----	-----	-----	-----
SAMPLAPP	LIBRARY	SAMPLAPP	USR40	97.339	USR40	98.094
@EXPENSE_ITEMS	REPORT					
@EXPENSE_SUMMARY	REPORT					
@EXPENSE_ITEMS	SCREEN					
@EXPENSE_SELECT	SCREEN					
@CURRENCIES	TABLE					
@DETAIL_DATE_BRK	TABLE					
@DETAIL_FINALBRK	TABLE					
@EXCHANGE	TABLE					
@EXCHANGE_RATES	TABLE					
@EXCHANGE_RATES\$	TABLE					
@EXPENSE_BODY	TABLE					
@EXPENSE_EMP	TABLE					
@EXPENSE_FOOTER	TABLE					
@EXPENSE_HEADER	TABLE					
PFKEYS: 1=HELP 3=SAVE 12=CANCEL 2=DOC 22=DEL 4=SAVE & SEC 9=DEFINE 21=MSGLOG						

Task B Specify a valid object type

In the **OBJECT TYPE** field, specify a valid object type by using one of the following methods:

- Type in a value directly and press Enter.
- Position your cursor on the **OBJECT TYPE** field and then press PF1. This displays a listing of values from which you can select a value and press PF3 to return to the Object Set Definition screen. Press Enter when the value appears in the **Object Type** field.
- Press PF5 without specifying an object type. The Object Selection screen appears and you can select from a list of all types of objects. Refer to [Selecting Objects for Object Set Definitions on page 161](#) for more information.

Task C Specify a value for the Name field

To specify a value for the **Name** field, use one of the following methods:

- In the **Name** field, type the name of an existing object of the type specified in the **OBJECT TYPE** field and press Enter.
- In the **Name** field, type the name of a new object of the type specified in the **OBJECT TYPE** field.

You can use one of the following options to define the new object:

PF3 Save the object set definition and return to the workbench. Access the appropriate definer for the object and define the object.

PF9 Invoke the Definer for the object type specified. Define the object as required. When you press PF3 or PF12 from the Definer screen, you are returned to the Object Set Definer.

- Press PF5 to display a screen that you can use to select the objects that you require. Refer to [Selecting Objects for Object Set Definitions on page 161](#) for more information.



If the object is a table, refer to [Using Tables in Object Set Definitions on page 164](#) for more information on adding tables to the object set definition.

Task D Specify additional object types

If you want to specify additional object types, repeat the above tasks. A sub-screen appears for each object type that you are defining.

Task E Specify access permissions

If you want to specify access permissions to any of the objects, complete the following steps:

- 1. Press PF4 from the Object Definition screen.
This saves the definition of the object set and invokes the Manage Permissions screen for TIBCO Object Service Broker security.
- 2. Use the primary command **FETCH** to retrieve any or all of the securable objects within an object set.
For more information on the **FETCH** command and access permissions, refer to *TIBCO Object Service Broker Managing Security*.

Task F View the full object definition

To view the full definition of your object set, leave the **Object Type** field on the Object Definition screen blank and press Enter.

Task G Print the object set definition

The following is an example of the screen used for printing object set definitions:

```
DEFINE OBJECTSET: DOCEXAMPLE
COMMAND ==>

ENTER ARGUMENTS FOR PRINTING:

LIBRARY      ==>

ENVIRONMENT  ==>

LOCATION      ==>

PARENT ONLY  ==> Y
```

To print the object set definition, complete the following steps:

- 1. Press PF13 from the Object Set Definition screen.
The Print screen appears.
- 2. Indicate the name of the rules library.

Provide a library name only if the object set contains rules.

3. Leave the ENVIRONMENT field blank.

This field is currently not in use.

4. Specify the node name.

Provide a node name in the **LOCATION** field only if the objects are located on a node other than your home node. If this field is left blank, it defaults to your home node.

5. Indicate if all the objects should be printed.

If PARENT ONLY is Y, only the definition of the object set (parent) is printed. If it is N, the object set and all the objects it contains (children) have their definitions printed.

Task H Save the definition

Press PF3 from the Object Definition screen to save your definition and exit to the workbench.

Selecting Objects for Object Set Definitions

Overview

To select objects for object set definitions or modify a series of objects that are selected by default, complete the following tasks:

- [Invoking the Object Selection screen](#)
- [Narrowing the selection scope](#)
- [Selecting objects](#)

Task A Invoking the Object Selection screen

To invoke the Object Selection screen, complete the following steps:

1. Press PF5 from the Object Definition screen.

The Object Selection screen appears. The screen displays a series of objects that are selected by default based on the values in the **UNIT** field and the **OBJECT TYPE** field provided in the Object Set Definition screen. The **OBJECT TYPE** field could be blank if you pressed PF5 before specifying an object type.

The following is an example of the Object Selection screen:

```

                                O b j e c t   S e l e c t i o n                                Scroll: P
COMMAND ==>
Location:                               Select All: N
Library (for RULES):                     Deselect All: N
Presentation Environment:                 List Children: N
                                         Show selection specs: Y
===== Selection Specification =====
Attr   Op      Value
-----
NAME   _____ AND      unspecified
TYPE   =  SCREEN _____ AND  attributes will
UNIT   =  SAMPLAPP _____ AND  be ignored
AUTHOR _____
=====
Name      Type      Library      Environment      Unit
-----
-  @EXPENSE_ITEMS  SCREEN
-  @EXPENSE_SELECT  SCREEN  SAMPLAPP  SAMPLAPP

```

- 2. Modify the listing as required.
You can modify the listing by excluding fields that are not required when you are searching through the list of objects. This helps in object selection, since many lists are long and you must narrow down your selection requirements.

Task B Narrowing the selection scope

If you want to narrow the selection scope, you can use two different portions of the screen:

- Specify values in the header portion.
- Specify selection specifications.

Specifying Values in the Header Portion

You can use the **Location**, **Library**, and **Presentation Environment** fields in the header portion of the screen to narrow the selection scope for objects.

- Specifying a Location
In the **Location** field, you can specify a node where the selection criteria are to be applied. If you do not specify a value in this field, your home node is used.
- Specifying a Library
If your selection list contains rules or if the **OBJECT TYPE** field is empty, you can specify the name of the rules library to be searched. If the **OBJECT TYPE** field is empty, a library should be specified to ensure that all object types are included in the list. Press PF1 to display a list of valid values from which you can select.

Specifying a Presentation Environment	This field is currently not in use.
Listing Child Objects	If some objects have child objects associated with them, you can specify if you want to list all the child objects that compose the parent objects.

Specifying Selection Specifications

The middle section of the screen can be used to select items to be included in the definition or to narrow the selection list. You can use more than one type of selection criteria for each object type and you can specify multiple object types within one session. For a list of valid values for each of these fields, position your cursor on the field and press PF1.

Do the following when specifying selection criteria:

When specifying	In the Op Field, specify	In the Value Field, specify
Name	The logical operator to be used	The name of the object
Type		The name of the object type
Unit	The logical operator to be used	The name of the unit associated with the object
Author	The logical operator to be used	The name of the author of the object

Considerations

Note the following about the object type selection criteria:

- If you do not supply an object type, you must specify a value in at least one of the other selection fields.
- If you specify only an object type and no further selection values, a listing of the items for the object type defined in your TIBCO Object Service Broker database appears for further selection.

Task C Selecting objects

There are two ways you can select objects:

- Select objects using the header portion of the screen.
- Select objects from the list of objects at the bottom of the screen.

Select Objects from Header Portion

Using the **Select All** field, you can specify whether all the items displayed, based on selection criteria, should be copied into the Object Definition screen.

Deselecting All
Objects

Use the **Deselect All** field if you want to deselect the items that you selected.

Select Objects from the List of Objects

After you specify the selection criteria and press Enter, the selected items appear in the bottom portion of the screen. You can select the objects displayed in this section by typing an **S** in the **line command** fields beside the objects.

After selecting the items that you require, press PF3 to save the selections and return to the Object Set Definition screen. The items are listed in the appropriate type sub-screen within the Object Set Definition screen.

To un-select items that you selected, use the **Deselect All** field.

Using Tables in Object Set Definitions

Adding Tables to the Object Set Definition

When adding table objects, you are prompted to supply a value for the following:

Defn	Y – Include the definition of the table.
	N – Do not include the definition of the table.
Data	Y – Include the data in the table.
	N – Do not include data in the table.

At least one of these fields must contain a Y.

Security Requirements for a Table

If security is to be specified for temporary (TEM) or TDS tables, the following values must be set to Y:

Temporary (TEM).	Defn
TDS unparameterized.	Defn and Data
TDS parameterized and all instances.	Defn and Data

TDS parameterized and specific instances.

Data

Specifying Table Instances

If the object is a parameterized table, press PF6 to display one of two screens for selecting table instances to be part of your definition. The screen that appears depends on whether the table has a parameter value table (table of type PRM) associated with it.

Screen for Table with No PRM Table

The Specifying Table Instances screen appears when the table does not have a parameter (PRM) table associated with it. Use this screen to individually specify all the table instances that are to be part of the object set definition. If no values are specified, the default is all instances of the table.

```
-----
SPECIFY INSTANCE OF TABLE:  @EXPENSES
-----

Table Parameters:              REGION =

( ALL DATA: Y )
    < specify all parameter values or set ALL DATA = Y for whole table >
```

Screen for Table with a PRM Table

The Parameters for table screen appears when the table has a parameter (PRM) table associated with it. Use this screen to select all the table instances that are to be part of the object set definition. If no values are specified, the default is all instances of the table.

```
Parameters for table EMPLOYEES                                Scroll: P
COMMAND ==>                                                    Select All: N
Location:                                                       Deselect All: N
                                                                Show selection specs: Y
===== Selection Specification =====
      Selection: REGION LIKE '*'
AND      Op      Value
      -----
      REGION
=====
      Region
      -----
_ A
_ CANADA
_ CENTCANADA
```

- _ MEXICO
- _ MIDWEST
- _ SOUTHWEST

PFKEYS: ENTER=UPDATE 3=SAVE 12=CANCEL

Example The following example shows defined sub-screens for objects of type screen, table, and rule as well as the final screen with all the objects listed.

Subscreen for Screen Objects

DEFINE OBJECTSET: UPDATE_EMPLOYEES				UNIT: USR40		
COMMAND ==>				Scroll: P		
OBJECT TYPE: SCREEN						
Name	Type	Unit	Author	Created	Modifier	Modified
-----	-----	-----	-----	-----	-----	-----
DELETE_EMPLOYEE	SCREEN	USR40	USR40	97.312	USR50	98.175
NEW_EMPLOYEE	SCREEN	USR40	USR40	97.310	USR50	98.173

Subscreen for Table Objects

DEFINE OBJECTSET: UPDATE_EMPLOYEES					UNIT: USR40				
COMMAND ==>					Scroll: P				
OBJECT TYPE: TABLE									
Name	Parms	Defn	Data	Type	Unit	Author	Created	Modifier	Modified
-----	-	-	-	-	-----	-----	-----	-----	-----
\$EMPLOYEES	N	Y	Y	PRM	USR60	USR60	96.350	USR50	96.35
EMPLOYEES	Y	Y	Y	TDS	ACC	USR40	97.110	USR40	98.14
MANAGER	N	Y	Y	TDS	DOC01	USR40	97.290	USB80	98.11

PFKEYS: 1=HELP 3=SAVE 12=CANCEL 2=DOC 22=DELETE 4=SAVE & SECURITY 9=DEFINE

Subscreen for Rule Objects

DEFINE OBJECTSET: UPDATE_EMPLOYEES
COMMAND ==>

UNIT: USR40
Scroll: P

OBJECT TYPE: RULE

Name	Unit	Author	Created	Modifier	Modified
DEL_EMP	USR40	USR40	96.312	USR40	97.175
DELETE_EMPLOYEE	USR40	USR40	96.312	USR40	97.165
NEW_EMPLOYEE	USR40	USR40	96.310	USL12	97.175
PRINT_EMP	USR40	USR40	95.317	USR40	96.310
SAVE_EMP	USR40	USR40	96.310	USR40	97.160

Defined Object Set

DEFINE OBJECTSET: UPDATE_EMPLOYEES
COMMAND ==>

UNIT: USR40
Scroll: P

OBJECT TYPE:

Name	Type	Unit	Author	Created	Modifier	Modified
DEL_EMP	RULE	USR40	USR40	96.312	USR40	98.145
DELETE_EMPLOYEE	RULE	USR40	USR40	96.312	USR40	97.165
NEW_EMPLOYEE	RULE	USR40	USR40	96.310	USL12	97.175
PRINT_EMP	RULE	USR40	USR40	96.317	USR40	98.115
SAVE_EMP	RULE	USR40	USR40	96.310	USR40	97.160
DELETE_EMPLOYEE	SCREEN	USR40	USR40	96.312	USR40	97.175
NEW_EMPLOYEE	SCREEN	USR40	USR40	96.310	USR40	97.173
\$EMPLOYEES	TABLE	ACE60	USR40	97.350	USG60	97.350
EMPLOYEES	TABLE	ACC	USR40	95.110	USR40	96.149
MANAGER	TABLE	DOC01	USR40	96.290	USB80	97.110

DEFINE_OBJLIST

Defines, for a table, an object list to appear using the Object Manager or modifies an existing object list definition. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type DEFINE_OBJLIST(<i>table</i>) <Enter>
	COMMAND prompt	Type EX DEFINE_OBJLIST(<i>table</i>) <Enter>

Where:

<i>table</i>	The name of the table containing the list of objects to appear.
--------------	---

Usage Notes

- If you supply a new value for *table*, executing DEFINE_OBJLIST displays an empty Define Object List screen.
- If you supply an existing value for *table*, executing DEFINE_OBJLIST displays a screen with the current characteristics of the table.
- In both cases, more fields are available than can appear at one time on the screen. To view additional fields, press PF11 to move one screen to the right. This tool comprises four full lateral screens of fields.
- DEFINE_OBJLIST defines rules, specifies how they should be executed, and arranges up to two lines of titling. This information is then applied against the table given as an argument to [OBJECT_MGMT](#) (or its interactive version [OBJECTMGR](#), as both of these tools use DEFINE_OBJLIST when they are invoked).

Initial Screen

This initial screen appears when you execute DEFINE_OBJLIST:

```
Object Commands for Table: EMPLOYEES_ODPARM
Command ==>
```

Command	Description	Rule	New Trans	Browse	Search	Prompt
-	-----	-----	-	-	-	-


```
Titles to Appear Above and Below the Command Line
-----
try this
try this again
```

PFKEYS: 12=CANCEL 16=DELLINE 22=DELETE 3=SAVE 4=ADDLINE

Top Segment

The top segment of the screen contains the name of the table and the primary command line. By changing the name of a previously defined object list you can use an existing definition as a template.

The following primary commands are supported:

SAVE	Save the definition and exit.
CANCEL	Cancel the definition and exit.
ADDLINE	Add a line for input. The line is added after the line where your cursor is positioned.
DELLINE	Delete a line of input. The line the cursor is positioned on is deleted.

Middle Segment

You use the middle segment to define the line commands. To see all the fields, position your cursor within this segment and press PF11 to scroll right. It contains the following fields:

Command	The letter to be associated with the output command. For example, type S for select.
Description	A short description of the command. It appears at the bottom of the object manager screen, for example, S for select.

Rule	The name of a rule. This rule is processed when the line command is invoked, for example, STE.
New Trans	Enter one of Y (the rule is to be executed) or N (the rule is to be called).
Browse	<p>If the rule is executed, type Y (run in browse mode) or N (run in update mode).</p> <p>If the rule is called, this field has no effect; this field is ignored if New Trans is N. In the standard session manager, the mode is taken from the Browse field at the top of the screen, while in all other session managers, the mode is also taken from the menu.</p>
Search	<p>If the Object Manager is called by an application and a new transaction is required, the search path for the new transaction is determined by the character in this field, that is, one of:</p> <p>S Search the system library only.</p> <p>I Search the installation and system libraries.</p> <p>L Search the local, installation and system libraries.</p> <p>However, if the Object Manager is prompting for a session manager menu, the rule is executed with the search path specified for the menu item. If the rule is called, this field has no effect.</p>
Prompt	<p>Type one of:</p> <p>N Do not prompt for rule arguments.</p> <p>A Use the actual argument names as prompts.</p> <p>S Use the strings in the Prompt String field described below.</p> <p>The rule must have two or more arguments if prompting is expected; the first parameter is assigned a value from a specified field of the table.</p> <p>If Prompt is A, the actual names of all but the first arguments of the rule appear and the user is requested to supply values for them. If Prompt is S, the strings contained in fields ARG2, ARG3, and ARG4 appear, and the user is requested to supply values that are passed as the second, third, and fourth arguments of the rule.</p>
Confirm	Type either Y (confirmation is required) or N (confirmation is not required).
Confirm Key	Type the key to use for confirmation, for example, PF22.

Refresh	Type one of:
	<p>Y The screen is to be refreshed after the command is processed. Use this option if the command changed the object list, for example, if an object is deleted.</p>
	<p>N The screen is not to be refreshed after the command is processed.</p>
Field Source of First Argument	<p>If the rule in the Rule field has at least one argument, the entry must be the name of one of the fields in the table. This field is then used as the source of values passed into the argument. Type the name of the appropriate field to be used. If the entry is NULL, no arguments are passed to the rule being invoked.</p>
Field Names or Prompt Strings For Rule Arguments	<p>Up to four arguments can be used by the rule in the Rule field. Enter the appropriate value based on which of the following appears in the Prompt field:</p> <p>N Type the names of the fields to be used; these must be the names of fields in the table or null.</p> <p>If ARG2 is null, only one argument is passed to the rule. If ARG2 contains the name of a field of the table, the contents of this field are passed as the second argument of the rule. ARG3 and ARG4 have similar effects.</p> <p>S Type in a string to be used as a prompt for each of the arguments.</p> <p>The Prompt screen appears. If ARG2 is not null, the string it contains appears as the prompt for the second argument of the rule.</p> <p>Note There is no prompting for the first argument; it is always taken from the table. ARG3, and ARG4 are similar.</p> <p>A The Object Manager displays the actual arguments of the rule named in the Rule field. No entries are required.</p>
Title for Prompt Screen	<p>If prompts are given for a command, the Object Manager displays a screen for the prompts. Use this field for the title of the prompt screen.</p>
Titles to Appear Above and Below the Command Line	<p>The title to appear above the command line. Optionally, type the title to appear below the command line.</p>

Function Keys

The following function keys are recognized while the screen appears:

PF3	Save the definition and return to the workbench.
PF4	Add a line to enter information for the definition.
PF12	Cancel the definition and return to the workbench.
PF16	Delete a line of defined information.
PF22	Delete the definition of a defined object list. You are prompted to confirm the deletion.

Example After executing DEFINE_OBJLIST (EMPLOYEE_LIST) the following screen appears:

Object Commands for Table: EMPLOYEE_LIST						
Command ==>						
Command	Description	Rule	New Trans	Browse	Search	Prompt
-	-	-	-	-	-	-
D	Delete	DEL_OCC	Y	N	S	S
S	Select	STE	Y	N	S	N
Titles to Appear Above and Below the Command Line						
- - - - -						
Listing of Employee Information						
PFKEYS: 12=CANCEL 16=DELLINE 22=DELETE 3=SAVE 4=ADDLINE						

Press PF11 to scroll to additional parts of the screen not visible on your terminal display.

DEFINE_REPORT

Defines a new TIBCO Object Service Broker report or modifies an existing one. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DR define report option	Type <i>reportname</i> <Enter>
	EX execute rule option	Type DEFINE_REPORT (<i>reportname</i>) <Enter>
	COMMAND prompt	Type DR <i>reportname</i> <Enter>

Where

<i>reportname</i>	The name of the report.
-------------------	-------------------------

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Report Definer.
TIBCO Object Service Broker Defining Reports for information on how to define reports.

DEFINE_TABLE

Defines a new TIBCO Object Service Broker table or modifies an existing one. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DT define table option	Type <i>tbl_name</i> <Enter>
	EX Execute Rule option	Type DEFINE_TABLE (<i>tbl_name</i>) <Enter>
	COMMAND prompt	Type DT <i>tbl_name</i> <Enter>

Where:

<i>tbl_name</i>	The name of a table.
-----------------	----------------------

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Table Definer.

TIBCO Object Service Broker Managing Data for information on how to define tables.

\$DELCONTAINER

Deletes a container from a channel and discards the container's data, if any. (C)

Invocation CALL \$DELCONTAINER(*channel*, *container*)

<i>channel</i>	The name (1-16 characters) of the channel that owns the container.
----------------	--

<i>container</i>	The name (1-16 characters) of the container to be deleted.
------------------	--

DELETE_DATA

Deletes the data from a table or table instance. (C)

Invocation `CALL DELETE_DATA(tablespec, select, location)`

<i>tablespec</i>	The name of the table or table instance.
<i>select</i>	The selection criteria to be used.
<i>location</i>	The node name where the data is located.

- Usage Notes**
- If the table specified in *tablespec* is parameterized, you specify only the data parameters, not the location parameter.
 - If you specify an empty string for *select*, all occurrences are deleted from the table and table instance.
 - The syntax for *select* is `<field name> <relational operator> <value>`.

Specify a value for *location* only if the data is located on a different node.

- DELETE_DATA commits any updates.

Exceptions

NO_DATA_FOUND	Raised if the table is empty.
TRIGGER_FAIL	Raised for any other error situations.

In both cases the exception should be handled by the calling rule. More information describing the circumstances of the failure is available in the @OBJECTMSG.MSG field.

Example The following rule deletes selected occurrences from the table EMPLOYEES_REMOTE. The rule is accessing local data.

```

RULE EDITOR ==>>>
DELETE_DATA_1;
-----
--
--
-- CALL DELETE_DATA( 'EMPLOYEES_REMOTE(CANADA)', 'MGR#=79912', | 1
-- '' );
--
-----
```

SCROLL: P

The following rule deletes all occurrences of the DEPTS table, which is parameterized by region:

RULE EDITOR ==>

DELETE_DEPTS;

SCROLL: P

—

—

—

—

—

—

—

—

FORALL \$DEPTS :

CALL DELETE_DATA('DEPTS(' || \$DEPTS.REGION || ')', '',

'');

END;

|

|

|

|

|

|

|

|

1

DELETE_DEFN

Deletes the definition of an object. (C)

Invocation `CALL DELETE_DEFN(objecttype, instancename, library, environment, location, parentonly)`

<i>objecttype</i>	<p>The TIBCO Object Service Broker object type of the object that is to be deleted. Valid values are:</p> <ul style="list-style-type: none"> • GLOBALFIELD • LIBRARY • MENU • OBJECTSET • REPORT • RULE • SCREEN • TABLE • WEBSERVICEPROD
<i>instancename</i>	The name of the object to be deleted.
<i>library</i>	If the object is a rule, the name of the rules library where it is stored.
<i>environment</i>	This argument, although not currently used, must be supplied. You can enter a null ("") value.
<i>location</i>	If the object is located on a different node, the name of the node.
<i>parentonly</i>	<p>Specifies if all the objects or only the parent object should be deleted. Valid values are:</p> <p>Y – Delete only the parent.</p> <p>N – Delete the parent and child objects.</p>

Usage Notes

Specify a value for *parentonly* only if the object is composed of one or more other objects (if, for example, a report is composed of report tables).



DELETE_DEFN eventually calls **FORALLA** on the tables that define an object. If the call to DELETE_DEFN is nested within a FORALL that specifies the same table and parameter values, unexpected behavior occurs, because the set of values returned by the outer FORALL is replaced by the occurrence returned by the FORALLA. To work around this, consider buffering the set of objects to be deleted in another temporary or TDS table and do the FORALL on that table instead.

Exceptions

TRIGGER_FAIL	<div>Raised for any error situation. The exception should be handled by the calling rule.</div> <div>Further information describing the circumstances of the failure is available in the @OBJECTMSG.MSG field.</div>
--------------	--

Example

- The following statement deletes the screen NEW_EMPLOYEE2 and its child objects:

CALL DELETE_DEFN('SCREEN', 'NEW_EMPLOYEE2', '', '', '', 'N');
- The following statement deletes the library USR42:

CALL DELETE_DEFN('LIBRARY', 'USR42', '', '', '', '');

DELETESCREENDATA

Deletes all the occurrences from all the screen tables of a screen. (C)

Invocation `CALL DELETESCREENDATA(screen)`

<i>screen</i>	The name of the parent screen from which all the occurrences in its screen tables are deleted.
---------------	--

Example The rules in this example do the following:

1. Display a screen for data entry.
2. Save the entered data when the Enter key is pressed.
3. Clear the screen tables and re-display the function keys screen table.

NEW_EMPLOYEE Rule

NEW_EMPLOYEE displays the screen NEW_EMPLOYEE and initializes the function keys:

```
NEW_EMPLOYEE;
-
- -----
- -----+-----
- FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');      | 1
- INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                      | 2
- UNTIL EXIT_DISPLAY DISPLAY NEW_EMPLOYEE :                  | 3
-     CALL PROCESS_FCNKEY('NEW_EMPLOYEE');                    |
-     END;                                                     |
- -----
```

Rule using DELETESCREENDATA

SAVE_EMP is the rule initiated when Enter is used. It saves the data, clears the screen tables with DELETESCREENDATA, and re-displays the data contained in the function keys screen table:

```
SAVE_EMP;
-
- -----
- -----+-----
- GET ADD_EMPLOYEE('NEW_EMPLOYEE');                          | 1
- EMPLOYEE.* = ADD_EMPLOYEE.*;                                | 2
- INSERT EMPLOYEE(ADD_EMPLOYEE.REGION);                       | 3
```

```

_ CALL DELETESCREENDATA('NEW_EMPLOYEE');          | 4
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE'); | 5
_ INSERT FCNKEY_SPECS('NEW_EMPLOYEE');              | 6
_ CALL SCREENMSG('NEW_EMPLOYEE', 'EMPLOYEE ADDED'); | 7
-----
_ ON INSERTFAIL :
_   CALL SCREENMSG('NEW_EMPLOYEE', 'EMPLOYEE ALREADY EXISTS');
_ ON GETFAIL :
_   CALL SCREENMSG('NEW_EMPLOYEE', 'ENTER DATA');
```

DIFF_DATA

Compares the data of two tables or table instances and lists the differences. (F)

Invocation

```
differences = DIFF_DATA(table1, field1, location1, selection1, table2, field2,  
location2, selection2, printresult)
```

<i>differences</i>	On return, contains N if there are no differences and Y if there are any differences. The actual differences, if any, are listed in the message log.
<i>table1</i>	The name of the first table or table instance to be compared. Valid values are: <ul style="list-style-type: none"> tablename tablename[(parm_value{,parm_value})]
<i>field1</i>	Specific fields within <i>table1</i> to compare. Valid values are: <ul style="list-style-type: none"> "" – The comparison is based on primary key values only. * – The comparison is based on all fields. fieldnames – Indicates the comparison is based on primary key values and the fields specified in <i>field1</i> and <i>field2</i>. Specify a series of <i>fieldnames</i> in the format: fieldname fieldname fieldname.
<i>location1</i>	The name of the node where <i>table1</i> is located.
<i>selection1</i>	A selection string used to designate which occurrences in <i>table1</i> are to be used for comparison. Valid operators are: =, <, >, <=, >=, and LIKE
<i>table2</i>	The name of the second table or table instance to be compared. Valid values are the same as for <i>table1</i> .
<i>field2</i>	Specific fields within <i>table2</i> to compare. Valid values are the same as for <i>field1</i> .
<i>location2</i>	The name of the node where <i>table2</i> is located
<i>selection2</i>	A selection string used to designate which occurrences in <i>table2</i> are to be used for comparison. Valid operators are the same as for <i>selection1</i> .

<i>printresult</i>	Specifies type of output for the result. Valid arguments are: Y – Print details of the differences in the message log. N – Do not print any details.
--------------------	--

- Usage Notes**
- If the tables specified in *table1* and *table2* are parameterized, specify only the data parameters, not the location parameters.
 - Values for *location1* and *location2* are specified only if the tables are located on different nodes.
 - Concatenated primary keys are not supported with DIFF_DATA.
 - DIFF_DATA can fail if you do a comparison for all fields between tables whose definitions differ.
 - In the output, the letters D and I indicate which items are to be deleted or inserted to make the specified data of *table2* match the specified data of *table1*.

Exceptions

DIFFFAILED	Raised when: <ul style="list-style-type: none">• A table definition is incorrect (DEFINITIONFAIL)• An external or remote table is inaccessible (SERVERBUSY)• There are an unequal number of fields listed for the two tables being compared
-------------------	---

Example This example compares the primary key fields and the DEPTNO fields within each occurrence and returns the results of the comparison to the message log:

```

RULE EDITOR ==>
DIFF_DATA_1;
_ LOCAL DIFFERENCES;
_ -----
_ +-----+
_ DIFFERENCES = DIFF_DATA('EMPLOYEES(MIDWEST)', 'DEPTNO', '' | 1
_ , '', 'EMPLOYEES(CANADA)', 'DEPTNO', '', '', 'Y'); |
_ -----

```

Extract of the Results

The results indicate the changes that should be made to the tables to make the data the same:

COMMAND ==>	SOURCE TABLE	C KEYFIELD (EMPNO)	REASON	SCROLL ==> P
	EMPLOYEES(MIDWEST)	D 22001		
	...			
	EMPLOYEES(MIDWEST)	D 41007		
	EMPLOYEES(MIDWEST)	D 41009	(DEPTNO) = 50	
	EMPLOYEES(MIDWEST)	D 44385		
	EMPLOYEES(MIDWEST)	D 44622	(DEPTNO) = 40	
	EMPLOYEES(MIDWEST)	D 61622		
	.			
	EMPLOYEES(CANADA)	I 32001		
	EMPLOYEES(CANADA)	I 32007		
	EMPLOYEES(CANADA)	I 40058		
	EMPLOYEES(CANADA)	I 41009	(DEPTNO) = 150	
	...			

DIFF_DEFN

Compares the definitions of two objects and list the differences. (F)

Invocation `differences = DIFF_DEFN(objecttype, instance1, library1, environment1, location1, instance2, library2, environment2, location2, details)`

<i>differences</i>	On return, contains the results of the comparison, Y or N.
<i>objecttype</i>	The TIBCO Object Service Broker object type that is to be compared. Valid values are: <ul style="list-style-type: none"> • GLOBALFIELD • LIBRARY • MENU • OBJECTSET • REPORT • RULE • SCREEN • TABLE • WEBSERVICEPROD
<i>instance1</i>	The name of the first object to be compared.
<i>library1</i>	If the object is a rule, the name of the rules library where <i>instance1</i> is stored.
<i>environment1</i>	This argument, although not currently used, must be supplied. You can enter a null (") value.
<i>location1</i>	An identifier indicating the node where <i>instance1</i> is located.
<i>instance2</i>	The name of the second object to be compared.
<i>library2</i>	If the object is a rule, the name of the library where <i>instance2</i> is stored.
<i>environment2</i>	This argument, although not currently used, must be supplied. You can enter a null (") value.
<i>location2</i>	An identifier indicating the node where <i>instance2</i> is located

<i>details</i>	Specifies whether details of the differences are printed. Valid arguments are: Y – Print details of the differences in the message log. N – Do not print into message log.
----------------	--

- Usage Notes
- [DIFFDEFN](#) is the version of this tool used on the developer’s workbench.
 - DIFFDEFN locks the definitions of the items being compared while the comparison is in progress.
 - In the output, the letters D and I indicate which items should be deleted or inserted to make the definition of *instance2* match the definition of *instance1*.

Exceptions	DIFFFAILED Raised for any error situation. The exception should be handled by the calling rule. More information describing the circumstances of the failure is in the @OBJECTMSG.MSG field.
------------	--

Example

The DIFF_DEFN_1 rule compares the definitions of two tables and sends a detailed listing of differences to the message log.

```
RULE EDITOR ==>                                SCROLL: P
DIFF_DEFN_1;
_ LOCAL DIFFERENCES;
_ -----
_ |
_ DIFFERENCES = DIFF_DEFN('TABLE', 'EMPLOYEES', '|', '|', '|', | 1
_ 'EMPLOYEES_REMOTE', '|', '|', '|', 'Y'); |
_ -----
```

The following message log is produced:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
SOURCE TABLE C KEYFIELD (NAME) REASON
-----
FIELDS(EMPLOYEES) D LNAME (KEYTYPE) = S
FIELDS(EMPLOYEES_REMOTE) I LNAME (KEYTYPE) = ---
SOURCE TABLE C KEYFIELD (NUMBER) REASON
-----
PARMS(EMPLOYEES_REMOTE) I 2
```

SOURCE	TABLE	C	KEYFIELD (NAME)	REASON
TABLES	D	EMPLOYEES		
TABLES	I	EMPLOYEES_REMOTE		

DIFF_DEFN_2 compares the definitions of two rules and sends an end message indicating whether there are differences. The results are sent to the message log.

```

      RULE EDITOR ==>
DIFF_DEFN_2;
_ LOCAL DIFFERENCES;
_
_
_
_ DIFFERENCES = DIFF_DEFN('RULE', 'DELETE_EMPLOYEE', 'USR40' | 1
_   , '', '', 'NEW_EMPLOYEE', 'USR40', '', '', 'N');      |
_ CALL ENDMSG('ARE THERE DIFFERENCES? ' || DIFFERENCES);  | 2
_
_
```

The following message appears in the end message:

1:44:56 ARE THERE DIFFERENCES? Y

DIFFDEFN

Compares the definitions of one or more pairs of objects and list the differences.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DD diff defn option	Press Enter
	EX Execute Rule option	Type DIFFDEFN <Enter>
	COMMAND prompt	Type DD <Enter>

- Usage Notes**
- DIFF_DEFN is the callable version of this tool.
 - Using Enter displays the following screen:

```

COMMAND ==>                                COMPARE DEFINITIONS                                Scroll:  P

                                LIST1                                LIST2
-----
Location      :                                Location:
Default Library :                                Library :
Default Environment:
      FIRST List of Objects                                SECOND List
-----
Name          Type          Library/Env          | Name          | Parent
-----
-                                                     | -             | Only
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |
-                                                     | N             |

PFKEYS:  5=SELECT OBJECTS  4=COMPARE  2=LOG  3=EXIT  12=EXIT

```

- To view additional fields from this screen, position your cursor in the appropriate section and press PF11 to scroll right.

- You can type data directly into the fields displayed or you can use PF5 from within the Compare Definitions screen to display a screen that you can use to select the objects that you require.
- After specifying your objects for comparison, press PF4 to list the differences. In the listing, an I indicates that the item should be inserted to make the objects the same and a D indicates that the item should be deleted.

Compare Definitions Screen

The Compare Definitions screen is composed of three sections. Each section is described below:

Specification Section

The specification section contains the following fields:

Location	If the objects to be compared in List1 or List2 are located on a remote node, enter the name of the node.
Default Library/ Library	These fields indicate the library in which to look for rule objects in List1 and List2. The library specification for objects in List1 can be overridden by typing the name of another library in the section below.

List of Objects

This section contains fields for listing the items that are to be compared. Only two items per line can be compared and the items on each line must be the same object type. Different types of objects can be compared within one comparison session.

Name	<p>In the first list, type the name of the object that is to be compared. If you position your cursor in the FIRST List of Objects section and press PF5, a screen appears for selection.</p> <p>In the second list, type the name of the object that is to be used for comparison. If you position your cursor in the SECOND List of Objects section and press PF5, a screen appears for selection.</p> <p>Refer to Select Objects Screen on page 190 for more information about selecting objects.</p>
-------------	--

Type	The name of the object type. Use PF1 for a list of valid values.
Library/Env	If the object is a rule, type the name of the rules library where the rule is located.

Compare Attribute

Parent Only	<p>If the objects are composed of child objects (for example, an object set is composed of objects), specifies if the definitions of the child objects should also be compared. Valid values are:</p> <p>Y – Compare only the definition of the parent.</p> <p>N – Compare the definitions of the parent and children without specifying the child objects as individual items.</p>
-------------	---

PF Keys

In addition to the standard PF keys, the following PF keys are available:

PF4	Compare the definitions as defined.
PF5	Display a screen to select objects.

Select Objects Screen

Pressing PF5 from the Compare Definitions screen displays the following screen:

```

O b j e c t   S e l e c t i o n

COMMAND ==>
Location:                                     Select All: N
Library (for RULES):                         List Children: N
Presentation Environment:

+----- Selection Specification -----+
|      Attr      Op      Value      |
|      NAME      _____ AND      unspecified      |
|      TYPE      =      _____ AND      attributes will      |
|      UNIT      _____ AND      be ignored      |
|      AUTHOR      _____      |
+-----+
Scroll:

```

Name	Type	Library	Environment	Unit
-----	-----	-----	-----	-----

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

Top Section

Specify the following information in the fields. Press PF1 for valid values.

Location	The name of the node where the selection criteria are applied. If you do not specify a value, your home location is used.
Library	If the selection list is to contain rules, type the name of the rules library to be searched.
Presentation Environment	This field is currently not in use.
Select All	Specify if all the items displayed, based on the selection criteria, should be copied into the Compare Definitions screen. Y – Copy all the items displayed. N – Do not copy all the items displayed.
List Children	Specify if you want to list all the child objects from which an object is composed. Valid values are: Y – List all the child objects. N – Do not list the child objects.

Middle Section

The middle section of the screen can be used to select the items to be copied or to narrow the selection list. You can use more than one type of selection criteria for each object type and you can specify multiple object types within one session. For a list of valid values for each of these fields, position your cursor on the field and press PF1.

Bottom Section

When you press Enter after specifying the selection criteria, the selected items appear in the bottom portion of the screen. Select the objects displayed in this section by entering an **S** in the line command field beside the objects. Press PF1 for valid values.

NAME	If you know the name of the item, type the logical operator to be used in the Op field. Type the name of the object in the Value field.
TYPE	The name of the object type. If you do not supply an object type, you must specify a value in at least one of the other selection fields. If you specify an object type and no further selection values, a listing of the items for the object type defined in your TIBCO Object Service Broker database appears for further selection.
UNIT	In the Op field, type the logical operator to be used. In the Value field, type the name of the unit associated with the object.
AUTHOR	In the Op field, type the logical operator to be used. In the Value field, type the name of the author of the object.

PF Keys

You can use the following PF keys from within this screen:

Enter	Update the screen.
PF3	Save the selection and return the Compare Definitions screen.
PF12	Exit without selecting objects and return to the Compare Definitions screen.

Example The following example compares the DEPT_EXPENSE_SUM and DEPT_EXPENSE reports and lists the differences. Both parent and child objects are compared:

COMPARE DEFINITIONS				Scroll: P
COMMAND ==>				
	LIST1		LIST2	
	-----		-----	
	Location :		Location:	
	Default Library :		Library :	
	Default Environment:			

FIRST List of Objects			SECOND List	
Name	Type	Library/Env	Name	Parent Only
DEPT_EXPENSE_SUM	REPORT		DEPT_EXPENSE	N
				N
				N
				N
				N
				N

PFKEYS: 5=SELECT OBJECTS 4=COMPARE 2=LOG 3=EXIT 12=EXIT

Listing of Differences

The following list, produced after pressing PF4, indicates what items should be changed and the reason for the differences:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>
Differences between @REPORTS of REPORT DEPT_EXPENSE_SUM and @REPORTS of REPORT
SOURCE TABLE C KEYFIELD (NAME) REASON
-----
@REPORTS D DEPT_EXPENSE_SUM
@REPORTS I DEPT_EXPENSE

Differences between @REPORTTABLES of REPORT DEPT_EXPENSE_SUM and @REPORTTABLES
SOURCE TABLE C KEYFIELD (NAME) REASON
-----
@REPORTTABLES(DEPT_EXPENS D BANNER_1
@REPORTTABLES(DEPT_EXPENS D DEPT_EXPENSE_SUM
@REPORTTABLES(DEPT_EXPENS D HUR_TITLE (BLANKOVERLAP) = N
@REPORTTABLES(DEPT_EXPENS D TOTAL_EXP_BR
@REPORTTABLES(DEPT_EXPENS I BANNER
@REPORTTABLES(DEPT_EXPENS I DEPT_EXPENSE
@REPORTTABLES(DEPT_EXPENS I HUR_TITLE (BLANKOVERLAP) = ---

```

Object "HUR_TITLE" in LIST1 and object "HUR_TITLE" in LIST2 are alike

DISPLAY_MENU

Calls a specific menu into an application. (C)

Invocation `CALL DISPLAY_MENU(menuname)`

menuname The name of an existing menu.

- Usage Notes**
- The menu called must have been defined with [DEFINE_MENU](#).
 - If the menu has sub-menus embedded within it, these menus are also called.

See Also *TIBCO Object Service Broker Defining Screens and Menus* for information on defining menus.

Example The following example displays the menu SCR_EMPLOYEE:

```
      RULE EDITOR ==>>>                                SCROLL: P
DISPLAY_SCR_EMP;

-
- -----
- -----+-----
- CALL DISPLAY_MENU( 'SCR_EMPLOYEE' );                | 1
- -----
```

DISPLAY_USERS

Displays a list of all users currently logged in to TIBCO Object Service Broker. (E)

Invocation Do one of the following to invoke DISPLAY_USERS:

From the	Move the cursor to the	And
Administrator's workbench	DU Display Users option	Press Enter.
Developer's workbench	EX Execute Rule option	Type DISPLAY_USERS.
	COMMAND prompt	Type EX DISPLAY_USERS.

Executing any of the above commands displays a screen similar to the one illustrated in the example below.

- Usage Notes**
- DISPLAY_USERS shows only TAM type users, not servers.
 - Pressing Enter from the displayed screen refreshes the screen, updating the displayed list and the number of users currently logged in to TIBCO Object Service Broker.
 - You can also display users by issuing the TIBCO Object Service Broker operator command **MODIFY** *job_name*, **USERLIST** from the z/OS operator console.

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* or *TIBCO Object Service Broker for Open Systems Installing and Operating* for information on operator commands.

Example The following is an example of the DISPLAY_USERS screen:

```

                Users logged on
                -----
USR20      Callas, Bill                703-481-6536  TSO User
USR30      Max, Jane                  703-481-6300  TSO User
USR04      Hill, David                703-221-8989  TSO User
USR11      Wodlinger, Susan          703-481-6300  TSO User
USR13      Babalao, Nelson           703-481-6511  CICS User
USR43      Knight, Darelle           703-481-6543  CICS User
USR40      Baraldi, Henry             905-481-6377  TSO User
USR27      Simmons, Susan            408-737-5876  TSO User
USR12      Fernandez, Nellie        705-481-6300  CICS User
USR80      Finn, Tom                 703-481-6300  TSO User

```

USR88	Hudson, Janet	703-481-6308	TSO User
USR33	Strand, Milan	408-982-0880	TSO User
USR44	Dhillon, Al	905-481-6300	TSO User
USR79	Brine, Maureen	408-992-2849	TSO User
USR25	Hrodek, Maxine	703-481-6539	CICS User
USR63	Roeder, Henning	905-481-6326	TSO User

Number of users logged on: 55
Maximum number of users : 200

PFKEYS: ENTER=REFRESH 3=EXIT 12=EXIT

DRAW

Defines a new TIBCO Object Service Broker screen or modifies an existing one. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	DS define screen option	Type <i>screenname</i> <Enter>
	EX execute rule option	Type DRAW (<i>screenname</i>) <Enter>
	COMMAND prompt	Type DS <i>screenname</i> <Enter>

Where:

<i>screenname</i>	The name of a screen.
-------------------	-----------------------

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Screen Definer

TIBCO Object Service Broker Defining Screens and Menus for information on defining or modifying screens.

EDITRULE

Defines a new TIBCO Object Service Broker rule or modifies an existing one. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	ER edit rule option	Type <i>rulename</i> <Enter>
	EX execute rule option	Type EDITRULE(<i>rulename</i>) <Enter>
	COMMAND prompt	Type ER <i>rulename</i> <Enter>

Where:

<i>rulename</i>	The name of a rule.
-----------------	---------------------

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Rule Editor

TIBCO Object Service Broker Programming in Rules for information on creating and editing rules.

ENDMSG

Sets the transaction completion message. (C)

Invocation `CALL ENDMSG(message)`

<i>message</i>	A string specifying the new transaction completion message. Its syntax can be C (fixed-length character string), UN (Unicode), V (variable-length character string), or W (double-byte character).
----------------	--

- Usage Notes**
- The transaction completion message displays only if the transaction completes normally.
 - Only the first 148 characters are returned. Longer lengths can be passed in.
 - The last message of multiple calls to ENDMSG within the same transaction is printed as the transaction completion message.
 - If the value of *message* is of syntax UN and it cannot be coerced to syntax V, ENDMSG returns a Unicode literal.

Example The CHECK_EMP rule checks the EMPLOYEES table for a particular region to see if an employee exists. If the employee does exist, the name, number, position, and city of the employee appear in the end message. If the employee does not exist, the end message informs the user.

```

RULE EDITOR ==>
CHECK_EMP(REGION, LAST_NAME);
-----
GET EMPLOYEES(REGION) WHERE LNAME = LAST_NAME;
CALL ENDMSG('NAME: ' || 'EMPLOYEES'.LNAME ||
' EMPLOYEE #: ' || EMPLOYEES.EMPNO || ' POSITION: ' ||
EMPLOYEES.POSITION || ' CITY: ' || EMPLOYEES.CITY);
ON GETFAIL :
CALL ENDMSG(LAST_NAME || ' NOT FOUND');
```

Entering the value midwest for REGION and smith for LASTNAME results in the following end message:

NAME: SMITH EMPLOYEE #: 41009 POSITION: TESTER CITY: MILTON

Entering the value midwest for REGION and Takada for LAST_NAME results in :

TAKADA NOT FOUND

ENTERKEY

Returns the name of last key used when the specified screen appeared. (F)

Invocation `key_name = ENTERKEY(screen)`

<i>key_name</i>	On return, contains the name of the key used. Its syntax is V (variable-length character string) with length 7.
<i>screen</i>	A character string specifying the screen name. Its syntax is C (fixed-length character string) with length 16.

Usage Notes The value returned is one of the following:

- ENTER
- CLEAR
- PF1 to PF24
- PA1 to PA3



In some environments, the CLEAR key and the PA keys may be reserved by the external system and would therefore not be detected by TIBCO Object Service Broker or returned by ENTERKEY.

Exceptions

RULEFAIL	The request is being made on a non-existent screen or a screen that never appeared.
-----------------	---

Example The following rule fills the example screen with data from the example table, displays it, determines the key used to end the rule, and prints its name to the message log:

```

ENTERKEY_1;
_ LOCAL KEY_NAME;
_ -----
_ -----+-----
_ FORALL EMPLOYEE:                                | 1
_   EMPLOYEE_DATA.* = EMPLOYEE.*;                  |
_   INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');          |
_   END;                                             |
_ DISPLAY EMPLOYEE_SCR;                             | 2
_ KEY_NAME = ENTERKEY('EMPLOYEE_SCR');              | 3
_ CALL MSGLOG('THE NAME OF THE LAST KEY PRESSED IS: ' || | 4

```



```

_ KEY_NAME);
_ -----

```

Resulting Output

Executing this rule displays the following output:

Employee Name	Employee#
SMYTHE	80000
ROTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033
NELSON	81000
CAREY	81001
CHIU	81003
LYNGBAEK	81014
KINGSON	81019

Press Enter to end the rule. Pressing PF2 after the rule ends displays the following output:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE NAME OF THE LAST KEY PRESSED IS: ENTER

```

ESTIMATETBLDFN

Returns an estimate of the maximum CTABLESIZE and XTABLESIZE required for each table type. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type ESTIMATETBLDFN (<i>num_fields</i>) <Enter>
	COMMAND prompt	Type EX ESTIMATETBLDFN (<i>num_fields</i>) <Enter>

Where:

<i>num_fields</i>	The maximum number of fields to be accessible for a table defined in TIBCO Object Service Broker.
-------------------	---

- Usage Notes**
- This function returns an estimate of the maximum CTABLESIZE and XTABLESIZE required (for each TIBCO Object Service Broker table type) to support the number of fields entered as an argument.
 - The default CTABLESIZE shipped is 8 KB, which supports 150 fields.

See Also *TIBCO Object Service Broker Parameters* or the *TIBCO Service Gateway* manual for the appropriate external database for information on the CTABLESIZE and XTABLESIZE Data Object Broker parameters.

Example The screen below displays the results of executing this rule supplied with a parameter of 150 fields:

```

----- INFORMATION LOG -----
COMMAND ==>
DATE: Nov 27,2006          REPORT ON ESTIMATE CTABLESIZE
                           FOR "150" FIELDS

Table Type      CTablesiz(K)      XTablesiz(K)
-----
ADA             11
DAT             19
DB2             12
IDM             13

```

IMS	14	10
MAP	9	
SLK	9	
204	15	
TDS	8	

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT FIND 12=EXIT 13=PRINT 9=RECALL

EVENTFIELD

Returns the name of the field of the subview table that activated the current derived field rule, trigger rule, or validation rule. (F)

Invocation *string* = EVENTFIELD

<i>string</i>	On return, contains the name of the field of the subview table that activated the current derived field rule. Its syntax is C (fixed-length character string) with length 16.
---------------	---

Usage Notes EVENTFIELD can be called only by derived field rules.

Example The following derived field rule doubles the value of the field of the same name in the underlying base table:

```
DOUBLEFIELD;
```

```
—
— -----
— -----+-----
— RETURN(FLDVAL(EVENTTABLE, EVENTFIELD) * 2);      | 1
— -----
```

EVENTSCREEN

Returns the name of the screen that activated the current screen validation rule.
(F)

Invocation *string* = EVENTSCREEN

<i>string</i>	On return, contains the name of the screen that activated the current screen validation rule. Its syntax is C (fixed-length character string) with length 16.
---------------	---

- Usage Notes**
- EVENTSCREEN can be called only by a screen validation rule. It returns the current screen name to the rule.
 - Applications that use EVENTSCREEN cannot use XAL (XML Abstraction Layer).

Example The following validation rule uses EVENTSCREEN:

EVENTSCREEN_1;

—	-----	
—	EMPLOYEE_SCR.EMPNO < 1111;	Y N N
—	EMPLOYEE_SCR.EMPNO > 12000;	Y N
—	-----	+-----
—	CALL SETHILIGHT (EVENTSCREEN, EVENTTABLE, 'EMPNO' , 'Y');	1 1
—	CALL SETCURSOR (EVENTSCREEN, EVENTTABLE, 'EMPNO');	2 2
—	RETURN ('EMPNO CANNOT BE LESS THAN 1111');	3
—	RETURN ('EMPNO CANNOT BE MORE THAN 12000');	3
—	RETURN ('Y');	1
—	-----	-----

Resulting Output

When an invalid value of 12003 is entered into the field EMPNO, the output appears as follows:

Employee Name	Employee#
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033
NELSON	81000
CAREY	81001
CHIU	81003
LYNGBAEK	81014
KINGSON	120003
	—
EMPNO CANNOT EXCEED 12000	

EVENTSUBVIEW

Returns the name of the subview table that activated the current derived field rule. (F)

Invocation *string* = EVENTSUBVIEW

<i>string</i>	On return, contains the name of the field of the subview table that activated the current derived field rule. Its syntax is C (fixed-length character string) with length 16.
---------------	---

Usage Notes EVENTSUBVIEW can be called only by derived field rules.

Example The following derived field rule doubles the value of the field of the same name in the underlying base table, if the name of the subview employing this derived rule begins with the string 'DOUBLE':

```
DOUBLEFIELDIF;
```

```
— -----
— HEADSTRING(EVENTSUBVIEW, 6) = 'DOUBLE';                               | Y N
— -----+-----
— RETURN(FLDVAL(EVENTTABLE, EVENTFIELD) * 2);                           | 1
— RETURN(FLDVAL(EVENTTABLE, EVENTFIELD));                               | 1
— -----
```

EVENTTABLE

Returns the name of the table that activated the current derived field rule, trigger rule, or validation rule. (F)

Invocation *string* = EVENTTABLE

<i>string</i>	On return, contains the name of the table that activated the current derived field rule, trigger rule, or validation rule. Its syntax is C (fixed-length character string) with length 16.
EVENTTABLE	Used within a validation rule. Its syntax is C (fixed-length character string) with length 16.

- Usage Notes**
- EVENTTABLE can be called only by derived field, trigger, or validation rules.
 - Applications that use EVENTTABLE cannot use XAL (XML Abstraction Layer).

Example The following validation rule uses EVENTTABLE:

```
EVENTTABLE_1;
--
-- -----
-- EMPLOYEE_DATA.EMPNO < 1111;                | Y N N
-- EMPLOYEE_DATA.EMPNO > 12000;                |   Y N
-- -----+-----
-- CALL SETHILIGHT (EVENTSCREEN, EVENTTABLE, 'EMPNO' , 'Y'); | 1 1
-- CALL SETCURSOR (EVENTSCREEN, EVENTTABLE, 'EMPNO');        | 2 2
-- RETURN ('EMPNO CANNOT BE LESS THAN 1111');                | 3
-- RETURN ('EMPNO CANNOT BE MORE THAN 12000');                |   3
-- RETURN ('Y');                                              |    1
-- -----
```

Resulting Output

When an invalid value of 12003 is entered into the **EMPNO** field in the screen table EMPLOYEE_DATA, the output appears as follows:

Employee Name	Employee#
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019
CHESSEL	80020
TOWENSEND	80021
NAPIER	80024
CANON	80033
NELSON	81000
CAREY	81001
CHIU	81003
LYNGBAEK	81014
KINGSON	120003
EMPNO CANNOT EXCEED 12000	

\$EXCEPTION

Returns the name of the last exception signalled within the current transaction by either a SIGNAL statement, a \$SIGNAL call, or the system (GETFAIL, ZERODIVIDE, and so on). (F)

Invocation exception_name = \$EXCEPTION

<code>exception_name</code>	On return, contains the name of the last exception signalled.
-----------------------------	---

Usage Note If no exception is signalled in the transaction, \$EXCEPTION returns a null string.

Example This rule uses \$EXCEPTION.

TESTEXCEPTION_1;

```

-----
|
-----
SIGNAL STOP; | 1
|
-----
ON ERROR: |
CALL ENDMSG('Exception = ' || $EXCEPTION || |
          ' object = ' || $EXCEPTIONOBJECT || '') |

```

The resulting end message is shown here:

```
Exception = "STOP" object = ""
```

See Also The [\\$EXCEPTIONOBJECT](#) shareable tool.

\$EXCEPTIONOBJECT

Returns the name of the object (for example, a table) associated with the last exception signalled within the current transaction, if that exception is of the type that can be trapped with an `ON exception_name object_name: statement. (F)`

Invocation `exception_object = $EXCEPTIONOBJECT`

<code>exception_object</code>	On return, contains the name of the object associated with the last exception signalled.
-------------------------------	--

Usage Note `$EXCEPTIONOBJECT` returns a null string under the following conditions:

- If no exception is signalled in the transaction
- If the exception signalled is not of the type that can be trapped with an `ON exception_name object_name: statement`

Example This rule uses `$EXCEPTIONOBJECT`.

```
TESTEXCEPTION_2;
-
- -----
- GET TABLES WHERE NAME = 'DOESNOTEXIST';           | 1
-                                                     |
- -----
- ON ERROR:                                           |
-   CALL ENDMSG('Exception = "' || $EXCEPTION ||     |
-               '" object = "' || $EXCEPTIONOBJECT || |
-               '"')                                   |
-                                                     |
```

The resulting end message is shown here:

Exception = "GETFAIL" object = "TABLES"

See Also The [\\$EXCEPTION](#) shareable tool.

EXIT_DISPLAY

Signals the standard exception EXIT_DISPLAY. (C)

Invocation CALL EXIT_DISPLAY

Exceptions	<hr/>	
	EXIT_DISPLAY	Signalled when this rule is called.

Example In the following example, EXIT_DISPLAY is associated with PF3 in the FCNKEYS(DELETE_EMPLOYEE) table:

EDITING TABLE : FCNKEYS(DELETE_EMPLOYEE)			
COMMAND ==>			
PF_KEY	NAME	COMMAND	ROUTINE

_ PF22	DELETE		DEL_EMP
_ PF3	EXIT		EXIT_DISPLAY

You use the UNTIL clause with the exception EXIT_DISPLAY, as in this rule:

```
DELETE_EMPLOYEE;
_
_ -----
_ -----+-----
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('DELETE_EMPLOYEE'); | 1
_ INSERT FCNKEY_SPECS('DELETE_EMPLOYEE'); | 2
_ UNTIL EXIT_DISPLAY DISPLAY DELETE_EMPLOYEE : | 3
_ CALL PROCESS_FCNKEY('DELETE_EMPLOYEE'); |
_ END; |
_ -----
```

When you press PF3, the EXIT_DISPLAY rule is called. This raises the exception EXIT_DISPLAY, causing the UNTIL loop to end.

EXPOCC_SIZE

Returns the minimum record size required to hold the occurrences of the table being unloaded. (F)

Invocation MAXSIZE = EXPOCC_SIZE(*table*)

<i>table</i>	The name of a TDS table being unloaded.
--------------	---

Usage Notes Use this function to help determine the record length needed in an output data set when using [UNLOAD](#).

Example The following rule, when executed, determines the record size needed to store data unloaded from a given UNIT and returns the size to the end message:

<pre> RULE EDITOR ==> EXPOCC_SIZE_1(UNIT_TO_UNLOAD); _ LOCAL MAXSIZE; _ _ ----- _ _ MAXSIZE = 0; _ FORALL TABLES WHERE UNIT = UNIT_TO_UNLOAD & TYPE = 'TDS' : _ MAXSIZE = MAX(MAXSIZE, EXPOCC_SIZE(TABLES.NAME)); _ END; _ CALL ENDMSG('MINIMUM DATA SET RELOAD SIZE TO UNLOAD' _ ' TABLES IN ' UNIT_TO_UNLOAD ' IS ' MAXSIZE); _ _ ----- </pre>	<pre> SCROLL: P 1 2 3 </pre>
---	------------------------------

FCNKEY_MSG

Creates a string containing the function keys defined for a screen. (F)

Invocation `screentable.field = FCNKEY_MSG(screen)`

<code>screentable.field</code>	The name of the screen table field that displays the function key string.
--------------------------------	---

<code>screen</code>	The name of the screen being displayed.
---------------------	---

- Prerequisites**
- The PF keys must be predefined to the table FCNKEYS.
 - A screen table with a field large enough to contain the function key string must already exist.
- Usage Notes**
- This tool is used together with [PROCESS_FCNKEY](#) to initialize the function key screen table with the function keys defined for a screen.
 - The field MSG_INCLUDE in the FCNKEYS table determines whether the function key appears on the screen.
 - The field MSG_ORDER in the FCNKEYS tables determines the order of the function keys displayed on the screen.
 - The PF_KEY and NAME fields in the FCNKEYS table determine the function key number and name displayed on the screen.

Example The rule in this example prepares a screen called NEW_EMPLOYEE for display by using FCNKEY_MSG to initialize the function key screen table. First you must define the screen NEW_EMPLOYEE. This screen should contain a screen table called FCNKEY_SPECS, which, in turn, has a field called **FCNKEYS**. This is where the function key string appears.

When the screen appears in an application, the user sees the function keys available from the screen in the function key line.

FCNKEYS(NEW_EMPLOYEE) Table

The FCNKEYS (NEW_EMPLOYEE) table displays the function keys and associated routines defines for the NEW_EMPLOYEE screen:

```
BROWSING TABLE      : FCNKEYS(NEW_EMPLOYEE)
COMMAND ==>
```

PF_KEY	NAME	COMMAND	ROUTINE
--------	------	---------	---------

SCROLL: P

—	-----	-----
—	ENTER INPUT DATA	RU_DELSCRDATA_2
—	PF1 HELP	
—	PF12 EXIT	EXIT_DISPLAY
—	PF13 PRINT	PRINT_SCREEN
—	PF2 OPTION	SELECT_OPTION
—	PF3 SAVE	SAVE_EMP

CREATE_EMPLOYEE Rule

Inside an application, the CREATE_EMPLOYEE rule displays the screen:

CREATE_EMPLOYEE(EMPNO);	

CALL DELETESCREENDATA('NEW_EMPLOYEE');	1
FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW EMPLOYEE');	2
INSERT FCNKEY_SPECS('NEW_EMPLOYEE');	3
EMPLOYEE_INFO.EMPNO = EMPNO;	4
INSERT EMPLOYEE_INFO('NEW EMPLOYEE');	5
UNTIL EXIT_DISPLAY DISPLAY NEW_EMPLOYEE :	6
CALL PROCESS_FCNKEY('NEW EMPLOYEE');	
END;	

Displayed Screen

When CREATE_EMPLOYEE is executed, with 04334 as the EMPNO argument, the screen looks as follows:

Region:	Employee No.: 04334
Last Name:	
Position:	
MGR#:	
Deptno:	
Salary:	
Date Hired:	
Address:	
City:	
State or Prov:	
Code:	
FCNKEYS: 3=SAVE 13=PRINT 12=CANCEL	

FLDMGR

Adds fields to the global field dictionary. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	DF define field option	Type <i>fieldname</i> <Enter>
	EX execute rule option	Type FLDMGR(<i>fieldname</i>) <Enter>
	COMMAND prompt	Type DF <i>fieldname</i> <Enter>

Where:

<i>fieldname</i>	The name of the field to be added to the field dictionary.
------------------	--

Usage

If you supply the following value for <i>fieldname</i> ...	Executing FLDMGR displays...
A new value	A screen that you use to define a field.
An existing value	The definition for the field.
No value	<p>A list of fields in the global field dictionary when you press Enter.</p> <p>You can select fields from this list by typing an S in the line command area beside the field name. When you select a field, the definition appears as shown below.</p>

Define Global Field Screen

Define Global Field:	USERID	Unit:	ADMIN
Command ==>			
Type :	I	Syntax :	C
Length :	8	Decimal :	0
Reference :			
Business Name	: TIBCO Object Service Broker USER ID		
Display Mask	:		
Display Length	: 8		

The fields on this screen are described below. They are required unless marked otherwise. For more information about defining fields, refer to the *TIBCO Object Service Broker Managing Data* manual.

Define Global Field	The name of the field to be added to the global field dictionary. It must be a valid TIBCO Object Service Broker identifier.
Unit	[Optional] The user unit with which the field is associated. The UNIT marks the field as belonging to a particular application or a logical unit.
Type	The semantic data type of the field.
Syntax	The syntax of the field.
Length	The length of the field.
Decimal	[Optional] The number of digits to the right of the decimal point. This specification is relevant only for syntax P.
Reference	[Optional] The name of a table that contains a valid set of values for this table. The table must be un-parameterized, and the values must exist as primary key values in the reference table.
Business Name	[Optional] Business description of the field. Maximum 36 characters.
Display Mask	[Optional] The mask to be used on screens and reports.

Scripted Help Screen

When the screen appears, you can position the cursor on a field and press PF1 to get the help specified for the field, as illustrated here:

Formatted Output	Scroll: P
Enter the date in the format of MMM YYYY.	

This help information can then be linked to a screen table, if the field is added to a screen, and appear from within the screen.

If help for the field is not required, or if you want to modify the screen, you can edit the help text that appears within the Screen Table Painter. This new help text overrides the help associated with the global field definition.

Field Dictionary Implementation

As the system administrator, you decide to what extent you want the global field dictionary used. For information on the field dictionary implementation, refer to TIBCO Object Service Broker *Application Administration*.

\$FLUSHPRINT

Releases output into the print spool. (C)

Invocation `CALL $FLUSHPRINT`

- Usage Notes**
- Calls to [\\$PRINTLINE](#) are printed after the transaction completes.
 - Each call to `$FLUSHPRINT` starts a new print job.

Exceptions

ROUTINEFAIL `$FLUSHPRINT` is not preceded by a call to [\\$RESETPRINT](#) or [\\$SETPRINT](#).

Example

The following rule flushes the print spooler, causing two jobs to be printed to the message log:

```
FLUSHPRINT_1;
-
- -----
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N');          | 1
- CALL $PRINTLINE('THIS IS THE FIRST PRINT TRANSACTION. '); | 2
- CALL $FLUSHPRINT;                               | 3
- CALL $PRINTLINE('THIS IS THE SECOND PRINT TRANSACTION. '); | 4
- -----
```

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
----- NEW PAGE -----
THIS IS THE FIRST PRINT TRANSACTION                                Page 1
----- NEW PAGE -----
THIS IS THE SECOND PRINT TRANSACTION                                Page 2
```

If the output is directed to a printer, each of the pages prints as a separate job; each has its own banner pages.

FORALLA

Returns the first table occurrence that satisfies the selection criteria. Use if the value of every table parameter and every selection criterion is 99 or fewer characters long. (C)

Invocation `CALL FORALLA(table, parm, selection, ordering)`

<i>table</i>	A string specifying the table from which to select. Its syntax is C (fixed-length character string), with a length of 16.
<i>parm</i>	A string specifying the parameter set for the table. Its syntax is V (variable-length character string) with length of 512.
<i>selection</i>	A string specifying the selection criteria. Its syntax can be UN (Unicode) or V with a length of 1024.
<i>ordering</i>	A string specifying the order in which to return the occurrences during selection. Its syntax is V, with a length of 512.

The following sections describe the format of each of the arguments.



If the format of the arguments for FORALLA is incorrect, a fatal error can occur with no indication of the cause of the error.

Table The name of the table. For example, to access the table EMPLOYEE, pass 'EMPLOYEE' as the value for the table argument.

Parm The string specified for *parm* consists of a list of parameter values. Each parameter value consists of a two-character length descriptor and a parameter value. For example, the string 06CANADA0580002 represents two parameters. The first is six characters long with the value CANADA. The second is five characters long with the value 80002.

Both the data parameters and the location parameter can be specified in this manner. If the table has no parameters or the parameters are being specified in the selection parameter of FORALLA, the value of *parm* can be null (NULL).

Selection The string specified for selection is a postfix representation of the rules language WHERE clause. It is comprised of one or more terms joined by logical operators. Each term consists of the following:

- A field reference

- An expression

These elements are described here.

Field Reference

A field reference consists of the following:

- A two-character descriptor consisting of a space followed by an R (' R').
- A 16 character field name. If the name is not 16 characters long, it must be padded to 16 characters with spaces.

Expression

An expression consists of the following:

- A two-character value syntax descriptor consisting of a space followed by one of the following (for example, ' V') to denote the syntax of the value:
 - A – syntax RD (raw data): FORALLA uses, or coerces the value to, syntax V; and then coerces it to syntax RD (see note below)
 - E – syntax V (variable-length character string): FORALLA uses, or coerces the value to, syntax V; if the coercion to V fails, it raises an exception
 - M – a numeric string: FORALLA uses, or coerces the string to, syntax V; and then coerces it to syntax B (binary), P (packed), or F (floating point); if either coercion fails, it raises an exception
 - N – a null value: no length or value should follow this descriptor
 - U – syntax UN (Unicode)
 - V – syntax UN or V. FORALLA uses, or coerces the value to, syntax V; if the coercion to syntax V fails, it uses syntax UN
- A two-character value length descriptor (prefix a length of one digit with a zero if needed)

You must specify the length as returned by the LENGTH shareable tool.

- The value
- A relational operator

The relational operators are: =, \neq (not equal to), >, \geq , <, \leq , and L (LIKE).



The relational operators must be two characters in length. Prefix a space before an operator that consists of only one character.

- If necessary, an arithmetic or a logical operator

The arithmetic operators are: +, -, *, /, **, and ||.



The arithmetic operators must be two characters in length. Prefix with a space an operator that consists of only one character. For unary minus, place a space after the -.

The logical operators are: &, |, and ¬.



The logical operators must be two characters in length. Prefix a space before the operator.



The syntax of *selection* determines the representation of values in the expression. The usual technique for building *selection* is to use a series of concatenations. Here are some examples (where the local variable VAL contains the value in the expression to be built):

- `SEL_STR = SEL_STR || ' V' || $PIC(LENGTH(VAL), '99') || VAL`

If VAL is of syntax V, SEL_STR preserves its syntax; if VAL is of syntax UN, SEL_STR is coerced to syntax UN.

- `SEL_STR = SEL_STR || ' U' || $PIC(LENGTH(VAL), '99') || VAL`

This makes sense only if VAL is of syntax UN.

- `SEL_STR = SEL_STR || ' E' || $PIC(LENGTH(VAL), '99') || VAL`

Regardless of *selection*'s and VAL's syntax, FORALLA coerces the value to syntax V.

- `SEL_STR = SEL_STR || ' M' || $PIC(LENGTH(VAL), '99') || VAL`

Regardless of *selection*'s and VAL's syntax, FORALLA coerces the value to syntax B, P, or F, whichever is appropriate.



To include a raw-data value in the expression, use descriptor 'A', keeping in mind that a direct concatenation involving an RD value coerces the result to RD. If, as a result, the selection string is of syntax RD, FORALLA coerces it to syntax V prior to any processing, so some data loss is possible (for example, if the selection contains a value of syntax UN). Accordingly, the following is recommended: assuming VAL is of syntax RD, first cast it to syntax V:

```
VAL1 = $TYPECAST(' ', 'V', LENGTH(VAL), 0, VAL)
```

and then concatenate the regular way:

```
SEL_STR = SEL_STR || ' A' || $PIC(LENGTH(VAL1), '99') || VAL1
```

Even if SEL_STR is of syntax UN prior to the latter action, FORALLA coerces correctly the value to RD.

Examples of Selections

In these examples, the “B” represents a mandatory space.

- The following represents the selection CITY= 'TORONTO' :

```
BRCITYBBBBBBBBBBBBBBBVB07TORONTOB=
```

The pattern is the name of the field, followed by the value, followed by the relational operator.

- The following represents the selection CITY= 'TORONTO' AND SALARY > 600:

```
BRCITYBBBBBBBBBBBBBBBVB07TORONTOB=BSALARYBBBBBBBBBBBBBBBM03600B>B&
```

The field name (**CITY**) is followed by the value (TORONTO) and relational operator (=), and then another sequence of field (**SALARY**), value (600), and relational operator (>), followed by the logical operator (&) that joins the two expressions.

- The following represents the selection CITY= 'TORONTO' & SALARY > 600 | MGR# = 80002:

```
BRCITYBBBBBBBBBBBBBBBVB07TORONTOB=BSALARYBBBBBBBBBBBBBBBM03600B>B&BMRMGR#BBBBBBBBBBBBBBBM0580002B=B|
```

The first and second expressions are the same, and are followed by a third sequence of field (**MGR#**), value (80002), relational operator (=), and a logical operator (|) that joins the third expression to the selection string.

If Selection is Not Required

If selection on a table is not required, you can pass an empty value (") for the selection parameter.

Ordering The string specified for the ordering parameter is a list of terms consisting of the following:

- A 16-character field name. If the field name is less than 16 characters long, it must be padded to 16 characters.
- A one-character ordering specifier. The ordering specifier is A for ascending and D for descending.

The ORDERED clause ORDERED DESCENDING LNAME AND ORDERED ASCENDING EMPNO would translate to an ordering parameter like this:

```
LNAMEBBBBBBBBBBBDEMPNOBBBBBBBBBBBA
```

If Ordering is Not Required

If you do not need ordering, you can pass an empty value (") for the ordering parameter. In this case, the occurrences are sorted by primary key.

Usage Notes

- If the value of any table parameter or of the selection criterion is 100 or more characters long, use @FORALLA rather than FORALLA.
- Using FORALLA in conjunction with FORALLB and FORALLE, you can dynamically build a selection criterion to retrieve table occurrences.
- The table parameters can be specified in either the *parm* or *selection* arguments.
- There are a maximum of 16 tables per transaction that can be actively accessed with FORALLA. When you are finished accessing a table, you should call FORALLE to indicate that you are finished and to free up resources for another table to be accessed.
- The PARSE_TAM tool can be used to create the necessary arguments for FORALLA, or if you only need to build the selection argument you can use the SIMPLESELECT tool.



A FORALLA within a FORALL on the same table (and with the same parameter values) can result in unexpected behavior because the set of occurrences returned by the outer FORALL is replaced by the occurrence returned by FORALLA.

Exceptions

TABLEEND	No occurrences satisfy the selection criteria.
CONVERSION	One or more of the supplied parameters are invalid.

Example

Using the following rule, QUERY_EMPS, users can construct a query against the EMPLOYEE table. The rule retrieves employee names based on the selection criteria associated with one field, or with a combination of two fields, and prints them to the message log. Users can execute the rule directly or the rule could be part of a larger application in which input values are entered via a screen.

The rule supports queries such as:

- Which employees working for a particular manager have a salary greater than a given amount?
- Which employees in a given position live in a particular city?

The QUERY_EMPS Rule

QUERY_EMPS pads input data to appropriate lengths, constructs a FORALLA statement using the data from one or two fields, and then uses FORALLB, FORALLE, and [MSGLOG](#) to retrieve and display the names of all employees meeting the selection criteria in the FORALLA statement.

```

RULE EDITOR ==>
QUERY_EMPS(REGION, FIELD1, OPER1, VALUE1, ORD1, OPER3, FIELD2, OPER2,
  VALUE2, ORD2);
LOCAL REGLEN, VALLEN1, VALLEN2, FLD1, OP1, FLD2, OP2;

-----
OPER3 = '';
-----
REGLEN = PAD(LENGTH(REGION), 2, '0', 'R');
VALLEN1 = PAD(LENGTH(VALUE1), 2, '0', 'R');
VALLEN2 = PAD(LENGTH(VALUE2), 2, '0', 'R');
FLD1 = PAD(FIELD1, 16, ' ', 'L');
FLD2 = PAD(FIELD2, 16, ' ', 'L');
OP1 = PAD(OPER1, 2, ' ', 'R');
OP2 = PAD(OPER2, 2, ' ', 'R');
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ':'
);
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ' '
|| OPER3 || ' ' || FIELD2 || ' ' || OPER2 || ' ' ||
VALUE2 || ':');
CALL MSGLOG(' ');
CALL FORALLA('EMPLOYEE', REGLEN || REGION, ' R' || FLD1 ||
' V' || VALLEN1 || VALUE1 || OP1 || ' R' || FLD2 || ' V'
|| VALLEN2 || VALUE2 || OP2 || ' ' || OPER3, FLD1 ||
ORD1 || FLD2 || ORD2);
UNTIL TABLEND :
  CALL MSGLOG(EMPLOYEE.LNAME);
  CALL FORALLB('EMPLOYEE');
  END;
CALL FORALLE('EMPLOYEE');
CALL ENDMSG('PRESS PF2 FOR LIST');
-----
ON TABLEND :
  CALL FORALLE('EMPLOYEE');
  CALL ENDMSG('NO MATCHING OCCURRENCES FOUND');

```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

Query

To find those employees who work for manager number 79912 and have a salary of more than \$700.00, the following input values are supplied:

----- H U R O N RULE EXECUTION -----

```
ENTER ARGUMENTS FOR RULE QUERY_EMPS

REGION          ===> midwest
FIELD1          ===> mgr#
OPER1           ===> =
VALUE1          ===> 79912
ORD1            ===> a
OPER3           ===> &
FIELD2          ===> salary
OPER2           ===> >
VALUE2          ===> 700
ORD2            ===> a
```

The message log displays the following employee names, arranged in order of ascending salary:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
MGR# = 79912 & SALARY > 700:

HRODEK
BOIVIN
SCHULTZ
WONG
```

@FORALLA

Returns the first table occurrence that satisfies the selection criteria. Use if the value of any table parameter or of any selection criterion is 100 or more characters long. (C)

Invocation `CALL @FORALLA(table, parm, selection, ordering)`

<i>table</i>	A string specifying the table from which to select. Its syntax is C (fixed-length character string), with a length of 16.
<i>parm</i>	A string specifying the parameter set for the table. Its syntax is V (variable-length character string) with length of 512.
<i>selection</i>	A string specifying the selection criteria. Its syntax can be UN (Unicode) or V with length of 4096.
<i>ordering</i>	A string specifying the order in which to return the occurrences during selection. Its syntax is V, with a length of 512.

The following sections describe the format of each of the arguments.



If the format of the arguments for @FORALLA is incorrect a fatal error can occur with no indication of the cause of the error.

Table The name of the table. For example, to access the table EMPLOYEE, pass 'EMPLOYEE' as the value for the table argument.

Parm The string specified for *parm* consists of a list of parameter values. Each parameter value consists of:

- A three-character length descriptor
- A parameter value

The string 006CANADA00580002 contains two parameters. The first is 6 characters long with the value CANADA. The second is 5 characters long with the value 80002.

Both the data parameters and the location parameter can be specified in this manner. If the table has no parameters or the parameters are being specified in the selection parameter of @FORALLA, the value of *parm* can be null (NULL).

Selection The string specified for selection is a postfix representation of the rules language WHERE clause. It is comprised of one or more terms joined by logical operators. Each term consists of the following:

- A field reference
- An expression

These elements are described here.

Field Reference

A field reference consists of the following:

- A two-character descriptor consisting of a space followed by an R (' R ').
- A 16-character field name. If the name is not 16 characters long, it must be padded to 16 characters with spaces.

Expression

An expression consists of the following:

- A two-character value syntax descriptor consisting of a space followed by one of the following (for example, ' V') to denote the syntax of the value:
 - A – syntax RD (raw data): @FORALLA uses, or coerces the value to, syntax V; and then coerces it to syntax RD (see note below)
 - E – syntax V (variable-length character string): @FORALLA uses, or coerces the value to, syntax V; if the coercion to V fails, it raises an exception
 - M – a numeric string: @FORALLA uses, or coerces the string to, syntax V; and then coerces it to syntax B (binary), P (packed), or F (floating point); if either coercion fails, it raises an exception
 - N – a null value: no length or value should follow this descriptor
 - U – syntax UN (Unicode)
 - V – syntax UN or V. @FORALLA uses, or coerces the value to, syntax V; if the coercion to syntax V fails, it uses syntax UN
- A four-character value length descriptor (prefix a length shorter than four digits with zeroes as needed)

You must specify the length as returned by the LENGTH shareable tool

- The value
- A relational operator

The relational operators are: =, \neq (not equal to), >, >=, <, <=, and L (LIKE).



The relational operators must be two characters in length. Prefix a space before an operator that consists of only one character.

- If necessary, an arithmetic or a logical operator

The arithmetic operators are: +, -, *, /, **, and | |.



The arithmetic operators must be two characters in length. Prefix with a space an operator that consists of only one character. For unary minus, place a space after the -.

The logical operators are: &, |, and \neg .



The logical operators must be two characters in length. Prefix the operator with a space.



The syntax of *selection* determines the representation of values in the expression. The usual technique for building *selection* is to use a series of concatenations. Here are some examples (where the local variable VAL contains the value in the expression to be built):

- `SEL_STR = SEL_STR || ' V' || $PIC(LENGTH(VAL), '9999') || VAL`

If VAL is of syntax V, SEL_STR preserves its syntax; if VAL is of syntax UN, SEL_STR is coerced to syntax UN.

- `SEL_STR = SEL_STR || ' U' || $PIC(LENGTH(VAL), '9999') || VAL`

This makes sense only if VAL is of syntax UN.

- `SEL_STR = SEL_STR || ' E' || $PIC(LENGTH(VAL), '9999') || VAL`

Regardless of *selection*'s and VAL's syntax, @FORALLA coerces the value to syntax V.

- `SEL_STR = SEL_STR || ' M' || $PIC(LENGTH(VAL), '9999') || VAL`

Regardless of *selection*'s and VAL's syntax, @FORALLA coerces the value to syntax B, P, or F, whichever is appropriate.



To include a raw-data value in the expression, use descriptor 'A', keeping in mind that a direct concatenation involving an RD value coerces the result to RD. If, as a result, the selection string is of syntax RD, FORALLA coerces it to syntax V prior to any processing, so some data loss is possible (for example, if the selection contains a value of syntax UN). Accordingly, the following is recommended: assuming VAL is of syntax RD, first cast it to syntax V:

```
VAL1 = $TYPECAST(' ', 'V', LENGTH(VAL), 0, VAL)
```

and then concatenate the regular way:

```
SEL_STR = SEL_STR || ' A' || $PIC(LENGTH(VAL1), '9999') || VAL1
```

Even if SEL_STR is of syntax UN prior to the latter action, FORALLA coerces correctly the value to RD.

Examples of Selections

In these examples, the “B” represents a mandatory space.

- The following represents the selection CITY='TORONTO':

```
BRCITYBBBBBBBBBBBBBBBV0007TORONTOB=
```

The pattern is the name of the field, followed by the value, followed by the relational operator.

- The following represents the selection CITY='TORONTO' AND SALARY > 600:

```
BRCITYBBBBBBBBBBBBBBBV0007TORONTOB=BSALARYBBBBBBBBBBBBBBBM0003600B>B&
```

The name of the field (**CITY**) is followed by the value (TORONTO) and relational operator (=), and then another sequence of field (**SALARY**), value (600), relational operator (>), followed by the logical operator (&) that joins the two expressions.

- The following represents the selection CITY='TORONTO' & SALARY > 600 | MGR# = 80002:

```
BRCITYBBBBBBBBBBBBBBBV0007TORONTOB=BSALARYBBBBBBBBBBBBBBBM0003600B>B&BMGR#BBBBBBBBBBBBBBBM000580002B=B|
```

The first and second expressions are the same, and are followed by a third sequence of field (**MGR#**), value (80002), relational operator (=), and a logical operator (|) that joins the third expression to the selection string.

If Selection is Not Required

If selection on a table is not required, you can pass an empty value ("") for the selection parameter.

- Ordering
- The string specified for the ordering parameter is a list of terms consisting of the following:
- A 16-character field name. If the field name is less than 16 characters long, it must be padded to 16 characters.
 - A one-character ordering specifier, which is A for ascending and D for descending.

The ORDERED clause ORDERED DESCENDING LNAME AND ORDERED ASCENDING EMPNO would translate to an ordering parameter like this:

LNAMEBBBBBBBBBBDEMPNOBBBBBBBBBBBA

If Ordering is Not Required

If you do not need ordering, you can pass an empty value (") for the ordering parameter. In this case, the occurrences are sorted by primary key.

- Usage Notes
- If the value of every table parameter and of the selection criterion is 99 or fewer characters long, use [FORALLA](#) rather than @FORALLA.
 - Using @FORALLA in conjunction with [FORALLB](#) and [FORALLE](#), you can dynamically build a selection criterion to retrieve table occurrences.
 - The table parameters can be specified in either the *parm* or *selection* arguments.
 - There are a maximum of 16 tables per transaction that can be actively accessed with @FORALLA. When you finish accessing a table, you should call FORALLE to indicate that you are finished and to free up resources for another table to be accessed.
 - If you cannot create the necessary arguments for @FORALLA, consider using [PROCESS_TABLE](#).



An @FORALLA within a FORALL on the same table (and with the same parameter values) can result in unexpected behavior because the set of occurrences returned by the outer FORALL is replaced by the occurrence returned by @FORALLA.

Exceptions

TABLEEND	No occurrences satisfy the selection criteria.
CONVERSION	One or more of the supplied parameters are invalid.

Example Using the following rule, QUERY_EMPLOYEES, users can construct a query against the EMPLOYEE table. The rule retrieves employee names based on the selection criteria associated with one field, or with a combination of two fields, and prints them to the message log. Users can execute the rule directly or the rule could be part of a larger application in which input values are entered via a screen.

The rule supports queries such as:

- Which employees working for a particular manager have a salary greater than a given amount?
- Which employees in a given position live in a particular city?

The QUERY_EMPLOYEES Rule

QUERY_EMPLOYEES pads input data to appropriate lengths, constructs a @FORALLA statement using the data from one or two fields, and then uses FORALLA, FORALLE, and MSGLOG to retrieve and display the names of all employees meeting the selection criteria in the @FORALLA statement.

```

RULE EDITOR ==>
QUERY_EMPLOYEES(REGION, FIELD1, OPER1, VALUE1, ORD1, OPER3, FIELD2, OPER2,
  VALUE2, ORD2);
LOCAL REGLEN, VALLEN1, VALLEN2, FLD1, OP1, FLD2, OP2;
-----
OPER3 = '';
-----| Y N
-----|-----
REGLEN = PAD(LENGTH(REGION), 3, '0', 'R');| 1 1
VALLEN1 = PAD(LENGTH(VALUE1), 4, '0', 'R');| 2 2
VALLEN2 = PAD(LENGTH(VALUE2), 4, '0', 'R');| 3
FLD1 = PAD(FIELD1, 16, ' ', 'L');| 3 4
FLD2 = PAD(FIELD2, 16, ' ', 'L');| 5
OP1 = PAD(OPER1, 2, ' ', 'R');| 4 6
OP2 = PAD(OPER2, 2, ' ', 'R');| 7
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ':'| 5
);|
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ' '| 8
|| OPER3 || ' ' || FIELD2 || ' ' || OPER2 || ' ' ||
VALUE2 || ':');|
CALL MSGLOG(' ');| 6 9
CALL @FORALLA('EMPLOYEE', REGLEN || REGION, ' R' || FLD1 ||| 7
' V' || VALLEN1 || VALUE1 || OP1 || ' R' || FLD2 || ' V'
|| VALLEN2 || VALUE2 || OP2 || ' ' || OPER3, FLD1 ||
ORD1 || FLD2 || ORD2);|
UNTIL TABLEEND :| 8 B
CALL MSGLOG(EMPLOYEE.LNAME);|
CALL FORALLB('EMPLOYEE');|
END;|
CALL FORALLE('EMPLOYEE');| 9 C
CALL ENDMMSG('PRESS PF2 FOR LIST');| A D
-----
ON TABLEEND :
```

```

- CALL FORALLE('EMPLOYEE');
- CALL ENDMSG('NO MATCHING OCCURRENCES FOUND');

```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

Query

To find those employees who work for manager number 79912 and have a salary of more than \$700.00, the following input values are supplied:

```

----- H U R O N   RULE EXECUTION -----
ENTER ARGUMENTS FOR RULE QUERY_EMPLOYEES

REGION          ===> midwest
FIELD1          ===> mgr#
OPER1           ===> =
VALUE1          ===> 79912
ORD1            ===> a
OPER3           ===> &
FIELD2          ===> salary
OPER2           ===> >
VALUE2          ===> 700
ORD2            ===> a

```

The message log displays the following employee names, arranged in order of ascending salary:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
MGR# = 79912 & SALARY > 700:

HRODEK
BOIVIN
SCHULTZ
WONG

```

FORALLB

Returns the next table occurrence that satisfies the selection criteria following a call to [FORALLA](#). (C)

Invocation `CALL FORALLB(table)`

<i>table</i>	This is a string specifying the name of the table to retrieve from. It is syntax C (fixed-length character string) and length 16.
--------------	---

Prerequisites Before FORALLB can be called the table must be initialized by a call to FORALLA.

Usage Notes

- FORALLB continues to retrieve occurrences based on the selection and ordering criteria specified in the call to FORALLA. After one initial call to FORALLA, FORALLB can be called repeatedly to retrieve multiple occurrences from a table.
- To use FORALLB to retrieve all the occurrences in the table that match the selection criteria specified in a preceding call to FORALLA, call FORALLB within a loop created by an UNTIL TABLEEND statement. FORALLB continues to retrieve occurrences until there are no more occurrences that satisfy the criteria. At that point, FORALLB raises the TABLEEND exception and the loop exits.
- There are a maximum of 16 tables that can be actively accessed with FORALLB per transaction. When you are finished accessing a table, you should call [FORALLE](#) to indicate that you are finished and to free up a slot for another table to be accessed.

See Also *TIBCO Object Service Broker Programming in Rules* for information on the rules language.

Exception

TABLEEND	There are no more occurrences to process that satisfy the selection criteria.
-----------------	---

Example Using the following rule, QUERY_EMPS, users can construct a query against the EMPLOYEE table. The rule retrieves employee names based on the selection criteria associated with one field or with a combination of two fields and prints them to the message log. Users can execute the rule directly, or the rule could be part of a larger application in which input values are entered via a screen.

The rule supports queries such as:

- Which employees work for a given manager and earn a given amount?
- Which employees in a given position live in a particular city?

The QUERY_EMPS Rule

QUERY_EMPS pads input data to appropriate lengths, constructs a FORALLA statement using the data from one or two fields, and then uses FORALLB, FORALLE, and [MSGLOG](#) to retrieve and display the names of all employees meeting the selection criteria in the FORALLA statement.

The FORALLB statement at action 8 in the Y column shows the use of an UNTIL TABLEEND loop.

```

RULE EDITOR ==>
QUERY_EMPS(REGION, FIELD1, OPER1, VALUE1, ORD1, OPER3, FIELD2, OPER2,
  VALUE2, ORD2);
LOCAL REGLEN, VALLEN1, VALLEN2, FLD1, OP1, FLD2, OP2;
-----
OPER3 = '';
-----
REGLEN = PAD(LENGTH(REGION), 2, '0', 'R');
VALLEN1 = PAD(LENGTH(VALUE1), 2, '0', 'R');
VALLEN2 = PAD(LENGTH(VALUE2), 2, '0', 'R');
FLD1 = PAD(FIELD1, 16, ' ', 'L');
FLD2 = PAD(FIELD2, 16, ' ', 'L');
OP1 = PAD(OPER1, 2, ' ', 'R');
OP2 = PAD(OPER2, 2, ' ', 'R');
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ':'
);
CALL MSGLOG(FIELD1 || ' ' || OPER1 || ' ' || VALUE1 || ' '
|| OPER3 || ' ' || FIELD2 || ' ' || OPER2 || ' ' ||
VALUE2 || ':');
CALL MSGLOG(' ');
CALL FORALLA('EMPLOYEE', REGLEN || REGION, ' R' || FLD1 ||
' V' || VALLEN1 || VALUE1 || OP1, FLD1 || ORD1);
|| VALLEN2 || VALUE2 || OP2 || ' ' || OPER3, FLD1 ||
ORD1 || FLD2 || ORD2);
UNTIL TABLEEND :
CALL MSGLOG(EMPLOYEE.LNAME);
CALL FORALLB('EMPLOYEE');
END;
CALL FORALLE('EMPLOYEE');
CALL ENDMSG('PRESS PF2 FOR LIST');
-----
ON TABLEEND :
CALL FORALLE('EMPLOYEE');
CALL ENDMSG('NO MATCHING OCCURRENCES FOUND');

```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

Query

To find which employees working for manager number 79912 have a salary of more than \$700.00, the following input values are supplied:

```
----- H U R O N   RULE EXECUTION -----  
  
ENTER ARGUMENTS FOR RULE QUERY_EMPS  
  
REGION          ===> midwest  
FIELD1          ===> mgr#  
OPER1           ===> =  
VALUE1          ===> 79912  
ORD1            ===> a  
OPER3           ===> &  
FIELD2          ===> salary  
OPER2           ===> >  
VALUE2          ===> 700  
ORD2            ===> a
```

The message log displays the following employee names, arranged in order of ascending salary:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ===>                                SCROLL ===> P  
MGR# = 79912 & SALARY > 700 :  
  
HRODEK  
BOIVIN  
SCHULTZ  
WONG
```


FORALLE

Releases internally the resources used by FORALLA on a table. (C)

Invocation `CALL FORALLE(table)`

<i>table</i>	Name of a table involved in a previous FORALLA or FORALLB call. Its syntax is C (fixed-length character string) with length of 16.
--------------	--

- Usage Notes**
- There are a maximum of 16 tables that can be actively accessed with FORALLA and FORALLB per transaction. An index is maintained for each table being accessed by FORALLA and FORALLB, and there is a maximum of 16 simultaneous indexes per transaction. FORALLE clears the index for the specified table, and frees up resources for another table to be accessed. When you are finished accessing a table with FORALLA or FORALLB, you should call FORALLE to indicate that you are finished and free up resources for another table to be accessed.
 - All exit points for FORALLA and FORALLB should have a FORALLE statement.

Example The following rule uses FORALLA and FORALLB to retrieve all the occurrences in the MIDWEST parameter instance of the EMPLOYEE table and put them in the message log. FORALLE is used to clean up the table index when the loop exits normally or when an exception is raised. All the exception handlers call FORALLE, so that all ending points for the FORALLA, FORALLB sequence are covered by a FORALLE.

```
FORALLE_1;
-----
CALL FORALLA('EMPLOYEE', '07MIDWEST', '', '');
UNTIL TABLEEND:
    CALL MSGLOG('EMPLOYEE ' || EMPLOYEE.LNAME ||
               ' IS NUMBER ' || EMPLOYEE.EMPNO);
    CALL FORALLB('EMPLOYEE');
    END;
CALL FORALLE('EMPLOYEE');
-----
ON TABLEEND :
    CALL FORALLE('EMPLOYEE');
    CALL MSGLOG('NO OCCURRENCES WERE FOUND. ');
ON SECURITYFAIL:
    CALL FORALLE('EMPLOYEE');
    CALL MSGLOG('A SECURITYFAIL WAS RAISED');
ON LOCKFAIL:
```

```
— CALL FORALLE('EMPLOYEE');  
— CALL MSGLOG('A LOCKFAIL WAS RAISED');  
— ON DEFINITIONFAIL:  
— CALL FORALLE('EMPLOYEE');  
— CALL MSGLOG('A DEFINITIONFAIL WAS RAISED');
```

FROM_UNICODE

Converts a Unicode string to Raw Data encoded in an external Code page.(F)

Invocation `rd_string = FROM_UNICODE(unistring, externalcodepage)`

<code>rd_string</code>	On return, contains a string of RD (raw data) syntax. The data encoding corresponds to one of the 16 possible user-defined external syntaxes XC01 to XC16.
<code>unistring</code>	Contains a string of UN (unicode) syntax.
<code>externalcodepage</code>	One of the values 'XC01' to 'XC16' representing the user-defined external syntax.

Example This rule converts a string encoded in Unicode to external syntax XC02.

```

TESTFROMUNI;
_ LOCAL IDATA, RESULT;
_ -----
_ -----+-----
_ IDATA = U'ABC DEF';                               | 1
_ RESULT = FROM_UNICODE(IDATA, 'XC02');               | 2
_ CALL ENDMSG(RESULT);                               | 3

```

Line 1 sets local variable IDATA as a Unicode string. Line 2 converts it to a Raw Data string encoded in external code page XC02. Line 3 displays the result:
ABC DEF.

\$FUNCTION

Invokes a functional rule. (F)

Invocation `result = $FUNCTION(rulecall)`

<code>result</code>	The value returned by the rule.
<code>rulecall</code>	A character string of syntax V (variable-length character string) or UN (Unicode) containing the name of the rule to be invoked, and, if the rule takes one or more arguments, a left parenthesis, a list of the comma-separated arguments, and a right parenthesis.

- Usage Notes**
- You can pass a NULL value to the rule by omitting an actual value from the list of parameters in the position of the parameter to be passed the NULL. For example, in the sample rule below, to use ADD without a value for B, you would use:

`C = $FUNCTION('ADD(' || A || ',')');`
 - If the rule to be invoked does not exist in a library in the current search order of the transaction, an untrappable error occurs. You can use the \$RULE_EXISTS shareable tool to test for the existence of a rule before trying to invoke it.

Exceptions

EXECUTEFAIL	Raised if <i>rulecall</i> is not a string conforming to the format described above.
--------------------	---

Example This rule receives a request to process two numbers by adding them together or subtracting them from each other. It then calls the appropriate user-written rule to do that processing. If one of the rules does not exist, the terminal displays an end message.

RULE EDITOR ==>	SCROLL: P
ARITHF(A, OPERATION, B);	
_ LOCAL C;	

_ \$RULE_EXISTS('ADD') = 'N';	Y N N N N
_ \$RULE_EXISTS('SUBTRACT') = 'N';	Y N N N
_ OPERATION = '+';	Y N N
_ OPERATION = '-';	Y N

_ SIGNAL NORULE;	1 1

```
- C = $FUNCTION('ADD(' || A || ' , ' || B || ')');           |      1
- C = $FUNCTION('SUBTRACT(' || A || ' , ' || B || ')');       |      1
- CALL ENDMSG('THE RESULT OF ' || A || ' OPERATION ' || B ||  |      2 2
-   ' IS ' || C);                                             |
-----
- ON NORULE :
-   CALL ENDMSG('RULE NOT FOUND');
- ON EXECUTEFAIL :
-   CALL ENDMSG('CHECK SYNTAX IN RULE');
```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

This is one of the rules ARITHC calls:

```
      RULE EDITOR ==>>                                SCROLL: P
ADD(A, B);
-
- -----
-                                     +-----
- RETURN(A + B);                                | 1
- -----
```

PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT 14=EXPAND 2=DOCUMENT 22=DELETE

The output message from ARITHC is similar to:

THE RESULT OF 7+2 IS 9

GEN_TED

Presents a screen where text can be entered and edited under the control of the text editor. (C)

Invocation `CALL GEN_TED(tablespec, screenname, screentablename)`

<i>tablespec</i>	The name of the table (including its parameters, if any) that contains the text to appear and edited. This could be a predefined text table or another table that contains the same field names and syntaxes as @TEXT.
<i>screenname</i>	The name of the screen to appear. This screen must be defined with the following features: <ul style="list-style-type: none"> • Horizontal scroll keys are 0. • One screen table must be defined to meet the requirements of the <i>screentablename</i> argument given below. • The screen table PFKEY_SPECS must be included. • There must be a table instance of PFKEYS(<i>screen</i>). It can be copied from PFKEYS(@GEN_TED) and occurrences can be added and deleted as required.
<i>screentablename</i>	The name of the screen table that is manipulated by the text editor. It must contain the fields listed below. If an application does not specifically require some of the fields, they can be defined as SHOW = N, PROT = Y or be given a value of ROW = 0, COL = 0. This removes them from the physical screen. Even though the fields do not appear, other fields cannot be defined to the same position.

Usage Notes Requirements for Screen Table

The fields listed here must be in the screen table, in the titles area:

TABlename	This field must be long enough to contain <i>tablespec</i> . It must be TYP = S, SYN = C.
COMMAND	This field can be used for primary commands, if required. It must be TYP = S, SYN = V.

CMND	<p>This field is used to accept the line command I, if required. You use this line command to insert data at the top of the text. It should be positioned directly above the field CMD. It must be TYP = S, SYN = C, LEN = 1. If you use the FILL = _ (underscore), this field looks like the other line command field, CMD.</p> <p>The scrollable part of the screen table <i>must</i> also contain a line with these fields:</p>
CMD	<p>This is used for line commands. It must be TYP = S, SYN = V, LEN = 1. Set PROT = Y to prevent users from typing in this field before the line is available for text input. GEN_TED unprotects this field as needed.</p>
LINE	<p>This field holds the line of text. It must be TYP = S, SYN = V. Set PROT = Y to prevent users from typing in this field before the line is available for text input. GEN_TED unprotects this field as needed.</p>
NUMBER	<p>This holds the number for each line. It can be defined as SHOW = N, PROT = Y. This removes it from the physical screen. Even though the field does not appear, another field cannot be defined to the same position. It must be TYP = C, SYN = B, LEN = 4.</p>

Requirements for Text Table

A text table can be parameterized and it *must* contain these two fields:

NUMBER	A primary key field with a length of 4, a syntax of B, and a semantic type of I.
LINE	A field with a syntax of V and a semantic type of S.

Other Requirements

- The local variable MSG must be declared by the calling rule.
- The exception handler TED_ERROR must be provided. In the case of error, GEN_TED raises the TED_ERROR exception and stores the message in the MSG variable.

- The occurrence SAVE in the table PFKEYS(screen) *must* call a rule that ends by calling GEN_EDSAVER. GEN_EDSAVER saves the information entered to the text screen table.

Example The following example presents a screen that accepts text editing. The example is composed of:

- The field definition of the screen table PROJ_WEEK_STAT_1, from the screen WEEKLY_STATUS_G
- The GEN_TED_1 rule

Definition of PROJ_WEEK_STAT_1 Screen Table

The fields of the screen table PROJ_WEEK_STAT_1 are defined as follows:

Table: PROJ_WEEK_STAT_1 Unit: USR40															
ROW	COL	Field Name	Type	Syn	Len	Dec	Just	Fill	Prot	Show	Rqd	Hi	Skip	Null	
1	24	TABLENAME	S	C	40	0	L		Y	N	N	N	Y	Y	
1	67	COMMAND	S	C	1	0	L		Y	N	N	N	Y	Y	
2	3	CMND	S	C	1	0	L		N	Y	N	N	Y	Y	
3	3	CMD	S	C	1	0	L		Y	Y	N	N	Y	Y	
3	5	LINE	S	V	68	0	L		Y	Y	N	N	Y	Y	
3	74	NUMBER	C	B	5	0	L		Y	N	N	N	Y	Y	

Rule Using GEN_TED

The example rule uses the screen WEEKLY_STATUS_G and is as follows:

```
GEN_TED_1(PROJECT);
  _ LOCAL SCREEN, MSG;
  _
  -----
  |
  |
  | SCREEN = 'WEEKLY_STATUS_G';
  | PFKEY_SPECS.PCKEYS = PFKEY_MSG(SCREEN);
  | INSERT PFKEY_SPECS(SCREEN);
  | CALL GEN_TED('@TEXT(STATUS, ' || PROJECT ||')',
  | SCREEN, 'PROJ_WEEK_STAT_1');
  |
  -----
  _ ON TED_ERROR :
  _ CALL ENDMSG(MSG);
```

Output for the GEN_TED_1(PROJECT) Rule

After the rule executes, the following screen appears:

Project:	Weekly Status Report	Date:
Status:	Title:	
For the week of:		
Project Description		
* * * * *		
—		
—		
PFKEYS: 12=QUIT 22=DELETE 3=SAVE 5=SCRIPT 9=REPEAT CMND		

GENBIN

Returns a syntax V string containing the same internal binary value as the input numeric value, right-justified.(F)

Invocation `string = GENBIN(value, length)`

<code>string</code>	Syntax V string returned by the function.
---------------------	---

<code><i>value</i></code>	Input syntax B numeric value.
---------------------------	-------------------------------

<code><i>length</i></code>	Desired length of string returned.
----------------------------	------------------------------------

- Usage**
- Can be used to generate an internal buffer of arbitrary bytes.
 - Each byte of the returned string is initialized to 0 and the input value is moved into the string right-justified.
 - If the numeric value cannot fit into the specified length, it is truncated on the left.
 - Length must not be greater than the maximum number of digits for a BINARY value.
 - The returned string could display unprintable characters as either blanks or symbols.

Examples The following rule uses GENBIN to create strings of unprintable bytes:

RULE EDITOR ==>>> A(V,L); _ LOCAL STR; _ ----- _ STR = GENBIN(V,L); _ CALL ENDMSG(STR ' ; LENGTH OF STRING=' LENGTH(STR)); _ -----	SCROLL: P + 1 2
--	--

A(210,5) results in a 5-byte STR buffer, the first 4 bytes are 0, the fifth byte is 210. When STR appears in the message log, unprintable text displays either as blanks, or assorted symbols. In this instance, STR displays in the message log as “ K” (210 is EBCDIC code for the printable character K).

A(0,8) results in an 8-byte STR buffer, the eight bytes are 0 and display in the message log as blanks.

The following rule uses GENBIN(0,1) to create 1-byte of null to be used in PEEL for peeling:

```

      RULE EDITOR ==>
PEELBLANKANDNULL(MYSTRING);
      SCROLL: P
-
- -----
- -----+-----
- PEEL(' ' || GENBIN(0, 1), MYSTRING);      | 1
- -----

```

GENERATE_REPORT

Defines a new report or modifies an existing report using the Report Generator.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	GR generate report option	Type <i>reportname</i> <Enter>
	EX execute rule option	Type GENERATE_REPORT (<i>reportname</i>) <Enter>
	COMMAND prompt	Type GR <i>reportname</i> <Enter>

Where:

<i>reportname</i>	The name of the report.
-------------------	-------------------------

See Also *TIBCO Object Service Broker Defining Reports* for information on generating reports.

GENFLOAT

Returns a syntax V string containing the same internal float representation as the input float value, left-justified. (F)

Invocation `string = GENFLOAT(value, length)`

<i>string</i>	Syntax V string returned by the function.
<i>value</i>	Input syntax F (float) value.
<i>length</i>	Desired length of string returned (must be 4, 8, or 16).

- Usage**
- Can be used to generate an internal buffer containing internal IBM float values.
 - Each byte of the returned string is initialized to 0 and the input floating point value is moved into the string buffer left-justified.
 - If the floating point value cannot fit into the specified length, it is truncated on the right.
 - The returned string could display unprintable characters as blanks or symbols.

Example The following rule uses GENFLOAT to create a buffer containing float values.

RULE EDITOR ==>>>		SCROLL: P
A1(V,L);		
_ LOCAL STR;		
_ -----		
_ -----		
_ STR = GENFLOAT(V,L);		1
_ CALL ENDMSG(STR '; LENGTH OF STRING=' LENGTH(STR));		2
_ -----		

A1(1.23456789e2,8) results in an 8-byte STR buffer containing internal floating point representation for 123.456789.

A1(0,16) results in a 16-byte STR buffer of 0 (that displays as blanks).

GENPACK

Returns a syntax V string containing the same internal packed decimal value as the input syntax P value, right-justified. (F)

Invoking `string = GENPACK(value, length, decimal)`

<i>string</i>	Syntax V string returned by the function.
<i>value</i>	Input syntax P (packed) value.
<i>length</i>	Desired length of string returned.
<i>decimal</i>	Desired decimal places (currently must be 0) are ignored and treated as 0.

- Usage**
- Can be used to generate an internal buffer of IBM packed decimal value.
 - Each byte of the returned string is initialized to 0 and the input packed decimal representation is moved into the string right-justified.
 - If the packed value cannot fit into the specified length, it is truncated on the left, resulting in loss of significance.
 - Length must not be greater than the maximum number of digits for a packed value.
 - The returned string could display unprintable characters as blanks or symbols.

Example The following rule uses GENPACK to create a buffer containing packed values.

<pre> RULE EDITOR ==> A2(V,L); _ LOCAL STR; _ ----- _ _ STR = GENPACK(V,L,0); _ CALL ENDMSG(STR ' ; LENGTH OF STRING=' LENGTH(STR)); _ ----- </pre>	<pre> SCROLL: P +-----+ 1 2 +-----+ </pre>
--	--

A2(-0.003,5) results in a 5-byte STR buffer. The first 4 bytes are 0 (and display as blanks), and the fifth byte is x'3D' (a hexadecimal number that displays as "."), which is the internal packed representation for -3.

A2(0,7) results in a 7-byte STR buffer. The first 6 bytes are 0 (and display as blanks), and the seventh byte is x'0C' (a hexadecimal number that displays as “.”).

\$GETCONTAINER

Retrieves data from the specified channel container. (C)

Invocation `CALL $GETCONTAINER(channel, container, area, length, intoccsid)`

<i>channel</i>	The name (1-16 characters) of the channel that owns the container.
<i>container</i>	The name (1-16 characters) of the container that holds the data to be retrieved.
<i>area</i>	The pointer to an area large enough in which to place the retrieved data.
<i>length</i>	A fullword binary value. This is both an input and an output field: <ul style="list-style-type: none"> On input, <i>length</i> specifies the maximum length of the data that the rule accepts. If the actual data length exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On output (that is, on completion of the retrieval operation), <i>length</i> specifies the actual data length in the container. If the container holds character data that has been converted from one Coded Character Set Identifier (CCSID) to another, <i>length</i> is the data length <i>after conversion</i>.
<i>intoccsid</i>	A fullword binary number that represents the Coded Character Set Identifier (CCSID) into which the character data in the container is to be converted. For an explanation of CCSIDs, see the section “Data Conversion With Channels” in the <i>CICS Transaction Server for z/OS CICS Application Programming Guide</i> .

Example Following is a sample rule:

```

RULE EDITOR ==>
C_$GETCONTAINER(CHANNEL, CONTAINER);
_ LOCAL CONTENT, LEN, TOCCSID;
_
_ -----+-----
_ @MAP.ADDRESS = 0;                                | 1
_ @MAP.SIZE = 128;                                  | 2
_ INSERT @MAP('ENVIRONMENT');                       | 3
_ CONTENT = @MAP.ADDRESS;                           | 4
_ LEN = @MAP.SIZE;                                  | 5
_ CCSID = '';                                        | 6
_ CALL MSGLOG(                                       | 7
_   '$GETCONTAINER(CHANNEL, CONTAINER, CONTENT, LEN, TOCCSID)');
_ CALL MSGLOG('Channel name is: ==> ' || CHANNEL);  | 8
_ CALL MSGLOG('Container name is: ==> ' || CONTAINER); | 9
_ CALL $GETCONTAINER(CHANNEL, CONTAINER, CONTENT, LEN, TOCCSID); | A
_ GET CCN_CICSRESP(CONTENT);                         | B

```



```

_ CALL MSGLOG('Container ' || CONTAINER || ' content:');
_ CALL MSGLOG(' ' || CCN_CICSRESP.CODES);

```

Following is the MAP table CCN_CICSRESP:

COMMAND==>

TABLE DEFINITION

Table: CCN_CICSRESP					Type: MAP	Unit: HZS80	IDgen: Y				
Parameter Name	Typ	Syn	Len	Dc	Cls	Reference		Event	Rule	Typ	Acc
<hr/>											
ADDRESS		B	4	0	A						
LOCATION	I	C	16	0	L						
<hr/>											
Field Name		EXTERNAL				Metadata Definition					
	Xsyn	Xlen	Xdec	Offset	Key	Typ	Syn	Len	Dec	Rqd	Default
<hr/>											
KEY	B		4	0	0	P	I	B	4	0	
CODES	C		36	0	0			C	36	0	

\$GET_MAXSIZE

Retrieves the dictionary size of an expression. (F)

Invocation `result = $GET_MAXSIZE(value)`

<i>result</i>	The dictionary size of an expression. This argument is typeless, two bytes long, and of binary syntax with 0 decimal places.
---------------	--

<i>value</i>	The expression to be evaluated. This argument can be any semantic type, syntax, and size.
--------------	---

Example Rule

<pre> RULE EDITOR ==> GET_SPEC(TABLE, FIELD); _ LOCAL A, B, C, D, E; _ _ ----- _ GET TABLE; _ A = \$GET_SIZE((TABLE).(FIELD)); _ B = \$GET_TYPE((TABLE).(FIELD)); _ C = \$GET_SYNTAX((TABLE).(FIELD)); _ D = \$GET_DECIMALS((TABLE).(FIELD)); _ E = \$GET_MAXSIZE((TABLE).(FIELD)); _ CALL ENDMSG(TABLE ' ' FIELD ' ' (TABLE).(FIELD) _ ' ' A ' ' B ' ' C ' ' D ' ' _ E); _ ----- </pre>		<pre> SCROLL: P 1 2 3 4 5 6 7 </pre>
---	--	--------------------------------------

Output

Running GET_SPEC(TABLES, NAME) could produce:

```
TABLES NAME $$ACCESSLOG 11 I C 0 16
```

Explanation

GET_SPEC uses the \$GET_MAXSIZE tool, as well as \$GET_DECIMALS, \$GET_SIZE, \$GET_SYNTAX, and \$GET_TYPE, to get information about the field of a table.

You can also use these shareable tools to dump out the metadata information on the values involved in an error that you trapped while debugging a rule.

\$GET_SIZE

Retrieves the size of an expression. (F)

Invocation `result = $GET_SIZE(value)`

<i>result</i>	The size of an expression. This argument is typeless, two bytes long, and of binary syntax with 0 decimal places.
---------------	---

<i>value</i>	The expression to be evaluated. This argument can be any semantic type, syntax, and size.
--------------	---

Example Rule

<pre> RULE EDITOR ==> GET_SPEC(TABLE, FIELD); _ LOCAL A, B, C, D, E; _ _ ----- _ GET TABLE; _ A = \$GET_SIZE((TABLE).(FIELD)); _ B = \$GET_TYPE((TABLE).(FIELD)); _ C = \$GET_SYNTAX((TABLE).(FIELD)); _ D = \$GET_DECIMALS((TABLE).(FIELD)); _ E = \$GET_MAXSIZE((TABLE).(FIELD)); _ CALL ENDMSG(TABLE ' ' FIELD ' ' (TABLE).(FIELD) _ ' ' A ' ' B ' ' C ' ' D ' ' _ E); _ ----- </pre>		<pre> SCROLL: P 1 2 3 4 5 6 7 </pre>
---	--	--------------------------------------

Output

Running GET_SPEC(TABLES, NAME) could produce:

```
TABLES NAME $$ACCESSLOG 11 I C 0 16
```

Explanation

GET_SPEC uses the \$GET_SIZE tool, as well as \$GET_DECIMALS, \$GET_MAXSIZE, \$GET_SYNTAX, and \$GET_TYPE, to get information about the field of a table.

You can also use these shareable tools to dump out the metadata information on the values involved in an error that you trapped while debugging a rule.

\$GET_SYNTAX

Retrieves the syntax of an expression. (F)

Invocation `result = $GET_SYNTAX(value)`

<i>result</i>	The syntax of an expression. This argument is typeless, three bytes long, and of character syntax.
---------------	--

<i>value</i>	The expression to be evaluated. This argument can be any semantic type, syntax, and size.
--------------	---

Example Rule

<pre> RULE EDITOR ==> GET_SPEC(TABLE, FIELD); _ LOCAL A, B, C, D, E; _ _ ----- _ GET TABLE; _ A = \$GET_SIZE((TABLE).(FIELD)); _ B = \$GET_TYPE((TABLE).(FIELD)); _ C = \$GET_SYNTAX((TABLE).(FIELD)); _ D = \$GET_DECIMALS((TABLE).(FIELD)); _ E = \$GET_MAXSIZE((TABLE).(FIELD)); _ CALL ENDMSG(TABLE ' ' FIELD ' ' (TABLE).(FIELD) _ ' ' A ' ' B ' ' C ' ' D ' ' _ E); _ ----- </pre>	<pre> SCROLL: P 1 2 3 4 5 6 7 </pre>
---	--------------------------------------

Output

Running GET_SPEC(TABLES, NAME) could produce:

```
TABLES NAME $$ACCESSLOG 11 I C 0 16
```

Explanation

GET_SPEC uses the \$GET_SYNTAX tool, as well as \$GET_DECIMALS, \$GET_MAXSIZE, \$GET_SIZE, and \$GET_TYPE, to get information about the field of a table.

You can also use these shareable tools to dump out the metadata information on the values involved in an error that you trapped while debugging a rule.

\$GET_TYPE

Retrieves the semantic type of an expression. (F)

Invocation `result = $GET_TYPE(value)`

<i>result</i>	The semantic type of an expression. This argument is typeless, one byte long, and of character syntax.
---------------	--

<i>value</i>	The expression to be evaluated. This argument can be any semantic type, syntax, and size.
--------------	---

Example Rule

<pre> RULE EDITOR ==> GET_SPEC(TABLE, FIELD); _ LOCAL A, B, C, D, E; _ _ ----- _ GET TABLE; _ A = \$GET_SIZE((TABLE).(FIELD)); _ B = \$GET_TYPE((TABLE).(FIELD)); _ C = \$GET_SYNTAX((TABLE).(FIELD)); _ D = \$GET_DECIMALS((TABLE).(FIELD)); _ E = \$GET_MAXSIZE((TABLE).(FIELD)); _ CALL ENDMMSG(TABLE ' ' FIELD ' ' (TABLE).(FIELD) _ ' ' A ' ' B ' ' C ' ' D ' ' _ E); _ ----- </pre>	<pre> SCROLL: P 1 2 3 4 5 6 7 </pre>
--	--------------------------------------

Output

Running GET_SPEC(TABLES, NAME) could produce:

```
TABLES NAME $$ACCESSLOG 11 I C 0 16
```

Explanation

GET_SPEC uses the \$GET_TYPE tool, as well as \$GET_DECIMALS, \$GET_MAXSIZE, \$GET_SIZE, and \$GET_SYNTAX, to get information about the field of a table.

You can also use these shareable tools to dump out the metadata information on the values involved in an error that you trapped while debugging a rule.

\$GETATTRIBUTE

Queries the current attributes for the field of the screen table, in the specified screen. (F)

Invocation `flag = $GETATTRIBUTE(screen, table, field, attribute)`

<i>flag</i>	One of the following: Y – The attribute is on. N or anything except Y – The attribute is off.
<i>screen</i>	A character string specifying the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	A character string specifying the screen table. Its syntax is C with length 16.
<i>field</i>	A character string specifying the screen field. Its syntax is C with length 16.
<i>attribute</i>	The following attributes are available to query: P – The field is protected. H – The field is highlighted. V – The field is visible. D – The field is light-pen-detectable. The data-mapped graphical object is detectable by cursor (allow focus). N – The field accepts only numeric characters. U – The field is underlined (extended attribute). R – The foreground and background colors of the field are reversed (extended attribute). B – The field blinks (extended attribute).

- Usage Notes**
- Not all display devices can support the extended attributes. The extended attributes are supported only on some 3270 terminals on a z/OS system.
 - Valid values must be supplied for *all* the arguments or an error occurs.

- \$GETATTRIBUTE operates only on a screen table that has real occurrences, and has the current position set by a table access. If the current position is not set in the screen table, \$GETATTRIBUTE returns an empty string.
- For a particular screen field, specify only one of the extended attributes U, R, or B.

Example The following rule queries the highlighting attribute of the FCNKEYS field of the example screen, displays the screen, and reports the value of the highlighting attribute:

```
GETATTRIBUTE_1;
_  LOCAL HIGH;
_  -----
_  -----+-----
_  FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');           | 1
_  INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                           | 2
_  HIGH = $GETATTRIBUTE('NEW_EMPLOYEE', 'FCNKEY_SPECS',          | 3
_  'FCNKEYS', 'H');                                               |
_  DISPLAY NEW_EMPLOYEE:                                         | 4
_  CALL ENDMSG('HIGHLIGHTING': || HIGH);                          | 5
_  -----
```


\$GETBINARY

Stores character data in binary format. (F)

Invocation `result = $GETBINARY(string, offset, length)`

<i>result</i>	Binary number returned by the rule. Its syntax is B (binary).
<i>string</i>	The character string containing the number to be stored in binary format. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>offset</i>	The offset from which to start converting the string. It must be greater than or equal to zero (0), and smaller than the length of the string.
<i>length</i>	The number of characters to convert. It must be less than or equal to 4.

- Usage**
- Used to extract binary numbers from a string that contains these numbers in binary format.
 - The sum of the length and the offset must be less than or equal to the length of the string.

Examples Table SOURCE is an import table (IMP) mapped to an external file with a record length of 56. Each record of this external file contains numeric data in binary, packed, and floating point formats:

```

COMMAND ==>
      Table: SOURCE
      File: userid.dataset
      DDname:
      Server ID:
      Parameter Name  Typ  Syn  Len  Dec  Class  Src
      -----
      Table: SOURCE  Type: IMP  Unit: USERID  IDgen: N
      External Routine Name:
      Server ID:
      Parameter Name  Typ  Syn  Len  Dec  Class  Src
      -----
      Field Name      Xsyn Xlen  Xdec  Offset  Key  Typ  Syn  Len  Dec  Ord  Rqd  Default
      -----
      RECORD          V      56    0      0      P   S   V   56    0
  
```

Table TARGET is a TDS table to which this external data is to be copied:

COMMAND ==>						TABLE DEFINITION						
Table: TARGET				Type: TDS			Unit: USERID			IDgen: Y		
Parameter Name		Type	Syn	Len	Dec	Class				'	Event Rule	Type Acc
		-	-	--	--	-					-----	- -
										:		
										:		
										:		
Field Name		Type	Syn	Len	Dec	Key	Ord	Rqd		Default		Reference
		-	-	--	--	-	-	-		-----		-----
KEY		1	B	4	0	P						
BIN1		C	B	4	0							
BIN2		C	B	4	0							
PACK1		Q	P	5	0							
PACK2		Q	P	8	0							
PACK3		Q	P	3	0							
FLOAT1		Q	F	8	0							
FLOAT2		Q	F	8	0							
FLOAT3		Q	F	16	0							

The following rule uses \$GETBINARY to copy binary values from an external file to an TIBCO Object Service Broker table:

COPYNUMDATA;

FORALL SOURCE :	1
TARGET.BIN1=\$GETBINARY(SOURCE.RECORD,0,4);	
TARGET.BIN2=\$GETBINARY(SOURCE.RECORD,4,4);	
TARGET.PACK1=\$GETPACKED(SOURCE.RECORD,8,5);	
TARGET.PACK2=\$GETPACKED(SOURCE.RECORD,13,8);	
TARGET.PACK3=\$GETPACKED(SOURCE.RECORD,21,3);	
TARGET.FLOAT1=\$GETFLOAT(SOURCE.RECORD,24,8);	
TARGET.FLOAT2=\$GETFLOAT(SOURCE.RECORD,32,8);	
TARGET.FLOAT3=\$GETFLOAT(SOURCE.RECORD,40,16);	
INSERT TARGET;	
END;	

GETCHAR

Returns the first character from the specified string, removing it from the string.
(F)

Invocation `character = GETCHAR(string)`

<i>character</i>	On return contains the character.
<i>string</i>	The string from which to return the character.

Syntax and Length of Arguments

String Syntax	Character Syntax	Length
C (fixed-length character string)	V	1
RD (raw data)	RD	5
UN (Unicode)	UN	2 or 4
V (variable-length character string)	V	1

- Usage Notes**
- If *string* is null, a null string is returned.
 - *string* must be a field of a table or a local variable. It cannot be a constant value because *string* is changed by GETCHAR.

Example The following rule gets the first character from a string, gets the next remaining character, and prints the appropriate characters and strings to the message log:

```

GETCHAR_1;
_ LOCAL SOURCE_STRING, STRIP_CHAR;
_ -----
_ -----+-----
_ SOURCE_STRING = 'THIS IS THE SOURCE STRING';          | 1
_ STRIP_CHAR = GETCHAR(SOURCE_STRING);                   | 2
_ CALL MSGLOG('THE FIRST STRIPPED CHARACTER IS: ' ||     | 3
_ STRIP_CHAR);
_ CALL MSGLOG('THE SOURCE STRING IS NOW: ' || SOURCE_STRING | 4
_ );
_ STRIP_CHAR = GETCHAR(SOURCE_STRING);                   | 5
_ CALL MSGLOG('THE SECOND STRIPPED CHARACTER IS: ' ||    | 6
_ STRIP_CHAR);

```

```
_ CALL MSGLOG('THE SOURCE STRING IS NOW: ' || SOURCE_STRING | 7
_ );
_ -----
```

Output for the GETCHAR_1 Rule

Pressing PF2 after executing this rule displays the following on the screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE FIRST STRIPPED CHARACTER IS: T
THE SOURCE STRING IS NOW: HIS IS THE SOURCE STRING
THE SECOND STRIPPED CHARACTER IS: H
THE SOURCE STRING IS NOW: IS IS THE SOURCE STRING
```

\$GETCOLOUR

Queries the current color of a screen field. (F)

Invocation `color = $GETCOLOUR(screen, table, field, color_type)`

<i>color</i>	A character string specifying the color to be used. Valid values are any color in the @COLOURS table. Its syntax is C with length 25.
<i>screen</i>	A character string specifying the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	A character string specifying the screen table. Its syntax is C with length 16.
<i>field</i>	A character string specifying the screen field. Its syntax is C with length 16.
<i>color_type</i>	A character string specifying whether the color is to be foreground or background. Valid values: F – Foreground. B – Background.

- Usage Notes**
- Not all display devices can support background color. If your display device does not support background color, the specification is ignored.
 - You must supply valid values for all the arguments or an error occurs.
 - \$GETCOLOUR operates only on a screen table that has real occurrences and has the current position set by a table access. If the current position is not set in the screen table, \$GETCOLOUR returns an empty string.

Example The following rule queries the color of the FCNKEYS field of the example screen, displays the screen, and reports the color of the field:

```

GETCOLOUR_1;
_ LOCAL COLOUR;
_ -----
_ -----+-----
_ FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');          | 1
_ INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                          | 2
_ COLOUR= $GETCOLOUR('NEW_EMPLOYEE', 'FCNKEY_SPECS',           | 3
_   'FCNKEYS', 'F',);                                          |
_ DISPLAY NEW_EMPLOYEE:                                       | 4
_ CALL ENDMSG('COLOUR IS ' || COLOUR);                          | 5

```


GETENDMSG

Returns the current value of the end-of-transaction message. (F)

Invocation `message = GETENDMSG`

`message` On return, contains the message. Its syntax is V (character) with length 148.

Usage Notes Each transaction starts with an empty end-of-transaction message.

Example The set of rules in this example:

1. Validates a value
2. Uses the end-of-transaction message to pass a value for validation
3. Takes a subsequent action that is dependent on the value returned to the end-of-transaction message

Sample Rule 1

RULE EDITOR ==>	SCROLL: P
DELETE_EMPNO(REGION, EMPNO);	
—	
— -----	
—	+
— EXECUTE VALIDATE_EMP(REGION, EMPNO);	1
— CALL VALID_EMPNO(REGION, EMPNO);	2
— -----	

Sample Rule 2

RULE EDITOR ==>	SCROLL: P
VALIDATE_EMP(REGION, EMPNO);	
—	
— -----	
—	+
— GET EMPLOYEE(REGION) WHERE EMPNO = EMPNO;	1
— CALL ENDMSG('YES');	2
— -----	
— ON GETFAIL :	
— CALL ENDMSG('INVALID EMPLOYEE NUMBER');	

Rule Using GETENDMSG

RULE EDITOR ==>		SCROLL: P
VALID_EMPNO(REGION, EMPNO);		

GETENDMSG = 'YES';	Y N	

DELETE EMPLOYEE(REGION) WHERE EMPNO = EMPNO;	1	
CALL ENDMSG('THE EMPLOYEE IS DELETED');	2	

Executing this rule displays the following screen:

USR40		TEST: N BROWSE: N	12:48 AM	THURSDAY MAR 26 2000
ER edit rule	==>			SU MO TU WE TH FR SA
EX execute rule	==>	DELETE_EMPNO		1 2 3 4 5 6 7
DB debug rule	==>			8 9 10 11 12 13 14
BR browse table	==>			15 16 17 18 19 20 21
ED edit table	==>			22 23 24 25 26 27 28
				29 30 31
OS object set	==>			
DS define screen	==>			
DR define report	==>			
DT define table	==>			
DL define library	==>			
GR generate rpt	==>			
COMMAND ==>	__			
execute rule:	EX DELETE_EMPNO			1
PFKEYS: 2=LOGS 3=EXIT 12=EXIT				
12:48:55 THE EMPLOYEE IS DELETED				

The new end of transaction message is printed on the bottom row and the employee is deleted from the employee table.

\$GETENVCOMMAREA

Retrieves data passed into TIBCO Object Service Broker from a calling environment that is not TIBCO Object Service Broker. (F)

Invocation `value = $GETENVCOMMAREA(segment#)`

<code>value</code>	On return, contains the retrieval data. Its syntax is V (variable-length character string) with length of either 8,192 or 31,744 depending on retrieval data length.
<code>segment#</code>	The number of the segment from where to retrieve the data. Its syntax is B (binary) with length 2.

- Usage Notes**
- For IMS TM, use the following values to indicate the segment number:
 - 0 – Scratch Pad Area (SPA).
 - 1 – The user data or the session parameter string if using a non-seamless interface.
 - 2 – The user data.
 - For CICS, the value for `segment#` is ignored.
 - For the SDK (C/C++), the maximum length for `value` applies even though much larger COMMAREAs are supported through the MAP table interface.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for information about using TIBCO Object Service Broker with external environments

TIBCO Object Service Broker Managing Data for information about MAP tables

Example The following rule retrieves the data from segment 0.

```

RULE EDITOR ==>
GETENV_1;
_ LOCAL COMMAREA;
-----
_                                     +-----+
_ COMMAREA = $GETENVCOMMAREA(0);      | 1
_                                     -----

```

\$GETFLOAT

Stores character data in floating format. (F)

Invocation `result = $GETFLOAT(string, offset, length)`

<i>result</i>	Floating point number returned by the rule. Its syntax is F (float).
<i>string</i>	The character string containing the number to be stored in floating point format. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>offset</i>	The offset from which to start converting the string. It must be greater than or equal to zero (0) and smaller than the length of the string.
<i>length</i>	The number of characters to convert. It must be 4, 8, or 16.

- Usage**
- Used to extract floating point numbers from a string that contains these numbers in internal floating point format.
 - The sum of the length and the offset must be less than or equal to the length of the string.

Example Table SOURCE is an import table (IMP) mapped to an external file with a record length of 56. Each record of this external file contains numeric data in binary, packed, and floating point formats:

```

COMMAND ==>
      Table: SOURCE          Type: IMP      Unit: USERID          IDgen: N
      File: userid.dataset
      DDname:                External Routine Name:

      Parameter Name Type Syn Len Dec Class  Src          ' Event Rule Typ Acc
      -----
      |-----IMP-----|-----MetaStor Definition-----
      Field Name      Xsyn Xlen Xdec Offset Key Typ Syn Len Dec Ord Rqd Default
      -----
RECORD              V      56      0          0      P      S      V      56      0
      |-----
      |-----
      |-----

```

Table TARGET is a TDS table to which this external data is to be copied:

COMMAND ==>										TABLE DEFINITION			
Table: TARGET					Type: TDS		Unit: USERID			IDgen: Y			
Parameter Name		Type	Syn	Len	Dec	Class				Event	Rule	Typ	Acc
-----		-	-	---	--	-				:	-----	-	-
										:	-		
										:	-		
Field Name		Type	Syn	Len	Dec	Key	Ord	Rqd	Default	Reference			
-----		-	-	---	--	-	-	-	-----	-----			
KEY		1	B	4	0	P							
BIN1		C	B	4	0								
BIN2		C	B	4	0								
PACK1		Q	P	5	0								
PACK2		Q	P	8	0								
PACK3		Q	P	3	0								
FLOAT1		Q	F	8	0								
FLOAT2		Q	F	8	0								
FLOAT3		Q	F	16	0								

The following rule uses \$GETFLOAT to copy floating point values from an external file to an TIBCO Object Service Broker table:

```
COPYNUMDATA;

-
- -----
- -----+-----
- FORALL SOURCE:                                | 1
-   TARGET.BIN1=$GETBINARY(SOURCE.RECORD,0,4);
-   TARGET.BIN2=$GETBINARY(SOURCE.RECORD,4,4);
-   TARGET.PACK1=$GETPACKED(SOURCE.RECORD,8,5);
-   TARGET.PACK2=$GETPACKED(SOURCE.RECORD,13,8);
-   TARGET.PACK3=$GETPACKED(SOURCE.RECORD,21,3);
-   TARGET.FLOAT1=$GETFLOAT(SOURCE.RECORD,24,8);
-   TARGET.FLOAT2=$GETFLOAT(SOURCE.RECORD,32,8);
-   TARGET.FLOAT3=$GETFLOAT(SOURCE.RECORD,40,16);
-   INSERT TARGET;
-   END;
- -----
```

\$GETOPT

Returns the value of a session parameter or option. (F)

Invocation `value = $GETOPT(option_name)`

<code>value</code>	On return, contains the value. Its syntax is C (fixed-length character string) with length 16.
<code>option_name</code>	The parameter or option from which to return the value. Its syntax is C with length 16. Valid values for <code>option_name</code> are shown below.

Available Parameters and Options

Only the parameters and options identified are accepted by this tool.

Parameter or Option Name	Abbreviation	Description	Open Systems	z/OS
All parameters that can be set with \$SETOPT , except LANGUAGE, can be retrieved with \$GETOPT. The following can also be retrieved:				
COMMITSIZE		The maximum size of the intent list.	Y	Y
COMMITUSED		The current size of the intent list.	Y	Y
KERNEL	K	The Execution Environment type: BATCH, CICS, CLI, IMS TM, NATIVE, NT, TSO, or UNIX.	Y	Y
LTERM	LT	The name of the logical terminal defined to IMS TM.	Y	Y
REGION	REGION	The IDPREFIX, that is, the IMS TM Control Region used by the present TIBCO Object Service Broker session.		Y

\$GETPACKED

Stores character data in packed decimal format. (F)

Invocation `result = $GETPACKED(string, offset, length)`

<i>result</i>	Packed decimal number returned by the rule. Its syntax is P (packed).
<i>string</i>	The character string containing the number to be stored in packed decimal format. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>offset</i>	The offset from which to start converting the string. It must be greater than or equal to zero (0), and smaller than the length of the string.
<i>length</i>	The number of characters to convert. It must be less than or equal to 16.

Usage

- Used to extract packed decimal numbers from a string that contains these numbers in packed decimal format.
- The sum of the length and the offset must be less than or equal to the length of the string.

Examples Table SOURCE is an import table (IMP) mapped to an external file with a record length of 56. Each record of this external file contains numeric data in binary, packed, and floating point formats:

```

COMMAND ==>                                     TABLE DEFINITION
      Table: SOURCE                               Type: IMP      Unit: USERID                               IDgen: N
      File: userid.dataset
      DDname:                               External Routine Name:

      Parameter Name  Type  Syn  Len  Dec  Class   Src                               ' Event Rule Typ Acc
      -----
      -
      -
      -
      Field Name      Xsyn  Xlen  Xdec  Offset  Key  Typ  Syn  Len  Dec  Ord  Rqd  Default
      -----
RECORD               V      56      0           0    P    S    V    56    0
      -
      -

```

Table TARGET is a TDS table to which this external data is to be copied:

COMMAND ==>										TABLE DEFINITION			
Table: TARGET										Type: TDS	Unit: USERID	IDgen: Y	
Parameter	Name	Type	Syn	Len	Dec	Class						Event	Rule
												Typ	Acc
Field Name	Type	Syn	Len	Dec	Key	Ord	Rqd	Default	Reference				
KEY	1	B	4	0	P								
BIN1	C	B	4	0									
BIN2	C	B	4	0									
PACK1	Q	P	5	0									
PACK2	Q	P	8	0									
PACK3	Q	P	3	0									
FLOAT1	Q	F	8	0									
FLOAT2	Q	F	8	0									
FLOAT3	Q	F	16	0									

The following rule uses \$GETPACKED to copy packed decimal values from an external file to an TIBCO Object Service Broker table:

```
COPYNUMDATA;
-
- -----
- FORALL SOURCE:
-   TARGET.BIN1=$GETBINARY(SOURCE.RECORD,0,4);
-   TARGET.BIN2=$GETBINARY(SOURCE.RECORD,4,4);
-   TARGET.PACK1=$GETPACKED(SOURCE.RECORD,8,5);
-   TARGET.PACK2=$GETPACKED(SOURCE.RECORD,13,8);
-   TARGET.PACK3=$GETPACKED(SOURCE.RECORD,21,3);
-   TARGET.FLOAT1=$GETFLOAT(SOURCE.RECORD,24,8);
-   TARGET.FLOAT2=$GETFLOAT(SOURCE.RECORD,32,8);
-   TARGET.FLOAT3=$GETFLOAT(SOURCE.RECORD,40,16);
-   INSERT TARGET;
-   END;
- -----
```

\$GETTRANSACTION

Gets a transaction name set by [\\$SETTRANSACTION](#). (F)

Invocation `current_value = $GETTRANSACTION(name)`

<code>current_value</code>	The string that contains the transaction name.
<code>name</code>	The field containing the name, which is set by \$SETTRANSACTION . NAME is the only valid field. Its syntax is C (fixed-length character string) with length 16.

Usage On z/OS, the transaction name is used when SMF records are produced.

Example The following rule uses [\\$SETTRANSACTION](#) to set a new transaction name, then gets the transaction name and displays it in the end message:

<pre> RULE EDITOR ==> RET_TRANS(TRANSNAME); _ LOCAL NEWNAME; _ ----- _ -----+----- _ NEWNAME = \$SETTRANSACTION('NAME', TRANSNAME); _ NEWNAME = \$GETTRANSACTION('NAME'); _ CALL ENDMSG(NEWNAME); _ ----- </pre>	<pre> SCROLL: P 1 2 3 </pre>
---	------------------------------

\$GTFSET

Enables or disables the rules tracing facility in the Execution Environment and the Data Object Broker.

This facility runs on the z/OS version of TIBCO Object Service Broker and writes a trace record to a Generalized Trace Facility (GTF) data set that can be used as input to the third-party product Trace Analyzer for Mainframe-built TIBCO Object Service Broker Rules (TAMBOR). (C)

Invocation `CALL $GTFSET(function, keyname[, userid, termid, all, dob])`

<i>function</i>	The action to be taken. Must be one of the following: SET – Enable the tracing facility. RESET – Disable the tracing facility.
<i>keyname</i>	A licence key is not required but, for compatibility with previous releases of TIBCO Object Service Broker, the value of this parameter must be 'TAMBOR'.
<i>userid</i>	[OPTIONAL] The user ID of the session to be traced. If there are duplicate user IDs, the tracing facility is enabled for the first one encountered. To ensure that the facility is enabled for one specific user, use the <i>termid</i> argument.
<i>termid</i>	[OPTIONAL] The terminal ID of the session to be traced.
<i>all</i>	[OPTIONAL] Enables or disables tracing for all users running under the current Execution Environment. If it is required, use 'ALL' for this parameter.
<i>dob</i>	[OPTIONAL] Enables or disables tracing in the Data Object Broker to which the Execution Environment is connected. If present, must be one of the following: ALSO – Enables or disables tracing in the Data Object Broker as well as the Execution Environment. ONLY – Enables or disables tracing in the Data Object Broker only.

Usage Notes

- The trace facility produces four types of GTF records. Each GTF record has its own User Specified Event ID code (USR) contained in the GTF trace record header. The USR codes are:
 - Rule start record (USR=FE)
 - Rule end record (USR=FF)
 - BUILTIN start record (USR=FC)
 - BUILTIN end record (USR=FD)
- The layout of every type of GTF record is as follow:

RULE/BUILTIN Name	CL16
TOD value when the record is created	XL8
Connection ID	CL8
User name	CL8
Accumulated CPU time in TOD format	XL8
Transaction stream number	XL1
Filler	XL3
Transaction ID	XL4

- For the *userid*, *termid*, and *all* parameters, \$GTFSET acts on only the first non-null argument it encounters. If no parameter is specified, the trace facility for the session running the \$GTFSET tool (the current session) is enabled or disabled.
- The IBM Generalized Trace Facility (GTF) must be activated with USR=(0AA,0FC,0FD,0FE,0FF) if Data Object Broker tracing is turned on, or with USR=(0FC,0FD,0FE,0FF) otherwise.

Example

The following rule sets the tracing facility on in the Execution Environment for the current session:

```
SET_GTF_TRACE;  
-----  
-----+-----  
CALL $GTFSET('SET', 'TAMBOR', '', '', '', ''); | 1  
-----
```

To set the trace on in the Execution Environment for the USERXYZ user, use the following call:

```
CALL $GTFSET('SET', 'TAMBOR', 'USERXYZ', '', '', '');
```

or

```
CALL $GTFSET('SET', 'TAMBOR', 'USERXYZ', 'TERM9999', 'ALL', '');
```

To set the trace on in the Execution Environment for the user whose terminal ID is TERM9999, use the following call:

```
CALL $GTFSET('SET', 'TAMBOR', '', 'TERM9999', '', '');
```

To set the trace on in the Execution Environment and in the Data Object Broker for all users, use the following call:

```
CALL $GTFSET('SET', 'TAMBOR', '', '', 'ALL', 'ALSO');
```

HEADSTRING

Returns the head portion of the specified string. (F)

Invocation `head = HEADSTRING(string, length)`

<i>head</i>	On return, contains the head of the string. Its syntax is the same as <i>string</i> .
<i>string</i>	The string from which to return the head portion. Its syntax can be C (fixed-length character string), UN (Unicode), V (variable-length character string), or W (double-byte character).
<i>length</i>	An integer specifying the number of leading characters to return. Its data type is B (binary) with length 4.

- Usage Notes**
- *string* is not modified.
 - If *length* is less than or equal to 0, an empty string is returned.
 - If *length* is greater than the length of *string*, the entire string is returned.

Example The following rule determines the first character of a string and prints both the character and the string to the message log:

```
HEADSTRING_1;
_ LOCAL SOURCE_STRING, HEAD_STRING;
```

```
-----+-----
_ SOURCE_STRING = 'THIS IS THE SOURCE STRING.'; | 1
_ HEAD_STRING = HEADSTRING(SOURCE_STRING, 1); | 2
_ CALL MSGLOG('THE HEAD STRING IS: ' || HEAD_STRING); | 3
_ CALL MSGLOG('THE SOURCE STRING IS STILL: ' || | 4
_ SOURCE_STRING); |
_ -----
```

Output for the HEADSTRING_1 Rule

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE HEAD STRING IS: T
THE SOURCE STRING IS STILL: THIS IS THE SOURCE STRING.
```


HLIPREPROCESSOR

Invokes a language pre-processor to run against COBOL source programs that contain embedded TIBCO Object Service Broker access statements or SQL statements. (C)

Invocation `CALL HLIPREPROCESSOR(hostlang, imbedlang, infile, outfile, listfile, options)`

<i>hostlang</i>	The name of the programming language. Type COBOL.
<i>imbedlang</i>	The type of statement embedded in the COBOL program. Use HURON or SQL.
<i>infile</i>	Type the name of the data set to be processed.
<i>outfile</i>	The name of the allocated, partitioned data set that is to contain the processed COBOL program. It can be passed to the COBOL compiler.
<i>listfile</i>	The name of a file to contain the listing of the output, if a listing is to be produced. It is not compiled. The argument <i>listfile</i> must be null if <i>imbedlang</i> is SQL.
<i>options</i>	<p>A string specifying the options:</p> <p>LIST – A list is to be produced.</p> <p>NO LIST – No list is to be produced.</p> <p>ERRORSTOP – If an error is detected, this causes the preprocessor to fail and raise an exception.</p> <p>NO ERRORSTOP – This prevents the exception from being raised.</p>

One of the LIST and NOLIST options, and one of the ERRORSTOP and NOERRORSTOP options, must be specified. The argument *options* must be null if *imbedlang* is SQL.

See Also *TIBCO Object Service Broker for z/OS External Environments* for information about using HLIPREPROCESSOR.

Exceptions

STOP_AT_ERROR	Raised if an error in processing is detected. In a batch job, it raises a completion code that prevents the next job step from executing.
----------------------	---

HOUR

Returns the hour of the day when the current transaction started based on the local machine's time zone in which the Execution Environment is running. (F)

Invocation time = HOUR

time	On return, contains the hour. Its syntax is C (fixed-length character string) with length 2.
------	--

Usage Notes The returned value is a string containing the number of the hour (00-23).

Example The following rule determines the hour and prints it to the message log:

```

RULE EDITOR ==>
  HOUR_1;
  _ LOCAL TIME;
-----
-
-
- TIME = HOUR;
- CALL MSGLOG('THIS TRANSACTION WAS STARTED AFTER ' || TIME | 1
-   || ':00. ');
-   | 2
-   |
-----

```

Output for the HOUR_1 Rule:

Pressing PF2 after executing this rule displays the following on the screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THIS TRANSACTION WAS STARTED AFTER 13:00.
```


HURON_STATS

Displays statistics for performance analysis and problem determination. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	ST Statistics option	Press Enter
	EX Execute Rule option	Type HURON_STATS <Enter>
Developer's workbench	EX Execute Rule option	Type HURON_STATS <Enter>
	COMMAND prompt	Type EX HURON_STATS <Enter>

Using any of these methods invokes the main menu shown here:

Statistics Main Menu

```

      DOB Statistics Menu
      -----
      _ Segment Statistics
      _ General Statistics
      _ Buffer Pool Statistics
      _ Users logged on
  
```

See *TIBCO Object Service Broker for z/OS Monitoring Performance* for descriptions of the statistics displayed from each of these menu selections.

\$HTTPREQUEST

Issues an HTTP request and returns the response code and result. (F)

Invocation `value = $HTTPREQUEST(requesttype, url, header, data, result, message)`

value	On return, contains the HTTP response code as an integer value. The return value has type C, syntax B, and length 4.
requesttype	The HTTP request type. Valid values are GET or POST.
url	The URL being requested. It has a length of 1024.
header	The HTTP header parameters for the HTTP request, in the form <i>name=value</i> . It has a length of 2048.
data	The data being sent to the HTTP server with the request. It has a length of 16384.
result	A variable used to contain the result of the HTTP request. It has a length of 16384.
message	A variable used to contain any error message resulting from the HTTP request. It has a length of 512.

Usage Notes

- A response code of 1000 indicates a non-HTTP error.
- If the header parameter value contains single or double quotation marks, each quotation mark must be repeated.
- The data parameter can be used to pass FORMAT input fields to an HTML form page. Form parameters look like query string parameters on a URL request. They are passed as a series of *name=value pairs*, each separated by an ampersand character.
For example, if a form has two parameters, USERID and PASSWORD, and your rule passes the constant string 'USERID=ME,PASSWORD=PWD' as the value of the data parameter, the HTML form receives 'ME' as the value for the 'USERID' parameter and 'PWD' as the value for the 'PASSWORD' parameter.



Take care when passing data using the data parameter of this rule. By default, the data is passed as plain text for GET requests, and as application/x-www-form-urlencoded data for POST requests.

Example

The following example shows how \$HTTPREQUEST can be called in a rule:

```
GETTEST();
- LOCAL HTTPCODE, URL, HEADER, DATA, RESULT, MESSAGE;
- -----
- -----+-----
- URL = 'HTTP://WWW.TIBCO.COM/POSTTEST.ASP?P1=1&P2=AAA'; | 1
- HTTPCODE = $HTTPREQUEST('GET', URL, HEADER, DATA, RESULT, MESSAGE); | 2
- CALL ENDMSG('RC=' || HTTPCODE || ', M=' || MESSAGE || ', R=' || RESULT); | 3
- -----
```

IDMS

Displays the main menu used to define a CA-IDMS database to TIBCO Object Service Broker. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type IDMS <Enter>
	COMMAND prompt	Type EX IDMS <Enter>

Executing IDMS displays the screen illustrated here:

```
MANAGER UTILITIES FOR IDMS DATA
-----

Enter an 'S' to select a function
_ List IDMS Tables
_ List IDMS Subschemas
_ Load an IDMS Subschema
_ Delete IDMS Subschemas

PFKEYS:  1=HELP  3=EXIT 12=EXIT
```

See Also *TIBCO Service Gateway for IDMS/DB Installing and Operating* for information on defining IDMS data to TIBCO Object Service Broker.

IMS

Displays the main menu used to define an IMS/DB database to TIBCO Object Service Broker. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type IMS <Enter>
	COMMAND prompt	Type EX IMS <Enter>

Executing IMS displays the screen illustrated here:

```
MANAGER UTILITIES FOR IMS DATA
-----
```

```
Enter an 'S' to select a function
```

- _ Import IMS Database Definitions
- _ List IMS Databases
- _ IMS DBD Extract Utility

```
PFKEYS:  1=HELP  3=EXIT  12=EXIT
```

See Also *TIBCO Service Gateway for IMS/DB Installing and Operating* for information on defining IMS/DB data to TIBCO Object Service Broker.

INDEXCHK

Estimates the maximum number of data rows a table can contain before reaching the maximum index levels. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type INDEXCHK <Enter>
	COMMAND prompt	Type EX INDEXCHK <Enter>

On the Index Level Checking screen:

- Specify the name of the table you want to analyze. This table must already be defined and be of type TDS.
- Specify either "Average Data Row length" or "Sample existing data".



If you select "Sample existing data", the table must have a significant number of rows for the result to be meaningful.

Usage Notes

If you want to sample existing data to estimate the average row length, you must have Level-7 security clearance.

For sequential inserts into a table or table instance with no secondary keys, INDEXCHK calculates the estimate assuming that all the tables pages (both data and indexes) are filled to their maximal value.

For random inserts, INDEXCHK calculates the estimate assuming that the top level of any index is filled to its maximal value and that all lower pages are sixty seven percent full. The result is only an estimate and may not correspond to the practical reality. Where a table is subject to random inserts and deletes, the estimate may be reduced further.

When secondary keys are defined, the estimate assumed that, even if the data is inserted in sequential order in regard to the primary key, the inserts are random in regard to any secondary keys. The secondary key providing the lowest number of rows is displayed.

Example The following screen shows the result of invoking INDEXCHK:

```

Index Level Checking:
=====
Application to estimate the Maximum # of Data Rows a table can

```

contain before Maximum Index levels are reached (PF1 for more details)

Table Name (must already be defined): INDEXCHK_DEMO
Average Data Row length : or Sample existing Data (Y/N): Y
#-Rows Sampled: 50000 Average Row Length: 41 #-Parms Sampled: 50
RESULTS:

Based on Sequential Inserts (giving maximal page utilisation)
Max Data Rows Limited by I-Index: 5,297,307,999,872
Max Data Rows Limited by S-Index: 3,001,948,715 (FIELD3)
Max Parameter Instances allowed : 66,999,227,038,614

Based on True Random Inserts
Max Data Rows Limited by I-Index: 741,305,536,999
Max Data Rows Limited by S-Index: 3,001,948,715 (FIELD3)
Max Parameter Instances allowed : 13,514,414,924,151
MSG:
ENTER=CALCULATE PF1=HELP PF3=EXIT

@INSTALL

Requests the installation of the specified component. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type @INSTALL(<i>component</i> [, <i>path</i>]) <Enter>
	COMMAND prompt	Type EX @INSTALL(<i>component</i> [, <i>path</i>]) <Enter>
From a rule		Type @INSTALL(<i>component</i> [, <i>path</i>])

Where:

<i>component</i>	The component you want to install.
<i>path</i>	[Optional] The path where the source files are. This path can be absolute, or relative to the directory specified in the DSDIR Execution Environment parameter.

Usage Notes

- The argument *component* is the name of an instance of the @TOINSTALL table. This instance contains the names of objects, table instances, and occurrences to be included in the component.
- The user ID you use to run @INSTALL must have level-7 security clearance. This user ID becomes the owner of all objects listed in the *component* instance of the @TOINSTALL table, with the provision that they can grant access rights to other users.

Exceptions

ROUTINEFAIL	Raised if @INSTALL cannot find the component.
--------------------	---

Example The following example uses @INSTALL to install a new component.

```
@INSTALL(TIMELAPSE ../bin/new)
```

INSTALLIB

Returns the name of the currently designated installation library. (F)

Invocation `result = INSTALLIB`

`result` On return, contains the name of the currently designated installation library. This value is a typeless string of syntax C with a maximum length of 8.

Usage Note The default installation library is SITE.

Example

```

      RULE EDITOR ==>>                                SCROLL: P
WHAT_LIBRARY;
_ LOCAL LIB;
-----
_ LIB = INSTALLIB;                                     | 1
_ CALL MSGLOG('THE CURRENT INSTALLATION LIBRARY IS ' || LIB); | 2
_
_
_
_
_
_
_
-----
```

Pressing PF2 after executing the rule displays the following screen:

```

-----INFORMATION LOG -----
COMMAND ==>>                                SCROLL: P

THE CURRENT INSTALLATION LIBRARY IS SITE
.
```

KEYWORDMGR

Ensures that the TIBCO Object Service Broker keyword system conforms to the established formatting standards and that the keyword index table is up-to-date. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type KEYWORDMGR <Enter>
	COMMAND prompt	Type EX KEYWORDMGR <Enter>

QUERY SYSTEM MANAGER Screen

Executing KEYWORDMGR displays the following screen:

```

QUERY SYSTEM MANAGER          April. 04, 2000

Build Keyword Index and Cross Reference Index on Library: COMMON

CHECK KEYWORDS AGAINST MASTER LIST FOR:  @RULESDOCUMENTS: _
                                           ALL OTHER DOCUMENTS: _

BUILD KEYWORD INDEX: _

BUILD CROSS REFERENCE INDEX: _
  (Building this index may lock your terminal for some time!)
```

Place the cursor on the task of your choice and press enter.
PFKEYS: ENTER=EXECUTE TASK 12=END 3=END

Fields

Build Keyword Index and Cross Reference Index on Library:	The name of the default library appears. This is usually the library COMMON. The entry can be changed to the SITE library.
CHECK KEYWORDS AGAINST MASTER LIST FOR: @RULESDOCUMENTS	Option to check the syntax of keyword lists associated with rules documents for tables, screens, @Reports, and routines. These documents can be either SITE or COMMON. Only ENTRY rules are checked.
ALL OTHER DOCUMENTS	Option to check the syntax of keyword lists not associated with rules documents.
BUILD KEYWORD INDEX	Option to build a new keyword index by replacing the old index if it already exists.
BUILD CROSS REFERENCE INDEX	Option to build a new cross reference index by replacing the old index if it already exists.

Usage Notes

- The keywords are single tokens separated by commas and blanks. This choice puts the keyword list in the @RULESDOCUMENT into this form if they are not already.
- When one of the two **CHECK KEYWORDS AGAINST MASTER LIST** options is selected and one is found that is not in the master list, a screen appears with the names of the rules, its summary, and its list of keywords. The list of keywords can be edited and is replaced when you press PF3 again. The process can be stopped by pressing PF12.
- The master list of keywords is rebuilt when the **BUILD KEYWORD INDEX** option is chosen.
- [KEYWORDSEARCH](#) and [SEARCH](#) depend on the Keyword Index table for the current keyword information.

Required Permissions

The user of this rule must have the authority to insert new occurrences into the KEYWORDINDEX table and to replace the following dictionary tables:

- @REPORTSDOCUMENT
- @RULESDOCUMENT(COMMON)

- @RULESDOCUMENT(SITE)
- @SCREENSDOCUMENT
- @TABLESDOCUMENT
- SYSRULES

KEYWORDSEARCH

Searches the keyword index of a default library to answer a query. (C)

Invocation `CALL KEYWORDSEARCH(querystring, object_type)`

<i>querystring</i>	A query string, which can consist of one or more of the following: names or keywords (the wild card characters asterisk (*) and question mark (?) can be used if they are enclosed in single quotations), the AND (&) and OR () logical operators, the NOT operator (¬), and parentheses symbols "(" and ")".
--------------------	--

<i>object_type</i>	Indicates the type of object sought in keyword searches. The entry must be one of: GLOBALFIELD, LIBRARY, OBJECTSET, REPORT, RULE, SCREEN, TABLE, or ALL.
--------------------	---

- Usage Notes**
- [SEARCH](#) is the interactive version of this tool. It also searches the cross reference index.
 - The local variable *MSG* must be declared by the calling rule.
 - The results of the search are sent to the temporary table @RESULTLIST.

Exceptions

SYNTAX_ERROR	Raised if there is a syntax error in the <i>querystring</i> , if the value for <i>querystring</i> is invalid or if the value for <i>object_type</i> is invalid. A message is placed in <i>MSG</i> for any of these cases.
---------------------	---

Example The following rule searches for the keyword PROCESS_PFKEY and sends the results to the message log:

```
KEYWORDSEARCH_1;
  LOCAL MSG;
```

```
-----+-----
- CALL KEYWORDSEARCH('PROCESS_PFKEY', 'ALL');           | 1
- CALL $RESETPRINT(60, 80, 1 'SCR');                     | 2
- FORALL @RESULTLIST :                                   | 3
-   CALL $PRINTLINE(PAD(@RESULTLIST.INDEX, 4, ' ', 'R')  ||
-     PAD(@RESULTLIST.NAME, 18, ' ', 'R') || ' ' ||
-     @RESULTLIST.TYPE);                                |
-   END;                                                  |
- -----
```

Output for Rule KEYWORDSEARCH_1

Pressing PF2 displays the following screen:

----- INFORMATIONAL MESSAGE LOG -----		
COMMAND ==>		SCROLL: P
-----NEW PAGE-----		
1	GEN_TED	RULE
1	SCRIPT	RULE
1	TED	RULE

LEAPYEAR

Returns a logical value indicating whether a given year is a leap year. (F)

Invocation value = LEAPYEAR(*year*)

value	The value returned is either: Y - Year given is a leap year. N - Year given is not a leap year.
year	A given year.

- Example** The rules in this example:
- 1. Use LEAPYEAR as a condition to a rule
 - 2. Call in a rule to be a function of another rule

Rule Using LEAPYEAR:

LEAPYEAR is called in as a condition to the DAYS_OF_YEAR rule:

DAYS_OF_YEAR(YEAR);	
—	-----
— LEAPYEAR(YEAR);	Y N
—	-----+-----
— RETURN(366);	1
— RETURN(365);	1
—	-----

Definition of the YEAR_HOURS Rule:

DAYS_OF_YEAR is called in as a function to the YEAR_HOURS rule:

YEAR_HOURS(YEAR);	
— LOCAL NUM_HOURS;	
—	-----+-----
— NUM_HOURS = DAYS_OF_YEAR(YEAR) * 24;	1
— RETURN(NUM_HOURS);	2
—	-----

When YEAR_HOURS is executed with the year 1998 used as the argument it returns the following:

```
RESULT OF RULE YEAR_HOURS IS 8760.
```

LENGTH

Returns the length of the specified string. (F)

Invocation `number = LENGTH(string)`

number	On return, contains the length of the string, in characters. Its syntax is B (binary) with length 4.
string	The string whose length is to be returned. Its syntax can be C (fixed-length character string), UN (Unicode), V (variable-length character string), or W (double-byte character).

Usage Notes The current length of *string* is returned, not the definition length.

Example The following rule determines the length of a string and prints it to the message log:

```

      RULE EDITOR   ==>
LENGTH_1;
_  LOCAL NUMBER;
-  -----
-  -----+-----
-  NUMBER = LENGTH('THIS IS THE SOURCE STRING');      | 1
-  CALL MSGLOG('THE SOURCE STRING HAS A LENGTH OF: ' ||  | 2
-  NUMBER);      |
-  -----
```

Output for the LENGTH_1 Rule

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATION LOG -----
COMMAND ==>                                SCROLL ==> P
THE SOURCE STRING HAS A LENGTH OF: 25
```

LIBID

Returns the name of the currently designated local library. (F)

Invocation `result = LIBID`

`result` On return, contains the name of the currently designated local library. This value is a typeless string of syntax C with a maximum length of 8.

Usage Note The name of the default local library is usually the same as the current user’s user ID.

Example

```

      RULE EDITOR ==>>                                SCROLL: P
WHAT_LIBRARY;
_ LOCAL LIB;
-----
_
_
_ LIB = LIBID;                                         | 1
_ CALL MSGLOG('THE CURRENT LOCAL LIBRARY IS ' || LIB); | 2
_
_
_
_
_
_
_
_
_
_
-----
```

Pressing PF2 after executing the rule displays the following screen:

```

----- INFORMATION LOG -----
COMMAND ==>>                                SCROLL: P

THE CURRENT LOCAL LIBRARY IS USER40
.
```

\$LISTDSN

Lists the non-VSAM data sets and Generation Data Group (GDG) data sets of a certain level, using the z/OS Catalog Search Interface services. (C)

Invocation `CALL $LISTDSN(dsname_level, buffer_address)`

<i>dsname_level</i>	The filter defining the levels of data sets to be returned. Contains valid TSO data set name qualifier patterns, separated by periods ('.'). The length is 1 to 44 characters. Refer to Usage Notes below for more information.
<i>buffer_address</i>	The pointer to the first or only buffer that contains the data set names of the <i>dsname_level</i> . Each data set name is 1 to 44 characters long and padded with blanks. The size of each buffer is 4096 bytes and can contain up to 92 data set names. The buffers are provided by \$LISTDSN.

The content of each buffer has the following format for the data set name list:

Offset	Content
0	Pointer to the next buffer in a chain or null (hex zeroes).
4	Number of entries in this buffer.
8	Data set name1.
52	Data set name2.
...	...
4012	Data set name92.

Usage Notes

- Asterisks and percent signs can be used as wildcards in the qualifiers for filtering.
- Only non-VSAM data sets and Generation Data Group (GDG) data sets are retrieved from the z/OS Catalog.

Generic Filter Key

You can use a generic filter key in *dsname_level*. It can contain the following symbols:

Symbol	Meaning
*	A single asterisk by itself indicates that either a qualifier or one or more characters within a qualifier can occupy that position. An asterisk can precede or follow a set of characters.
**	A double asterisk indicates that zero or more qualifiers can occupy that position. A double asterisk cannot precede or follow any characters; it must be preceded or followed by either a period or a blank.
%	A single percent sign by itself indicates that exactly one alphanumeric or national character can occupy that position.
%%...	One to eight percent signs can be specified in each qualifier.

Examples

For non-VSAM data sets:

Specification	Returns	Does not Return
NONVSAM.DATA.SET%	NONVSAM.DATA.SET1 NONVSAM.DATA.SET2	NONVSAM.DATA.SET30
NONVSAM.DATA.SET%%	NONVSAM.DATA.SET30 NONVSAM.DATA.SET31	NONVSAM.DATA.SET1 NONVSAM.DATA.SET2
NONVSAM.*.SET	NONVSAM.DATA1.SET NONVSAM.DATA2.SET	NONVSAM.DATA.SET.BACKUP
NONVSAM.*A	NONVSAM.A NONVSAM.BA NONVSAM.BBA	NONVSAM.B NONVSAM.AB
NONVSAM.DATA.*	NONVSAM.DATA.SET1 NONVSAM.DATA.SET2	NONVSAM.DATA.SET.BACKUP
NONVSAM.DATA*	NONVSAM.DATA1 NONVSAM.DATA23	NONVSAM.DATA.SET

Specification	Returns	Does not Return
NONVSAM.**	NONVSAM.DATA.SET1 NONVSAM.DATA.SET2 NONVSAM.DATA.SET.BACKUP	NONVSAM1.DATA.SET
NONVSAM.DATA.SET	NONVSAM.DATA.SET only.	
**.DATA	Entry names whose low-level qualifier is DATA, such as: <ul style="list-style-type: none"> NONVSAM.DATA DATASET.WORK.DATA 	
**	Every entry name in a catalog.	

For a GDG base named DATASET.GDG containing the following entries:

- DATASET.GDG.G0001V00
- DATASET.GDG.G0002V00
- DATASET.GDG.G0003V00

and for the non-VSAM data set named DATASET.GDG.G0001V00.XYZ, the following keys return the following results:

Specification	Returns
DATASET.GDG.**	DATASET.GDG.G0001V00 DATASET.GDG.G0002V00 DATASET.GDG.G0003V00 DATASET.GDG.G0001V00.XYZ
DATASET.GDG.G0001V00	DATASET.GDG.G0001V00
DATASET.GDG.G0001V00.**	DATASET.GDG.G0001V00.XYZ
DATASET.GDG.G000%V00	DATASET.GDG.G0001V00 DATASET.GDG.G0002V00 DATASET.GDG.G0003V00

Specification	Returns
DATASET.GDG.G000%V00.**	DATASET.GDG.G0001V00
	DATASET.GDG.G0002V00
	DATASET.GDG.G0003V00
	DATASET.GDG.G0001V00.XYZ



The entries returned are not necessarily in ascending order.

Exceptions

ROUTINEFAIL	\$LISTDSN can fail for the following reasons:
	<ul style="list-style-type: none">• A Catalog Management error• An invalid filter key• Internal \$LISTDSN error: IGGCSI00 parmlist error• Internal \$LISTDSN error: IGGCSI00 parmlist pointer is zero• The buffer provided to the Catalog Search Interface is too small
	You can retrieve error messages using RETURN_SYSMMSG .

Examples These sets of rules lists the data set names at a certain level on the message log:

```

      RULE EDITOR ==>
LIST_DSN(LVL);
_ LOCAL DSNBUFF, BNEXT, DCOUNT;
_ -----
_ -----
_ CALL MSGLOG('DSN_LEVEL entered: ' || LVL);
_ CALL $LISTDSN(LVL, DSNBUFF);
_ UNTIL DONE :
_   @MAP.ADDRESS = DSNBUFF;
_   @MAP.SIZE = 4096;
_   INSERT @MAP('EXTERNALRO');
_   GET HZS81_MAP_WORD(DSNBUFF);
_   BNEXT = HZS81_MAP_WORD.WORD;
_   GET HZS81_MAP_WORD(DSNBUFF + 4);
_   DCOUNT = HZS81_MAP_WORD.WORD;
_   CALL LIST_DSN1(DSNBUFF, BNEXT, DCOUNT);
_   DSNBUFF = BNEXT;
_   END;
_ CALL ENDMSG('Press PF2 to see the data set list');
```

SCROLL: P

1
2
3

4


```
-----
- ON ROUTINEFAIL :
-   CALL ENDMSG(RETURN_SYMSG);
-
-----
-
- RULE EDITOR ==>                                SCROLL: P
LIST_DSN1(LIST, BNEXT, DCOUNT);
-
- -----
- DCOUNT = 0;                                     | Y N N
- BNEXT = 0;                                       |   Y N
- -----
- CALL MSGLOG('No data set names found');          | 1
- FORALL HZS81_MAP_DSN(LIST + 8) WHERE KEY <= DCOUNT : | 1 1
-   CALL MSGLOG(HZS81_MAP_DSN.DSNAME);            |
-   END;                                           |
- SIGNAL DONE;                                    | 2 2
- -----
-
-----
```

Executing the rules produces the following message log:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
DSN_LEVEL entered: USR40.**
USR40.CLIST
USR40.CNTL
USR40.OBJSTAR.PARMS
USR40.ISPF.ISPPROF
-----
```

See Also The [@MAP](#) shareable tool.

\$LISTPDS

Lists the member names of a partitioned data set (PDS), or retrieves the statistics for a PDS member. (C)

Invocation `CALL $LISTPDS(pds_name, buffer_address, member_name)`

<i>pds_name</i>	The name of a PDS. The length is 1 to 44 characters.																
<i>buffer_address</i>	<p>The pointer to the first or only buffer that contains the member names of the PDS. Each member name is 1 to 8 characters long and padded with blanks. The size of each buffer is 4096 bytes and can contain up to 511 member names. The buffers are provided by \$LISTPDS.</p> <p>The content of each buffer has the following format:</p> <table> <tr> <th>Offset</th><th>Content</th></tr> <tr> <td>0</td><td>Pointer to the next buffer in a chain, or null (hex zeroes).</td></tr> <tr> <td>4</td><td>Number of entries in this buffer.</td></tr> <tr> <td>8</td><td>Member name1.</td></tr> <tr> <td>16</td><td>Member name2.</td></tr> <tr> <td>...</td><td>...</td></tr> <tr> <td>4080</td><td>Member name510.</td></tr> <tr> <td>4088</td><td>Member name511.</td></tr> </table>	Offset	Content	0	Pointer to the next buffer in a chain, or null (hex zeroes).	4	Number of entries in this buffer.	8	Member name1.	16	Member name2.	4080	Member name510.	4088	Member name511.
Offset	Content																
0	Pointer to the next buffer in a chain, or null (hex zeroes).																
4	Number of entries in this buffer.																
8	Member name1.																
16	Member name2.																
...	...																
4080	Member name510.																
4088	Member name511.																
<i>member_name</i>	<p>To get a list of the PDS members: 8 blank characters.</p> <p>To get the statistics for one PDS member: the 1- to 8-character member name padded with blanks.</p>																

Exceptions

ROUTINEFAIL \$LISTPDS can fail for the following reasons:

- Allocation of the PDS file failed
- Cannot get virtual storage for the buffer
- File not found
- Open for the PDS file failed
- PDS member not found
- The file specified is not a partitioned data set
- Too many open files

You can retrieve error messages using [RETURN_SYMSG](#).

Examples Listing Member Names

These sets of rules lists the names of the members of a PDS on the message log.

```

      RULE EDITOR ==>                                SCROLL: P
LIST_PDS(PDS);
_ LOCAL PDSBUFF, BNEXT, MCOUNT, MEMBER;
_ -----
_ -----+-----
_ CALL MSGLOG('PDS_NAME entered: ' || PDS);           1
_ MEMBER = ' ';                                       2
_ CALL $LISTPDS(PDS, PDSBUFF, MEMBER);                 3
_ UNTIL DONE :                                       4
_   @MAP.ADDRESS = PDSBUFF;
_   @MAP.SIZE = 4096;
_   INSERT @MAP('EXTERNALRO');
_   GET HZS81_MAP_WORD(PDSBUFF);
_   BNEXT = HZS81_MAP_WORD.WORD;
_   GET HZS81_MAP_WORD(PDSBUFF + 4);
_   MCOUNT = HZS81_MAP_WORD.WORD;
_   CALL LIST_PDS1(PDSBUFF, BNEXT, MCOUNT);
_   PDSBUFF = BNEXT;
_   END;
_ CALL ENDMSG('Press PF2 to see the member list');     5
_ -----
_ ON ROUTINEFAIL :
_   CALL ENDMSG(RETURN_SYMSG);

```

```

      RULE EDITOR ==>                                SCROLL: P
LIST_PDS1(LIST, BNEXT, MCOUNT);
_

```

```
-----
- MCOUNT = 0;                                | Y N N
- BNEXT = 0;                                  |   Y N
- -----
- CALL MSGLOG('No members in data set');        | 1
- FORALL HZS81_MAP_PDS(LIST + 8) WHERE KEY <= MCOUNT : | 1 1
-   CALL MSGLOG(HZS81_MAP_PDS.MEMBER);
-   END;
- SIGNAL DONE;                                | 2 2
- -----
```

Executing the rules produces the following message log:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
PDS_NAME entered: USR40.CNTL
EMAIL
FTP
```

Listing Information for a Member

The following set of rules receives the name of a member of a PDS and lists the statistics for that member.

```
-----
- RULE EDITOR ==>                                SCROLL: P
- LIST_MEMSTATSS(PDS, MEMBER);
-   LOCAL PDSBUFF, LEN;
- -----
- CALL MSGLOG('PDS_NAME entered: ' || PDS || '(' || MEMBER || | 1
-   ')');
- CALL $LISTPDS(PDS, PDSBUFF, MEMBER);           | 2
- @MAP.ADDRESS = PDSBUFF;                         | 3
- @MAP.SIZE = 130;                                | 4
- INSERT @MAP('EXTERNALRO');                     | 5
- GET HZS81_MAP_HALF(PDSBUFF);                   | 6
- LEN = HZS81_MAP_HALF.HALF;                     | 7
- CALL LIST_MEMSTATSS1(PDS, MEMBER, PDSBUFF, LEN); | 8
- -----
- ON ROUTINEFAIL :
-   CALL ENDMSG(RETURN_SYMSG);
```

```
-----
- RULE EDITOR ==>                                SCROLL: P
- LIST_MEMSTATSS1(PDS, MEMBER, PDSBUFF, LEN);
-   LOCAL VV, MM, SS, CREATED, CHANGED, HH, MN, SIZE, INITIAL, MODIFIED, USERID
-   ;
- -----
- LEN = 0;                                | Y N N
- LEN = 30;                              |   Y N
```

— -----+-----	
— GET HZS81_MAP_STATS(PDSBUFF);	1
— HZS81_MEMSTATS_H.* = HZS81_MAP_STATS.*;	2
— HZS81_MEMSTATS_H.PDS_NAME = PDS;	3
— HZS81_MEMSTATS_H.MEMBER = MEMBER;	4
— INSERT HZS81_MEMSTATS_H('HZS81_MEMSTATS');	5
— DISPLAY HZS81_MEMSTATS;	6
— CALL ENDMSG('No ISPF statistics exist for member: '	1
— MEMBER);	
— CALL ENDMSG(1
— 'User data is not ISPF statistics for member: '	
— MEMBER);	
— -----	

Executing the rules produces the following information:

PDS NAME entered: USR40.CNTL										
NAME	VV	MM	CREATED	CHANGED	HH:MM:SS	SIZE	INITIAL	MODIFIED	USERID	Len
EMAIL	1	2	103317	103317	21 22 12	18	22	1	USR40	

See Also The [@MAP](#) shareable tool.

LIT_TO_VAL

Converts a string to a typeless value as described in the string. (F)

Invocation `new_value = LIT_TO_VAL(string)`

<code>new_value</code>	On return, this value is the value described in <i>string</i> . Its semantic data type is typeless.
------------------------	---

<code>string</code>	The string to convert. Its syntax can be C (fixed-length character string) or V (variable-length character string).
---------------------	---

The syntax of *new_value* depends on the format of *string* as follows:

string Format	new_value Syntax
R'xx...'	RD (raw data)
U'xx...'	UN (Unicode)
X'xx...'	V (variable-length character string)

Example The following rule creates three new values from the information contained in a string:

```
CREATE_RD;
_ LOCAL SOURCE_STRING, NEW_RD_VALUE, NEW_UN_VALUE, NEW_X_VALUE;
_ -----
_ SOURCE_STRING = 'R''1234''';
_ NEW_RD_VALUE = LIT_TO_VAL(SOURCE_STRING);
_ SOURCE_STRING = 'U''/20AC//1234''''''';
_ NEW_UN_VALUE = LIT_TO_VAL(SOURCE_STRING);
_ SOURCE_STRING = 'X''5BF1F2F3F4''';
_ NEW_X_VALUE = LIT_TO_VAL(SOURCE_STRING);
_ -----
```

NEW_RD_VALUE is now a raw data value that contains “1234”.

NEW_UN_VALUE is a Unicode value containing “?/1234' ” (? = the euro symbol).

NEW_X_VALUE is a variable-length character value that contains “\$1234”.

LLOAD

Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded from files with names in mixed case or lowercase. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type LLOAD <Enter>
	COMMAND prompt	Type EX LLOAD <Enter>

Pressing Enter displays the screen shown here:

LLOAD Utility
File name:
Log msgs to: SCR
PFKEYS: ENTER=LOAD 3=EXIT 12=EXIT

In the **File Name** field, type the name of the file to be unloaded. Valid values for the **Log msgs to** field are SCR or PRT; SCR is the default.

- Prerequisites**
- Items being loaded must have been unloaded using [UNLOAD](#), [UNLOAD_DATA](#), [UNLOAD_DEFN](#), or [UNLOADLIBRARY](#). Refer to the appropriate entries in this manual for more information about these tools.
 - If the data is being loaded into a table that uses required, reference, or default fields, you must have MODIFY_DEFN rights to that table to load the data.

Usage Notes

If...	Then...
You execute LLOAD	Pressing Enter displays a screen prompting for values.
You are loading a file from z/OS	The value for <i>importfile</i> must be a fully qualified data set name.

If...	Then...
If you are loading a file from Windows or Solaris	Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to the <i>TIBCO Object Service Broker Parameters</i> manual for more information about this parameter.
A rule already exists	It is replaced with the rule being loaded. Rules being loaded are saved in the current local library.
You want to load large volumes of data quickly	Consider using the batch load utilities.
You are loading a table, screen, or report definition	No definition (and associated occurrences) is loaded if the item already exists.
Only table occurrences are being loaded	The table must already exist. At minimum, it must contain the same fields (extra fields take on the default value). Changes in type, syntax and field order are allowed. If conversion errors result, that occurrence only is skipped.

- LLOAD functions in exactly the same way as [LOAD](#), except that with LLOAD you can load files with names that are in mixed case or in lowercase in operating environments (such as Solaris) that are case sensitive in their handling of filenames.
- Certain options are disabled when loading TDS data. These include reference checking, defaults, event rules, and the IDgen flag.
- When loading the definition and data of a table that has a secondary index, the secondary index is lost and the indicator is removed from the definition. You must run [SIXBUILD](#) to recreate the index.
- On non-z/OS systems, set the DSBIFFYPE Execution Environment parameter to LENGTH_PREFIXED_EBCDIC.



This setting can affect the behavior of other tools.

- To LLOAD a DB2 table definition that was unloaded from Release 5.0.0 into the current release, set library to S6B50DB2 and execute the LLOAD tool. That task loads the definition and converts it to the format of the current release.

See Also *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities* for information about the batch load utilities.
TIBCO Object Service Broker Parameters for information on the DSBIFTYPE Execution Environment parameter.

Example On z/OS, specifying the following value loads the items in the data set USR40.EXAMPLE.DATA(RULES). By default, any messages are sent to the screen:

```
File Name:USR40.EXAMPLE.DATA(RULES)
```

```
Log msgs to: SCR
```

On Open Systems, specifying the following value loads the items in the file rules; your DSDIR session option determines the directory. Any messages are sent to the printer:

```
File Name: rules
```

```
Log msgs to: PRT
```

LOAD

Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrators workbench	LO Load from a File option	Press Enter
Developer's workbench	EX Execute Rule option	Type LOAD (<i>importfile</i> , <i>media</i>) <Enter>
	COMMAND prompt	Type EX LOAD (<i>importfile</i> , <i>media</i>) <Enter>
From a rule		Type CALL LOAD(<i>importfile</i> , <i>media</i>).

Where:

<i>importfile</i>	The name of the file containing the previously unloaded information.
<i>media</i>	One of: SCR – Send messages to the message log. PRT – Send messages to the printer.

- Prerequisites**
- Items being loaded must have been unloaded using [UNLOAD](#), [UNLOAD_DATA](#), [UNLOAD_DEFN](#) or [UNLOADLIBRARY](#). Refer to the appropriate entries in this manual for more information about these tools.
 - To load the data into a table that uses required, reference, or default fields, you must have MODIFY_DEFN rights to that table.

Usage Notes **Search Paths**

- If LOAD is executed using the EX option from the workbench, the search path used for event rules is local, SITE, and then COMMON.

- If LOAD is executed using the LO or UL options from the @ADMIN menu, the search path is COMMON since the search path is indicated in MENU_ITEMS(@ADMIN) as S.
- To LOAD a DB2 table definition that was unloaded from Release 5.0.0 into the current release, set library to S6B50DB2 and execute the LOAD tool. That task loads the definition and converts it to the format of the current release.

Loading Different Types of Objects

If...	Then...
You are loading data from an operating system that is case-sensitive in its handling of filenames (such as Solaris) and you are loading a file with a name that is not entirely in upper case	Use the LLOAD shareable tool to load the file. In case-sensitive operating systems, if the file is in mixed or lower case, LOAD fails. Alternately, rename the file so it is all uppercase.
You want to load large volumes of data quickly	Consider using the BATCHLOAD utilities.
You are loading a z/OS data set	The value for <i>importfile</i> must be a fully qualified data set name.
If you are loading a Windows or Solaris file	Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to the <i>TIBCO Object Service Broker Parameters</i> manual for more information about this parameter.
Only table occurrences are being loaded	The table must already exist. At minimum, it must contain the same fields (extra fields take on the default value). Changes in type, syntax and field order are allowed. If conversion errors result, that occurrence only is skipped.

LOAD and Existing Objects

- If a rule already exists, it is replaced with the rule being loaded. Rules being loaded are saved in the current local library.

- If a table, screen, or report definition already exists, no definition or associated occurrences are loaded.

Other Notes

- If you execute LOAD without supplying values for *importfile* and *media*, pressing Enter displays a screen prompting for values.
- When loading TDS data, if the definition of the table into which the data is being loaded is not bound, certain options are disabled. These options include reference checking, defaults, event rules, and the IDgen flag. These options are not disabled if the table definition is bound.
- Definitions of all table types can be loaded using the LOAD tool. However, data for only TDS and session (SES) tables can be loaded using the LOAD tool.
- When loading the definition and data of a table that has a secondary index, the secondary index is lost and the indicator is removed from the definition. You must run [SIXBUILD](#) to recreate the index.
- On non-z/OS systems, set the DSBIFFTYPE Execution Environment parameter to LENGTH_PREFIXED_EBCDIC.



This setting can affect the behavior of other tools.

- To load on z/OS a file unloaded on Windows or Solaris, simply FTP the file, in binary format, from the platform of origin to z/OS.
- To load on Windows or Solaris a file unloaded on z/OS, you must either issue FTP's QUOTE SITE RDW command before sending the file from z/OS, or run the S6BBRFRU (Reformat TIBCO Object Service Broker Files Transferred with FTP) utility against the file on z/OS before using FTP.

See Also *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities* for information on batch load utilities.

TIBCO Object Service Broker Parameters for information on the DSBIFFTYPE Execution Environment parameter.

Example The following statement loads the items in the data set USR40.EXAMPLE.DATA(RULES); any messages are sent to the screen:

```
EX LOAD( 'USR40.EXAMPLE.DATA(RULES)' , 'SCR' )
```

The following statement loads the items in the file /usr/usr40/RULES; any messages are sent to the printer:

```
EX LOAD( 'RULES' , 'PRT' )
```

LOADER

Loads definitions and data of TIBCO Object Service Broker objects that were previously unloaded, with selection control. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type LOADER <Enter>
	COMMAND prompt	Type EX LOADER <Enter>
From a rule, to use one action for all files		Type CALL LOAD_BY_ACTION(<i>file</i> [, <i>defaultlib</i>], <i>defn_action</i> , <i>data_action</i> , <i>media</i>)
From a rule, to use different actions for different files		Type CALL FILL_LOADPROMPT(<i>file</i> [, <i>defaultlib</i>]), and then type CALL LOADER_TABLE(<i>file</i> [, <i>defaultlib</i>], <i>media</i>)

Where:

<i>file</i>	The name of the file containing the previously unloaded information.
<i>defaultlib</i>	[Optional] The name of the library where you want the information loaded. If entered, this argument must be the name of an existing library.
<i>defn_action</i>	<p>The action to be applied to all definitions in the file. Must be one of:</p> <p>IS – Insert or Skip. When an object in the file has the same name as one already in the database, this action skips the object in the file, and the database is to remain unchanged.</p> <p>IR – Insert or Replace. When an object in the file collides with one already in the database, the object in the file replaces the one in the database.</p>

<i>data_action</i>	<p>The action to be applied to all data in the file. Must be one of:</p> <p>IS – Insert or Skip. If there is already data in the table, or in one instance of a parameterized table, for example, this actions skips the data in the file.</p> <p>IR – Insert or Replace. If there is already data in the table, or in one instance of a parameterized table, for example, this action clears the table and loads the data in the file.</p> <p>MI – Merge with Insert only. When there is data in the file and data in the table, this action tests every occurrence from the file against the table. If the occurrences collide, in the sense of having identical primary keys, the occurrence from the file is skipped; otherwise, it is inserted.</p> <p>MR – Merge with Replace. This action tests individual occurrences from the file against the table. If the occurrences have identical primary keys, the one from the file replaces the one in the table; otherwise the one from the file is inserted.</p>
<i>media</i>	<p>One of:</p> <p>SCR – Send messages to the message log.</p> <p>PRT – Send messages to the printer.</p>

- Prerequisites**
 - Items being loaded must have been unloaded using [UNLOAD](#), [UNLOAD_DATA](#), [UNLOAD_DEFN](#) or [UNLOADLIBRARY](#). Refer to the appropriate entries in this manual for more information about these tools.
 - To load the data into a table that uses required, reference, or default fields, you must have MODIFY_DEFN rights to that table.
- Usage Notes**
 - Running LOADER from the workbench displays the following screen:

File Loader Utility

File: _____

Send Report to: SCR SCR/PRT

Destination Library: _____ (Optional)

To LOAD on PF3 you must specify the two Actions

Action for definitions: __
Choose IS - Insert or Skip, IR - Insert or Replace

Action for data: __
Choose IS - Insert or Skip

IR - Insert or Replace
 MI - Merge with Insert only
 MR - Merge with Insert or Replace

PFKEYS: 5=SELECT 3=LOAD 2=LOGS 12=CANCEL

From this screen, you can load information by filling in the different fields according to the specifications under [Invocation on page 325](#).

- To use PF3, you must specify the actions to be used with all files in the Action for definitions field and in the Action for data field.
- When you press PF5, you can specify, on the screen that follows, the actions to be taken for each individual objects. If the action is left blank for a definition, that item is not loaded.

Search Paths

If LOADER is executed using the EX option from the workbench, the search path used for event rules is the local, the installation (the default is SITE), and the system (the default is COMMON) libraries.

Other Notes

- When loading TDS data, if the definition of the table into which the data is being loaded is not bound, certain options are disabled. These options include reference checking, defaults, event rules, and the IDgen flag. These options are not disabled if the table definition is bound.
- Definitions of all table types can be loaded using the LOADER tool. However, only TDS and session (SES) table data can be loaded using the LOADER tool.
- When loading the definition and data of a table that has a secondary index, the secondary index is lost and the indicator is removed from the definition. You must run [SIXBUILD](#) to recreate the index.
- On non-z/OS systems, set the DSBIFFTYPE Execution Environment parameter to LENGTH_PREFIXED_EBCDIC.



This setting can affect the behavior of other tools.

- Do not load DB2 table definitions unloaded from Release 5.0.0 into a system at the current release with LOADER. Use the LOAD shareable tool instead.

- Problems may be experienced if Borrower Rights are held on an existing object being replaced, even if the Borrower is the user invoking LOADER:
Cannot delete TDS_TABLE “%”, unable to verify promotion rights
- Replace of a definition will fail if the user running LOADER is not either the Owner of the existing object or a level-7 user:
Security Failure : Denied "OWNER" access to table “%”
- Using LOADER to replace (IR) an existing table definition with a modified one containing, for example, a new field - and then attempting to load data to that modified table at the same time will fail as the 'old' definition is still being reference at the time the data is loaded.
Unable to INSERT because of INTEGRITYFAIL: “%” is not a field of table “%”
This can be worked around by using the PF5 SELECT capability, selecting just the definition first, and then repeating the process for the data.

See Also *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities* for information on batch load utilities
TIBCO Object Service Broker Parameters for information on the DSBIFTYPE Execution Environment parameter

Example The following screen shows LOADER loading the items in the data set USR40.EXAMPLE.DATA(SCREENS) into the SCREENS library, with messages sent to the printer:

File Loader Utility

File: USR40.EXAMPLE.DATA(SCREENS)_____

Send Report to: PRT SCR/PRT

Destination Library: SCREENS_ (Optional)

To LOAD on PF3 you must specify the two Actions

Action for definitions: ____
Choose IS - Insert or Skip, IR - Insert or Replace

Action for data: ____
Choose IS - Insert or Skip
IR - Insert or Replace
MI - Merge with Insert only
MR - Merge with Insert or Replace

PFKEYS: 5=SELECT 3=LOAD 2=LOGS 12=CANCEL

With no actions specified, after pressing PF5, the following screen appears, where you specify an action for each definition in the file:

Select Objects to Load			Scroll: P
File: USR40.EXAMPLE.DATA(SCREENS)			
Command:			
Name	Object	Exists	Parameters/Library
-----	-----	-	-----
___ SCR31	SCREEN	N	
___ SCRTAB31	SCR_TABLE	N	

Defn: IS Insert/skip IR Insert/replace
Data: IS Insert/skip IR Insert/replace MI Merge&insert MR Merge&replace
PFKEYS: 3=LOAD 12=CANCEL 5=REFIND 9=RECALL

LOCALTIME

Returns the local time when the transaction started. (F)

Invocation now = LOCALTIME

now	On return, contains the current local time. Its syntax is C (fixed-length character string) with length 8.
-----	--

Usage notes LOCALTIME is typically used to notify the user of the local time a transaction is started.

Example The set of rules in this example:

- 1. Schedules a transaction and uses LOCALTIME to notify the user of the local start time of a transaction
- 2. Starts a transaction
- 3. Updates an audit table

The user sees the transaction start time in the local time and the system time is entered to the audit table.

Rule Using LOCALTIME:

USER_TIME schedules the TRANSACT rule and notifies the user with an ENDMSG:

```
USER_TIME;
_
_
_ SCHEDULE TRANSACT;
_ CALL ENDMSG(( 'TRANSACT SCHEDULED ' ) || LOCALTIME);
_
```

The following ENDMSG is returned:

11:54:12 TRANSACT SCHEDULED 11:54:11

Definition of the TRANSACT Rule

TRANSACT calls in another rule and inserts the start and end time into the table AUDIT, using system time:

TRANSACT;	
-----	+-----
_ CALL EMPLOYEE_DEPTNO;	1
_ AUDIT.START = TIME;	2
_ AUDIT.END = REALTIME;	3
_ INSERT AUDIT;	4

The following values are added to the sample table AUDIT:

TRANS_NUMBER	START	END
-----	-----	-----
1	08:54:11	08:54:17

LOG_BROWSE

Displays the contents of the message log. (C)

Invocation `CALL LOG_BROWSE`

The message log appears as if you pressed PF2 from the workbench.

Usage Notes

- The transaction must end before the contents of its message log can appear.
- The message log is cleared at the beginning of a new transaction.

See Also *TIBCO Object Service Broker Programming in Rules* for information about the message log.

Example The following set of rules lists the staff for a manager in the message log.

Rule Using LOG_BROWSE

The HIERARCHY rule executes STAFF and then saves the information in the message log by calling LOG_BROWSE.

```

HIERARCHY(NUMBER);
_
_ EXECUTE STAFF(NUMBER);                                | 1
_ CALL LOG_BROWSE;                                       | 2
_ -----

```

Rule Using MSGLOG

The STAFF example rule uses the message log to list all the people that work for the manager.

```

STAFF(NUM);
_
_ CALL MSGLOG('THE FOLLOWING EMPLOYEES WORK FOR MANAGER ' || | 1
_   NUM || ':' );
_ GET EMPLOYEE WHERE MGR# = NUM;                               | 2
_ FORALL EMPLOYEE WHERE MGR# = NUM:                             | 3
_   CALL MSGLOG(EMPLOYEE.LNAME);
_   END;
_ -----
_ ON GETFAIL:
_   CALL MSGLOG(NUM || ' IS NOT A MANAGER. ');

```

Results of the STAFF Rule

If the manager number is valid, results such as the following appear in the message log:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ==>                                SCROLL ==> P  
THE FOLLOWING EMPLOYEES WORK FOR MANAGER 79912:  
SMITH  
HRODEK  
CANNON  
KIMURA  
WONG  
SCHULTZ  
BOIVIN
```

LOWER_EBCDIC

Converts a string to lowercase EBCDIC characters. (F)

Invocation `lower_string = LOWER_EBCDIC(string)`

<code>lower_string</code>	<p>On return, contains the string in lowercase letters.</p> <p>If <i>string</i> is not Unicode, or if it is Unicode and entirely convertible to EBCDIC, <i>lower_string</i>'s syntax is V (variable-length character string).</p> <p>If <i>string</i> is Unicode and not entirely convertible to EBCDIC, <i>lower_string</i>'s syntax is UN (Unicode). In <i>string</i>, characters that can be converted are cased and then reverted to Unicode before being added to <i>lower_string</i>. Characters that cannot be converted are added to <i>lower_string</i> unchanged.</p>
<code>string</code>	<p>The string to convert to lowercase letters. Its syntax is C (fixed-length character), RD (raw data), UN (Unicode), V, or W (double-byte character).</p>

Usage Note LOWER_EBCDIC lowercases strings using the TIBCO Object Service Broker EBCDIC casing rules.

Example This rule lowercases a string and prints the result to the message log:

```

LOWERCASE_SAMPLE;
_ LOCAL A;
_ -----
_ -----+-----
_ A = U'AÇaç';                                | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A);      | 2
_ CALL MSGLOG(' ');                                | 3
_ CALL MSGLOG('LOWER_EBCDIC GIVES ' || LOWER_EBCDIC(A)); | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWER_EBCDIC(A)); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A);                | 6
_ CALL MSGLOG(' ');                                | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A);      | 8
_ CALL MSGLOG(' ');                                | 9
_ CALL MSGLOG('LOWER_EBCDIC GIVES ' || LOWER_EBCDIC(A)); | A
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWER_EBCDIC(A)); | B

```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Displayed Output for the Rule

Pressing PF2 after executing this rule displays the following:

```
COMMAND ===>                                SCROLL ===> P
CASING OF UNICODE STRING AÇaç
LOWER_EBCDIC GIVES aÇaç
SYNTAX IS V

CASING OF EBCDIC STRING AÇaç
LOWER_EBCDIC GIVES aÇaç
SYNTAX IS V
```

See Also Related tools: [LOWER_UNICODE](#), [LOWERCASE](#), [UPPER_EBCDIC](#), [UPPER_UNICODE](#), and [UPPERCASE](#).

LOWER_UNICODE

Converts a string to lowercase Unicode characters. (F)

Invocation `lower_string = LOWER_UNICODE(string)`

<code>lower_string</code>	On return, contains the string in lowercase letters. Its syntax is UN (Unicode).
<code>string</code>	The string to convert to lowercase letters. Its syntax is C (fixed-length character string), UN, or V (variable-length character string).

Usage Note LOWER_UNICODE lowercases strings using the TIBCO Object Service Broker Unicode casing rules.

Exceptions

ROUTINEFAIL Signaled if string is syntax W (double-byte character).

Example This rule lowercases a string and prints the result to the message log:

```

LOWERCASE_SAMPLE;
_ LOCAL A;
_ -----
_                                     |
_ A = U'AÇaç';                       | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A); | 2
_ CALL MSGLOG(' ');                  | 3
_ CALL MSGLOG('LOWER_UNICODE GIVES ' || LOWER_UNICODE(A)); | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWER_UNICODE(A)); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A); | 6
_ CALL MSGLOG(' ');                  | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A); | 8
_ CALL MSGLOG(' ');                  | 9
_ CALL MSGLOG('LOWER_UNICODE GIVES ' || LOWER_UNICODE(A)); | A
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWER_UNICODE(A)); | B

```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Displayed Output for the Rule

Pressing PF2 after executing this rule displays the following:

COMMAND ===>	SCROLL ===> P
CASING OF UNICODE STRING AÇaç	
LOWER_UNICODE GIVES açaç	
SYNTAX IS UN	
CASING OF EBCDIC STRING AÇaç	
LOWER_UNICODE GIVES açaç	
SYNTAX IS UN	

See Also Related tools: [LOWER_EBCDIC](#), [LOWERCASE](#), [UPPER_EBCDIC](#), [UPPER_UNICODE](#), and [UPPERCASE](#).

LOWERCASE

Converts all uppercase characters in a string to lowercase characters. (F)

Invocation `lower_string = LOWERCASE(string)`

<code>lower_string</code>	On return, contains the string in lowercase letters. Its syntax is V (variable-length character string) except that, if <i>string</i> is UN (Unicode), <i>lower_string</i> becomes UN.
<code>string</code>	The string to convert to lowercase letters. Its syntax can be C (fixed-length character string), UN, V, or W (double-byte character).

Usage Note LOWERCASE lowercases EBCDIC strings using the TIBCO Object Service Broker EBCDIC casing rules and Unicode strings using the TIBCO Object Service Broker Unicode casing rules.

Example This rule lowercases a string and prints the result to the message log:

```

LOWERCASE_SAMPLE;
_ LOCAL A;
_ -----
_ -----+-----
_ A = U'AÇaç';                               | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A); | 2
_ CALL MSGLOG(' ');                           | 3
_ CALL MSGLOG('LOWERCASE GIVES ' || LOWERCASE(A)); | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWERCASE(A))); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A);           | 6
_ CALL MSGLOG(' ');                           | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A); | 8
_ CALL MSGLOG(' ');                           | 9
_ CALL MSGLOG('LOWERCASE GIVES ' || LOWERCASE(A)); | A
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(LOWERCASE(A))); | B

```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Displayed Output for the LOWERCASE_1 Rule

Pressing PF2 after executing this rule displays the following:

```

COMMAND ==>
CASING OF UNICODE STRING AÇaç

```

```

SCROLL ==> P

```

LOWERCASE GIVES açaç
SYNTAX IS UN

CASING OF EBCDIC STRING AÇaç

LOWERCASE GIVES aÇaç
SYNTAX IS V

See Also Related tools: [LOWER_EBCDIC](#), [LOWER_UNICODE](#), [UPPER_EBCDIC](#), [UPPER_UNICODE](#), and [UPPERCASE](#).

@MAKEMEMBERS

Creates the member list for an object set to be enabled through the BATCH_ENABLE utility. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type @MAKEMEMBERS <Enter>
	COMMAND prompt	Type EX @MAKEMEMBERS <Enter>
From a rule		Type CALL @MAKEMEMBERS(<i>objectset</i>)

Where:

<i>objectset</i>	An object set whose member list is to be prepared. The (security) permissions of the object set be specified and it is enabled for its members by BATCH_ENABLE utility.
------------------	---

Prerequisites To be able to use the @MAKEMEMBERS tool, an associated object set also named @MAKEMEMBERS must be enabled by a system administrator for those users who are allowed to use the tool.

Usage Notes Running the @MAKEMEMBERS rule displays a screen from which you can add users and groups in any of the following ways:

- Type in the name of the user IDs or groups that you want to have access to your object set.
- Press PF6 to select from a list of all available user IDs.
- Press PF9 to select a group name from the list of all security groups.

You must have at least one member listed to save any object set that is enabled later via the [BATCH_ENABLE](#) tool.



Make sure that your promotions administrator suspends all users from the system before running BATCH_ENABLE (not @MAKEMEMBERS).

MANAGE_APPLY

Invokes the Promotion facility on the target system. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	PT Target Promotion Admin option	Press Enter
Developer's workbench	EX Execute Rule option	Type MANAGE_APPLY <Enter>
	COMMAND prompt	Type EX MANAGE_APPLY <Enter>

- Usage notes**
- When you execute MANAGE_APPLY, the target promotions menu appears.
 - MANAGE_APPLY must be executed with the option BROWSE = Y.
 - From the target system, you can apply change requests that are extracted from your source system, or back out change requests that were applied previously. You must have the appropriate security permissions to perform these functions.

See Also *TIBCO Object Service Broker Managing Deployment* for complete information about administering the Promotion system.

MANAGE_REQUESTS

Invokes the Promotion facility on the source system. (E)

Invocation Do one of the following:

From...	Move the cursor to the...	And...
Administrator workbench	PS Source Promotion Admin option	Press Enter.
Any workbench	EX Execute Rule option	Type MANAGE_REQUESTS<Enter>
	COMMAND prompt	Type EX MANAGE_REQUESTS<Enter>

- Usage Notes**
- When you execute MANAGE_REQUESTS, the first panel for MANAGE_REQUESTS appears.
 - MANAGE_REQUESTS must be executed with the option BROWSE = Y.
 - From the source system, you can accept or reject change requests, promote them within your source system, extract them to apply them to another system, or back them out.

See Also *TIBCO Object Service Broker Managing Deployment* for complete information about administering the Promotion system.

MANAGE_RIGHTS

Releases or transfers a user's promotion rights on rules, screens, reports, menus, object sets, and tables. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	MR manage rights option	Press Enter.
	EX execute rule option	Type MANAGE_RIGHTS <Enter>
	COMMAND prompt	Type EX MANAGE_RIGHTS <Enter>

Usage Notes When you execute MANAGE_RIGHTS, the first panel for MANAGE_RIGHTS appears.

See Also *TIBCO Object Service Broker Managing Deployment* for information about using MANAGE_RIGHTS.

@MAP

Registers and allocates storage for use with MAP tables. (TBL)

Prerequisites Before you use @MAP, you must define a MAP table using the Table Definer.

See Also *TIBCO Object Service Broker Managing Data* for details on defining and using MAP tables.

Parameters @MAP has the following parameter:

Parameter	Typ	Syn	Len	Description
SCOPE		C	16	The scope of storage. The individual values for this parameter are discussed in the table below.

Values for the SCOPE Parameter

@MAP is parameterized by storage scope and location, and different security controls apply to each of the storage scope parameter values.

TRANSACTION	An insert with a NULL or zero ADDRESS and a positive SIZE allocates and registers memory as requested, and returns its address in @MAP.ADDRESS. The storage is valid only for the life of the current transaction. Both the storage and the corresponding row in @MAP disappear at transaction end. The storage can be both read and written by the user.
SESSION	An insert with a NULL or zero ADDRESS, and a positive SIZE allocates and registers memory as requested, and returns its address in @MAP.ADDRESS. The storage is valid for the life of the TIBCO Object Service Broker session, and can be both read and written by the user.

ENVIRONMENT	<p>An insert with a NULL or zero ADDRESS, and a positive SIZE allocates and registers memory as requested and returns its address in ADDRESS.</p> <p>The storage is SESSION storage that is also made known to any external environment (for example, CICS) and is suitable for use as a COMMAREA.</p> <p>The ADDRESS returned is valid for use as a COMMAREA and can be placed in field COMMHANDLE in the table @SESSION to make this storage into the active COMMAREA.</p> <p>Any number of COMMAREAs can exist at one time but only the one pointed to by @SESSION is recognized by the external environment.</p> <p>You can also allocate ENVIRONMENT space in an environment that does not support COMMAREAs.</p> <p>The storage can be both written and read by users.</p>
EXTERNALRO	<p>An insert with a non-zero ADDRESS registers existing storage at that address.</p> <p>The storage is not known to, or provided by, TIBCO Object Service Broker storage management. Typically this is storage obtained by an external routine that is communicating with TIBCO Object Service Broker rules. TIBCO Object Service Broker does not validity-check the address of EXTERNAL storage at registration time. Addresses that are invalid can cause rejection or failure when actually used as the argument to a MAP table. What addresses are valid depends on the Execution Environment platform and the current operating environment.</p> <p>The storage is read-only memory.</p>

EXTERNALRW	<p>An insert with a non-zero address registers existing storage at that address.</p> <p>The storage is not known to, or provided by, TIBCO Object Service Broker storage management. Typically this is storage obtained by an external routine that is communicating with TIBCO Object Service Broker rules. TIBCO Object Service Broker does not validity-check the address of EXTERNAL storage at registration time. Addresses that are invalid can cause rejection or failure when actually used as the argument to a MAP table. What addresses are valid depends on the Execution Environment platform and the current operating environment.</p> <p>The storage can be both written and read by users.</p>
------------	---

Fields The following are the fields of the @MAP table:

Field	Typ	Syn	Len	Dec	Key	Ord	Rq
KEY (ID-gen style key)	I	B	4	0	P		
ADDRESS (of the block of storage)		B	4	0			
SIZE (storage amount in bytes)		B	4	0			

- Usage Notes**
- Inserting a row in @MAP registers a block of storage for use by MAP tables and can also obtain the storage from the TIBCO Object Service Broker storage manager. No explicit function is provided to delete storage (or corresponding rows in @MAP).
 - Parameter value (PRM) tables and subviews on @MAP are not supported.
 - Selection is supported on @MAP, as well as the INSERT, GET, and FORALL operations. GET and FORALL retrieve the appropriate rows in ascending KEY order. For more information on these operations as used by MAP tables, refer to *TIBCO Object Service Broker Managing Data*.

- When inserting to @MAP, the value used in the SIZE field does not have to allow for the 4 bytes that appear to be used for the KEY field of the MAP table. For Example, suppose the MAP table has fields as shown in this table:

	Xsyn	Xlen	Xdec	Offset	Key
KEY	B	4	0	0	P
F1	C	3	0	0	
F2	C	2	0	3	

Each row of the table occupies 5 bytes (not 9 bytes, because the KEY field occupies no physical storage space). Storage should be allocated in multiples of 5 bytes.

- @MAP behaves like a session table in that its content and effects are local to one TIBCO Object Service Broker session and are not seen by other users, even if they are sharing a single Execution Environment.
- You need to manage the storage you acquire with @MAP. This is especially true for environment and session storage used for programmatically invoked sessions; for example, sessions invoked via the SDKs.

You can reuse storage by having a storage acquisition rule such as the following:

```

GETSESSION(LENGTH) ;
_
_ -----
_
_ GET @MAP('SESSION') WHERE SIZE >= LENGTH;          | 1
_ RETURN(@MAP.ADDRESS);                               | 2
_ -----
_ ON GETFAIL @MAP :
_   @MAP.ADDRESS = 0;
_   @MAP.SIZE = LENGTH;
_   INSERT @MAP('SESSION');
_   RETURN(@MAP.ADDRESS);
_

```

If your application is more complex and requires multiple storage areas, you can use a more complex rule that validates the contents of the storage before returning it for use.

The following is an example from the WMQ interface code that checks to see if a) the length is correct, b) the existing area contains a valid eyecatcher, and c) initializes a new area and eyecatcher if required.

```
@MOMGETMOMMAPA;

-
- -----
- -----+-----
- GET @SESSION(0);                                     1
- GET @MOM_PLATFORM WHERE PLATFORM = @SESSION.PLATFORM; 2
- FORALL @MAP('SESSION') WHERE SIZE = @MOM_PLATFORM.MAP_SIZE 3
-   :
-   GET @MOMMAP(@MAP.ADDRESS);
-   CALL @EQ(@MOMMAP.MANAGER_TYPE, '@MOMMAP');
-   CALL @EQ(@MOMMAP.MANAGER_TYPE, @MOM_PLATFORM.QUEUEMGR);
-   END;
- @MAP.SIZE = @MOM_PLATFORM.MAP_SIZE;                  4
- @MAP.ADDRESS = 0;                                     5
- INSERT @MAP('SESSION');                               6
- @MOMMAP.KEY = 1;                                       7
- @MOMMAP.MANAGER_TYPE = '@MOMMAP';                     8
- REPLACE @MOMMAP(@MAP.ADDRESS);                         9
- RETURN(@MAP.ADDRESS);                                  A
- -----
- ON EQ :
-   RETURN(@MAP.ADDRESS);
```

Exception

DATAREFERENCE	Raised under the following conditions: <ul style="list-style-type: none">• The address value is invalid for the specified scope.• The size value is invalid for the specified scope.• The scope parameter is invalid.
---------------	---

Example The following rule uses @MAP to allocate and register 100 bytes of TRANSACTION storage for use with MAP tables. After the INSERT, the system sets the ADDRESS field of the @MAP table to the address of the allocated storage. At transaction end, the storage and the matching row in @MAP are deleted by the system.

```
@MAP_SAMPLE1;
  LOCAL P;
  -----
  -----+-----
-  @MAP.ADDRESS=0;                               | 1
-  @MAP.SIZE=100;                                | 2
-  INSERT @MAP('TRANSACTION');                   | 3
-  P = @MAP.ADDRESS;                              | 4
-  -----
```

The @MAP_Sample2 Rule

The following rule uses @MAP to register external storage for read-only access by MAP tables. The address (A) and size (S) of the storage block to be registered are passed as arguments to the rule. No storage is allocated by the system; it is the programmer's responsibility to ensure that the address to be registered is valid and points to the desired data in storage. The registration information is not deleted by the system until the TIBCO Object Service Broker session ends.

```
@MAP_SAMPLE2(A,S);
-
- -----
- -----+-----
-  @MAP.ADDRESS = A;                               | 1
-  @MAP.SIZE = S;                                  | 2
-  INSERT @MAP('EXTERNALRO');                      | 3
-  -----
```

MATCH

Returns the starting position, in characters, of the specified pattern in the specified string, relative to the start of the string. (F)

Invocation `position = MATCH(string, pattern)`

<i>position</i>	On return, contains the position of the string. Its syntax is B (binary) with length 2.
<i>string</i>	The string to search for the pattern. Its syntax can be C (fixed-length character string), UN (Unicode), V (variable-length character string), or W (double-byte character).
<i>pattern</i>	The string to be searched for. Its syntax can be C, UN, V, or W.

- Usage Notes**
- Zero (0) is returned if *pattern* is not found or if *string* is a null string.
 - One (1) is returned if a null string is specified for *pattern*. The null string is assumed to match the left-most character of *string*.

Example The following rule determines the starting position of a pattern in a string and prints it to the message log:

```

RULE EDITOR  ==>
MATCH_1;
_ LOCAL SOURCE_STRING, PATTERN, POSITION;
-----
_ SOURCE_STRING = 'THIS IS THE SOURCE STRING';
_ PATTERN = 'SOURCE';
_ POSITION = MATCH(SOURCE_STRING, PATTERN);
_ CALL MSGLOG('THE PATTERN STARTS AT POSITION: ' || POSITION
_ );
-----

```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE PATTERN STARTS AT POSITION: 13

```


MAX

Returns the larger of two given values. (F)

Invocation `bigger = MAX(x, y)`

<code>bigger</code>	The value returned, either <i>x</i> or <i>y</i> .
---------------------	---

<code><i>x</i></code>	A value to be compared to <i>y</i> .
-----------------------	--------------------------------------

<code><i>y</i></code>	A value to be compared to <i>x</i> .
-----------------------	--------------------------------------

Usage Notes The values that you pass to MAX must represent numbers, and can be any syntax.

Example The following rule finds the highest paid employee by comparing each salary in the EMPLOYEE table.

```
HIGH_SALARY;  
  _ LOCAL HIGHEST;  
  _ -----  
  _ -----+-----  
  _ FORALL EMPLOYEE:                                | 1  
  _   HIGHEST = MAX(HIGHEST,EMPLOYEE.SALARY);        |  
  _   END;                                           |  
  _ RETURN(HIGHEST);                                | 2  
  _ -----
```

MESSAGE

Returns a customized message by taking a root message in the MESSAGES table and inserting customizing tokens. (F)

Invocation `m = MESSAGE (utility, msg_num, tokenlist)`

<i>m</i>	On return, contains the message.
<i>utility</i>	The name of the utility where the root message can be found in the MESSAGES table.
<i>msg_num</i>	An integer specifying the root message to use.
<i>tokenlist</i>	A string specifying zero or more tokens to use to customize the root message.

- Usage Notes**
- Message text is contained in the MESSAGES table, which is parameterized by *utility*. You can use this table to contain the messages for your applications. The *utility* parameter value cannot start with an at sign (@).
 - MESSAGE customizes the message by replacing each percent sign (%) in the root message with the next token in the list of tokens. If no such message exists in the MESSAGES table, MESSAGE returns a message indicating this.
 - Separate multiple tokens in *tokenlist* with spaces.
 - No check is made to ensure that the number of customizing tokens is equal to the number of percent signs in the message.

Example The example rule is used by a function key. SCREENMSG uses MESSAGE to return a value and MESSAGE modifies the root message for the EMPLOYEE *utility*:

```

RULE EDITOR ==>                                SCROLL: P
DEL_EMP_2;
-
- -----
- -----+-----
- GET EMPLOYEE_INFO('DELETE_EMPLOYEE');          | 1
- GET EMPLOYEES(EMPLOYEE_INFO.REGION) WHERE EMPNO = | 2
-   EMPLOYEE_INFO.EMPNO;                          |
- CALL CONFIRMACTION('DELETE_EMPLOYEE',          | 3
-   'ABOUT TO DELETE EMPLOYEE ' || EMPLOYEE_INFO.EMPNO,
-   'PF22', '', '', '');                          |
- DELETE EMPLOYEES(EMPLOYEE_INFO.REGION);          | 4
- CALL SCREENMSG('DELETE_EMPLOYEE', MESSAGE('EMPLOYEE', 2, | 5

```

```
- EMPLOYEE_INFO.EMPNO)); |  
- -----
```

In this example, the root message in the MESSAGES(EMPLOYEE) table is:

Deleted employee with no.: %

After deleting the employee, the following screen message appears:

Deleted employee with no.: 99999

MESSAGE_LOG

Preserves the contents of the message log across transactions. (C)

Invocation `CALL MESSAGE_LOG(msglog, destin)`

<i>msglog</i>	<p>Specifies the message log to access. Valid values are:</p> <p>USER – Processes the user log.</p> <p>SYSTEM – Processes the system log.</p>
<i>destin</i>	<p>Specifies the destination for the contents of the log. Valid values are:</p> <p>VIEW – The logs appear as if you pressed PF2 from the workbench. Because you can see both the user and system logs, the <i>msglog</i> argument value is ignored.</p> <p>PRINT – The log is printed.</p> <p>PRT – The log is printed.</p> <p>LOG – The log is saved back to the message log and then the parent transaction can access it. Note Whether or not the system log or user log is saved, it is written to the user log for the parent transaction.</p> <p><i>filename</i> – A string containing a dot (.) is interpreted as a filename. The log is written to the file. A new file is not created unless the file is a partitioned data set; then a new member could be created. The previous contents of the file or member are overwritten.</p> <p>@LOGTEXT(<i>parm</i>) – The log is saved in the @LOGTEXT table in a table instance specified by:</p> <ul style="list-style-type: none"> — The user ID of the person who invoked the rule that called MESSAGE_LOG — A parameter value of your choice (<i>parm</i>) <p>Previous contents of the table instance are overwritten.</p>

Usage Notes

- To send the MESSAGE_LOG to a file in Windows or Solaris, the DSBIFFTYPE Execution Environment parameter must be set to LINE_SEPARATED_ASCII. The default setting is LENGTH_PREFIXED_EBCDIC.



Setting the parameter to LINE_SEPARATED_ASCII affects the behavior of other tools (for example, LOAD, UNLOAD).

- Output to a file and output to a printer are interconnected. If the program is expected to send output to a printer after this output is directed to a file, the print destination must be reset.
- You can view the @LOGTEXT table using the Single Occurrence Editor, but the Table Editor cannot display strings as long as the field in this table. You can copy the contents to any other suitable table.
- The log records are truncated to 76 characters because this is the longest string that the text editor can display on its screen.
- You can give the standard argument values in their lowest unique truncated form. For example, VIEW, VIE, and V all have the same effect.

See Also *TIBCO Object Service Broker Programming in Rules* for more information about the message log.

Exceptions

EMPTY_LOG	Raised if the specified log is empty. If <i>destin</i> is VIEW, both logs are empty.
INVALID_LOG	Raised if the value for <i>msglog</i> does not specify USER or SYSTEM, and the value for <i>destin</i> is not VIEW.
INVALID_LOGDEST	Raised if the value for <i>destin</i> does not specify one of the valid destinations (refer to the previous exception). If you specified the @LOGTEXT(<i>parm</i>) value, the parameter name is not a valid TIBCO Object Service Broker identifier. If you specified a data set name, an error occurred when writing to the data set.
LOGLIMIT	Raised if the value for <i>destin</i> is LOG and the user log overflowed.

Example The set of rules in this example does the following:

- 1. Lists the staff for a manager in the message log
- 2. Reports the manager’s manager in the message log

Rule Using MESSAGE_LOG

The HIERARCHY rule executes the STAFF sample rule, saves the information in the message log by calling MESSAGE_LOG, and then executes MANAGER.

```
HIERARCHY(NUMBER);
- -----
-                                     +-----
- EXECUTE STAFF(NUMBER);                                     | 1
- CALL MESSAGE_LOG('USER', 'LOG');                           | 2
- EXECUTE MANAGER(NUMBER);                                    | 3
- -----
```

Sample Rule 1

The STAFF rule uses the message log to list the people that work for the manager.

```
STAFF(NUM);
- -----
-                                     +-----
- CALL MSGLOG('THE FOLLOWING EMPLOYEES WORK FOR MANAGER ' || | 1
-   NUM || ':');
- GET EMPLOYEE WHERE MGR# = NUM;                               | 2
- FORALL EMPLOYEE WHERE MGR# = NUM:                             | 3
-   CALL MSGLOG(EMPLOYEE.LNAME);
-   END;
- -----
- ON GETFAIL:
-   CALL MSGLOG(NUM || ' IS NOT A MANAGER.');
```

Sample Rule 2

The MANAGER rule uses the message log to display the manager of the manager.

```
MANAGER(NUM);
- -----
-                                     +-----
- CALL MSGLOG(' ');
- GET EMPLOYEE WHERE EMPNO = NUM;                               | 1
- CALL MSGLOG('THE MANAGER OF ' || NUM || ' IS ' ||          | 2
-   EMPLOYEE.MGR#);
-                                     | 3
-                                     |
```

```
-----  
_ ON GETFAIL:  
_ CALL MSGLOG(NUM || ' IS NOT AN EMPLOYEE IN THIS DIVISION.');
```

Results of the HIERARCHY Rule

If the manager number is valid, and the manager is also entered as an employee in the EMPLOYEE table, results such as the following appear in the message log:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ==>                                SCROLL ==> P  
THE FOLLOWING EMPLOYEES WORK FOR MANAGER 79912:  
SMITH  
HRODEK  
WONG  
SCHULTZ  
BOIVIN  
THE MANAGER OF 79912 IS 98895
```

@MESSAGEDUMP

Writes traced messages to this table in HEX form, when @TRACEMESSAGES.DUMP is set to Y. (TBL)

Table Definition

Parameters There is one parameter for this table—LOCATION.

Fields The following are the fields in the @MESSAGEDUMP table:

ID	This field can be used to index into the TAM Table Trace.
TYPE	This field indicates whether the message is outbound (O) or inbound (R) from the Data Object Broker.
OFFSET	This is the hexadecimal offset from start of message of this row.
DUMP	This data contains the hexadecimal interpretation of the data stream as well as the actual character interpretation to the right.

Usage Notes Trace information is written to this file when the value of @TRACEMESSAGES.DUMP is set to Y and then applications to be traced are run. Message tracing can be further refined by modifying other fields of the @TRACEMESSAGES(0) table.

One row is inserted for every 16 bytes of both outbound and reply messages.

Constraints

- Messages greater than X'A0' characters could appear in the wrong sequence: X'A0' displays prior to line X'F0', X'F1', and so on.
- This facility is available only on a z/OS system.



- This table should be used only on the advice of an TIBCO Support representative.
- HEX information can include unprintable characters; therefore, if you want to print the contents of @MESSAGEDUMP, you must first edit out these unprintable characters. Use the Table Editor to do this.

Example The following example illustrates a sample message trace as recorded to @MESSAGEDUMP. The trace contains each message that is sent or received between the current session and the Data Object Broker. The ID field can be used to index to the TAM Table Trace [@MESSAGETRACE](#).

Use the DSECTs MAILOUT, MAILIN and MAILSYNC to interpret the meaning of each message.

TABLE : @MESSAGEDUMP

ID	TYPE	OFFSET	DUMP									
1	O	0	00050000	8160D9							a-R	
1	O	10										
1	R	0	00020000									
1	R	10										
2	O	0	00708070	C7120000	806B7CE4	E2C5D9E2	G				,@USERS	
2	O	10	D6D7E3C9	D6D5E240	4040E3C4	E2400000	OPTIONS				TDS	
2	O	20	00000057	00010048	00487E7E	C1000000					==A	
2	O	30	00000000	00400000	00000000	00000000						
2	O	40	000004A0	00000000	02000001	00010008						
2	O	50	C3007E05	C5E9D3F3	F0001744	C9C30008	C =	USR30			IC	
2	O	60	00000001	48C9C300	08000005	C5E9D3F3		IC			USR3	
2	O	70	F020				0					

@MESSAGETRACE

Stores table access message requests between the Execution Environment and the Data Object Broker collected when using trace facility. (TBL)

Prerequisites @TRACEMESSAGES.TRACE must be set to Y prior to running applications for which trace information is required.

Table Definition

Parameters This table has no parameters.

Fields

ID	Integer identifier incremented by one for each message pair (outbound and reply). Can be reset only by clearing the table via a FORALL loop. Also used as high-order primary composite for table @MESSAGEDUMP.
CODE	Outbound message request type. Refer to CODENAME below.
TABLERNAME	The table name as specified in the outbound message. Not included in certain administrative or transaction messages, such as commits (CODE = 'R').
FIRSTPARM	Contains up to 16 bytes of the value (if any) of the table's first parameter. Not present for certain supervisory messages, nor for messages that are generated internally. Normally can be depended on if the message is explicitly generated by a rule.
UNIQUE	Indicates, when the entry is Y, that a unique selection based on a primary key is supplied. For example, 'GET TABLE WHERE KEY = VALUE ... & ... (... ...)' and other queries of the same form represent unique primary selections. This indicates that a server does not have to sweep the table and is normally the preferred optimization. Although it reduces server work, it does not necessarily reduce message traffic.

RANGE	<p>Y indicates that a query is of the form:</p> <p><i>GET table WHERE key operator somevalue</i></p> <p>where <i>operator</i> is one of <i>></i>, <i>>=</i>, <i><</i>, <i><=</i>.</p> <p>The query can also be of a logically equivalent form. This is normally the second choice for server optimization, as it restricts the range of sweeps by enabling servers to use any index that supports ordering.</p>
SECONDARY	<p>Y indicates that a message contains either unique secondary key values or secondary value ranges. They are provided to the server in a normalized form so that non-unique secondary indexes can take part in data access. Normally the third preferred optimization.</p>
FILTER	<p>Y indicates that fields that are not indexed could possibly need to be examined by the server. This does not preclude the use of indexes. It can require that the server examine each row that is accessed via indexes before returning it.</p>
LOCK	<p>Contains the internal lock code. For example, single a quotation mark (") represents table share; S, row share; and X, row exclusive. Blank does not necessarily indicate that no lock exists; a previous message could have taken an equivalent or larger lock.</p>
LOCKNAME	<p>This field is reserved for future use.</p>
RETURNCODE	<p>This is the internal return code as provided by the reply message.</p>
SIGNAL	<p>This field is reserved for future use.</p>
CODENAME	<p>This is the descriptive name of CODE (from the table @TFMIRS).</p>
LENGTH	<p>This is the outbound message length (the total length as provided to TIBCO Object Service Broker communication services).</p>
RETURNLENGTH	<p>This is the total length of reply message as returned by TIBCO Object Service Broker communication services.</p>

REMOTE	Y indicates that an outbound message is directed to a different Data Object Broker than the local Data Object Broker's NODENAME.
LOCATION	This field contains the Data Object Broker NODENAME that the outbound message attempted to send to, for example, GET TABLE WHERE LOCATION = 'NODENAME'.
TABLETYPE	This is the type of logical table, that is, of TABLENAME above.
VIEWCODE	This field contains the request type of view table, if applicable, for example, G means GET, A means FORALL HEAD, B means FORALL TAIL. This could be different from CODE . For example, a non-optimizable query could generate a server sweep via CODE = 'N' (next rows).
VIEWCODENAME	This field contains the table name as specified by the calling rule, after any indirection has been performed.
VIEWTYPE	This is the table type of VIEWCODENAME .
DATE	This indicates the Julian date of the transaction.
TIME	This indicates the clock time as of time of outbound message send.
PROBABLERULE	This field contains the name of the rule that explicitly caused the message or on whose behalf the message is sent, or the most recently run rule (or possibly a program name). The purpose is to enable the correlation of messages to specific rules.
HEADDRULE	This field contains the name of the starting rule or program name for the transaction.
TRANSACTION	This is the hexadecimal transaction identifier as sent by the outbound message.
RETURNTRAN	This is the hexadecimal transaction identifier as returned by the reply message. NOTE: It could be inapplicable if RETURNCODE is non-zero.
STREAM	This field contains the number of the transaction stream that generated the outbound message.

TRIGGERLEVEL	This field contains a number representing the executor's trigger level. It enables distinction of messages from trigger rules.
MODULE	The name of internal module that generated the message. For TIBCO Support use only.

Usage Notes Trace information is written to this file automatically when the value of @TRACEMESSAGES.TRACE is set to Y and the applications to be traced are run. Message tracing can be further refined by modifying other fields of the @TRACEMESSAGES table.

Since these trace results are in table format, they can be processed by rules. Apart from debugging uses, this table is intended for application profiling; no particular profile is assumed. Applications must define their own tables to hold whatever trace attributes are deemed important for later processing.

This table is subject to the same security clearance checking as all session tables, but the definition must not be changed without corresponding engine changes.

- Constraints**
- Using @MESSAGETRACE to store message information necessarily causes extra data accesses (INSERTs to the @MESSAGETRACE table), and therefore, extra messages during message collection. To avoid this overhead, @TRACEMESSAGES.MESSAGELOG can be set to Y prior to running applications being traced.
 - This facility is available only on a z/OS system.

Example

If you request a generalized trace using an appropriate rule, the contents of @MESSAGETRACE are similar to the following trace. Due to space limitations, not all fields are shown. The output contains timestamps, rule names, and other information.

BROWSING TABLE : @MESSAGETRACE							
COMMAND ==>							
						SCROLL: P	
	ID	CODE	TABLENAME	FIRSTPARM	UNIQUE	RANGE	SECONDARY
-	-	-	-	-	-	-	-
-	1	R					
-	2	G	@USERSOPTIONS		Y		
-	3	G	@USERSOPTIONS		Y		
-	4	G	@MONTH_CODES		Y		
-	5	G	@MONTHLIST		Y		
-	6	G	@WEEKDAYS		Y		
-	7	G	@MONTHLIST		Y		
-	8	G	@RULESLIBRARY	COMMON	Y		
-	9	G	APPOINTMENTS	AZDAO		Y	
-	10	G	APPOINTMENTS	AZDAO		Y	
-	11	G	@MONTH_CODES		Y		

—	12	G	@LIBRARIES		Y
—	13	G	@RULESLIBRARY	TZHAO	Y

PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND

MIN

Returns the smaller of two given values. (F)

invocation `smaller = MIN(x, y)`

<code>smaller</code>	The value returned, either <i>x</i> or <i>y</i> .
----------------------	---

<i>x</i>	A value to be compared to <i>y</i> .
----------	--------------------------------------

<i>y</i>	A value to be compared to <i>x</i> .
----------	--------------------------------------

Usage Notes The values passed to MIN must represent numbers, and can be any syntax.

Example The following rule finds the lowest and highest paid employees by comparing each salary in the EMPLOYEE table.

```

SALARY_RANGE
_  LOCAL  HIGHEST, LOWEST;
_
_  -----
_  LOWEST = 99999;
_  FORALL EMPLOYEE:
_      HIGHEST= MAX(HIGHEST,EMPLOYEE.SALARY);
_      LOWEST = MIN(LOWEST,EMPLOYEE.SALARY);
_      END;
_  CALL MSGLOG('THE HIGHEST SALARY IS ' || HIGHEST);
_  CALL MSGLOG('THE LOWEST SALARY IS ' || LOWEST);
_  -----

```


MINUTE

Returns the minute in the hour the transaction started based on the local machine’s time zone in which the Execution Environment is running. (F)

Invocation time = MINUTE

time On return, contains the minute. Its syntax is C (fixed-length character string) with length 2.

Usage Notes The returned value is a character string containing the number of minutes (00, 01, 02, ..., 59).

Example The following rule determines the minute when the current transaction started and prints it to the message log:

```

      RULE EDITOR  ==>
MINUTE_1;
_  LOCAL TIME;
_  -----
_  -----+-----
_  TIME = HOUR || ':' || MINUTE;                      | 1
_  CALL MSGLOG('THIS TRANSACTION WAS STARTED AT ' || TIME); | 2
_  -----

```

Output for the MINUTE_1 Rule:

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THIS TRANSACTION WAS STARTED AT 13:17

```

@MNG_USERS

Modifies your user security profile. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	Option UP user profile	Press Enter
Developer's workbench	EX Execute Rule option	Type @MNG_USERS(<i>userid</i>) <Enter>
	COMMAND prompt	Type UP <Enter>

Where:

<i>userid</i>	Your user ID.
---------------	---------------

Usage Notes Pressing Enter displays the MANAGE USERID screen of the Security Manager. The screen displays the current values for your user ID.

See Also *TIBCO Object Service Broker Managing Security* for information on the MANAGE USERID screen and the Security Manager.

MOD

Returns the modulus from dividing the dividend by the divisor. The function MOD handles negative dividends and divisors. (F)

Invocation `number = MOD(dividend, divisor)`

<i>number</i>	On return, contains the remainder. Its syntax depends on the syntax of the dividend and divisor.
<i>dividend</i>	The number to be divided. Its syntax can be any of the numeric syntaxes.
<i>divisor</i>	The number to divide by. Its syntax can be any of the numeric syntaxes.

- Usage Notes**
- One or both of the values for *dividend* and *divisor* can be negative.
 - When both the dividend and the divisor are positive, use the [REMAINDER](#) function.
 - The returned value is the modulus of the dividend by the divisor.
 - MOD does not alter the values of either *dividend* or *divisor*.

Exceptions

ZERODIVIDE	Raised if the second operand is zero.
-------------------	---------------------------------------

Example The following rule determines the modulus of the division operation and prints it to the message log:

<pre> RULE EDITOR ==> MOD1(DIVIDEND,DIVISOR); - LOCAL RESULT; - ----- - -----+----- - RESULT = MOD(DIVIDEND, DIVISOR); - CALL MSGLOG(DIVIDEND ' MOD ' DIVISOR ' IS ' - RESULT); - ----- </pre>	<pre> SCROLL: P 1 2 </pre>
---	----------------------------------

Resulting Output

Executing this rule with a dividend of -12 and a divisor of 10 displays the following to the message log.

```
-----INFORMATIONAL MESSAGE LOG -----  
COMMAND ==>                                SCROLL===> P  
-12 MOD 10 IS -2
```

@MOMCLOSE

Closes a Message Oriented Middleware (MOM) message queue. (F)

Invocation `queue = @MOMCLOSE(connection , queue)`

<code>queue</code>	A token describing the queue. Set by @MOMOPEN. Reset by @MOMCLOSE.
--------------------	--

<code>connection</code>	A token describing the queue manager. Returned by @MOMCONNECT.
-------------------------	--

Exceptions

<code>MOM_INV_@MOMMAP</code>	The address received does not point to a valid control block.
------------------------------	---

<code>MOM_INV_COMMAND</code>	The command received is invalid.
------------------------------	----------------------------------

<code>MOM_INV_MOMMSG</code>	The control message received is invalid.
-----------------------------	--

<code>MOM_SMALL_MAPSTG</code>	The control block passed is too small.
-------------------------------	--

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMCOMMIT

Commits all changes to queues from a single Message Oriented Middleware (MOM) message manager. (C)

Invocation `CALL @MOMCOMMIT(connection)`

connection A token describing the queue manager. Set by @MOMCONNECT.

- Usage Notes**
- When running under the CICS MQSeries Adapter, the MQSeries COMMIT verb is not allowed. CICS handles the committing of changes at the end of the CICS transaction or when the “EXEC CICS SYNCPOINT” command is issued.
 - @MOMCOMMIT, when run under the CICS MQSeries Adapter, generates an “EXEC CICS SYNCPOINT” to commit MQSeries changes. Any other pending CICS changes are also committed at this time.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMCONNECT

Connects to a Message Oriented Middleware (MOM) message queue (MQ) manager. (F)

Invocation `connection = @MOMCONNECT(name)`

<i>connection</i>	A token describing the queue manager. Used in subsequent @MOM calls.
-------------------	--

<i>name</i>	The name of the queue manager, or blank when running on a CICS system with the CICS MQSeries Adapter active.
-------------	--

Usage Notes

- For TIBCO Object Service Broker WebSphere MQ Integrator, it is possible to get an error if an earlier session was using the z/OS Task Control Block (TCB) that the current session is using and ended or failed without issuing a disconnect. @MOMCONNECT rolls back the existing MQS session and establishes a new one.
- Each session accessing an MQ manager takes exclusive control of the TCB where it is running. This has no impact for a single-user Execution Environment, that is, one running in batch or under TSO. For a multi-user Execution Environment, that is, one running under CICS or a Native Execution Environment, you must ensure that you have enough TCBs for these sessions and your other work. You specify this number in the TASKEEXECNUM Execution Environment parameter.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
------------------------	---

MOM_INV_COMMAND	The command received is invalid.
------------------------	----------------------------------

MOM_INV_MOMMSG	The control message received is invalid.
-----------------------	--

MOM_SMALL_MAPSTG	The control block passed is too small.
-------------------------	--

Example

Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMDISCONN

Disconnects from a Message Oriented Middleware (MOM) message manager. (F)

Invocation `connection = @MOMDISCONN(connection)`

<code>connection</code>	A token describing the queue manager. Set by @MOMCONNECT. Reset by @MOMDISCONN.
-------------------------	---

Exceptions

<code>MOM_INV_@MOMMAP</code>	The address received does not point to a valid control block.
<code>MOM_INV_COMMAND</code>	The command received is invalid.
<code>MOM_INV_MOMMSG</code>	The control message received is invalid.
<code>MOM_SMALL_MAPSTG</code>	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMGET

Reads a message from a Message Oriented Middleware (MOM) message queue.
(C)

Invocation `CALL @MOMGET(connection, queue, table)`

<i>connection</i>	A token describing the queue manager. Returned by @MOMCONNECT.
<i>queue</i>	A token describing the queue. Returned by @MOMOPEN.
<i>table</i>	The name of a MAP table to be read from the @MOM queue.

Usage Notes If *table* is null, the address and length of the returned record are in @MQSMAP.BUFFER_ADDRESS and @MQSMAP.DATA_LENGTH.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SHUTDOWN	A SHUTDOWN control message was received.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: @MOMINIT, @MOMCONNECT, @MOMOPEN, @MOMPUT, @MOMCOMMIT, @MOMROLLBACK, @MOMSPECIALCMD, @MOMOPTION, @MOMSETOPT, @MOMMAPLENGTH, @MOMVALIDRC, @MOMCLOSE, @MOMDISCONN, @MQSMAP and @MQSMAP_PORT.

@MOMINIT

Identifies the type of Message Oriented Middleware (MOM) message manager, and initializes its environment (map and control structures) to enable subsequent @MOM calls. (C)

Invocation `CALL @MOMINIT(buflen, mom_type)`

<i>buflen</i>	The size of the largest message to be processed.
<i>mom_type</i>	The type of queue manager you want to use in the current session. If <i>mom_type</i> is null, an appropriate queue manager is selected. The only queue manager currently available is MQSERIES.

- Usage Notes**
- If you prefer to allocate your own record storage map table, set *buflen* to 0 and update @MQSMAP.BUFFER_ADDRESS and @MQSMAP.BUFFER_LENGTH with the corresponding information after the @MOMINIT call.
 - WebSphere MQ on z/OS is TCB-specific. Therefore, @MOMINIT sets TCB affinity for the transaction and an interpreter TCB is held for the duration of the transaction. In a multi-user Execution Environment, consider increasing the value of your TASKEEXECNUM Execution Environment parameter.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_NO_MQSSTUB	The stub program for WebSphere MQ has not been link-edited into the HRNDRSES load module.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

The *TIBCO Object Service Broker Parameters* manual for information on the TASKEXECNUM Execution Environment parameter

Related tools: [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMMAPLENGTH

Returns the length of a MAP table. (F)

Invocation `var = @MOMMAPLENGTH(table_name)`

<i>var</i>	A variable to hold the length of the <i>table_name</i> table.
------------	---

<i>table_name</i>	The name of the MAP table.
-------------------	----------------------------

Usage Notes This is a time-consuming rule to run (multiple Data Object Broker accesses are required). If you need to calculate a table length, it is better to do it once prior to any processing loop.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMOPEN

Opens a Message Oriented Middleware (MOM) message queue. (F)

Invocation `queue = @MOMOPEN(connection, name)`

<code>queue</code>	A token describing the queue.
<code>connection</code>	A token describing the queue manager. Returned by @MOMCONNECT.
<code>name</code>	The name of the queue.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMOPTION

Queries the numeric equivalent of a Message Oriented Middleware (MOM) option. (F)

Invocation `value = @MOMOPTION(description)`

<code>value</code>	The numeric equivalent of the option description.
--------------------	---

<code><i>description</i></code>	The symbolic name used for an option setting in the MOM manager documentation.
---------------------------------	--

- Usage Notes**
- MOM applications generally use symbolic descriptions heavily for many numeric values. @MOMOPTION makes the same descriptions available for your use.
 - This table is complete at the time of publication. As new levels of MOM manager software are released, you can add your own values to the @MOMOPTIONS table until upgrades are available.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMPUT

Writes a message to a Message Oriented Middleware (MOM) message queue. (C)

Invocation `CALL @MOMPUT(connection, queue, table, len)`

<i>connection</i>	A token describing the queue manager. Returned by @MOMCONNECT.
<i>queue</i>	A token describing the queue. Returned by @MOMOPEN.
<i>table</i>	The name of a map table to be written to the @MOM queue.
<i>len</i>	The length of the table.

Usage Notes

- If *table* is null, @MQSMAP.BUFFER_ADDRESS and @MQSMAP.DATA_LENGTH must be set prior to the call.
- If *table* is supplied and *len* is null, the length of the table is used. Calculation of the length of the table is time-consuming, because you need to access the Data Object Broker. If you are doing multiple @MOMPUT calls, it is better to use @MOMMAPLENGTH(table) to store the length in a local variable and use the local variable in the @MOMPUT call.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example

Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also

TIBCO Object Service Broker for z/OS External Environments or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: @MOMINIT, @MOMCONNECT, @MOMOPEN, @MOMGET, @MOMCOMMIT, @MOMROLLBACK, @MOMSPECIALCMD, @MOMOPTION, @MOMSETOPT, @MOMMAPLENGTH, @MOMVALIDRC, @MOMCLOSE, @MOMDISCONN, @MQSMAP and @MQSMAP_PORT.

@MOMROLLBACK

Backs out all database changes from a single Message Oriented Middleware (MOM) message manager since the start of the transaction or since the previous @MOMCOMMIT. (C)

Invocation CALL @MOMROLLBACK(*connection*)

connection A token describing the queue manager. Set by @MOMCONNECT.

- Usage Notes**
- Under the CICS MQSeries Adapter, the MQSeries ROLLBACK verb is disallowed. CICS handles the roll back of changes by CICS transaction abend or when the “EXEC CICS SYNCPOINT ROLLBACK” command is issued.
 - @MOMROLLBACK, when run under the CICS MQSeries Adapter, generates an “EXEC CICS SYNCPOINT ROLLBACK” to roll back MQSeries changes. Any other pending CICS resource changes are also rolled back at this time.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMSETOPT

Sets a MOM option to a specified value. (C)

Invocation `CALL @MOMSETOPT(description)`

description The symbolic name used for an option setting in the MOM manager documentation.

- Usage Notes**
- This is a short way of setting a single value into the @MQSMAP.OPTIONS field, that is, a short form of @MQSMAP.OPTIONS = @MOMOPTION('description').
 - To set multiple values, use the @MOMOPTION tool; for example, @MQSMAP.OPTIONS = @MOMOPTION('desc1') + @MOMOPTION('desc2').

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMSPECIALCMD

Sends a Message Oriented Middleware (MOM) command to a queue listener task.
(C)

Invocation CALL @MOMSPECIALCMD(*manager_name*, *queue_name*, *command*)

<i>manager_name</i>	The name of the queue manager.
---------------------	--------------------------------

<i>queue_name</i>	The name of the queue.
-------------------	------------------------

<i>command</i>	The command to be issued.
----------------	---------------------------

Usage Notes At present, the only valid command is SHUTDOWN. This tells a queue listener task to come out of its wait state and causes the MOM_SHUTDOWN exception to be raised. This exception can then be trapped by the listening rule.

Exceptions

MOM_INV_@MOMMAP	The address received does not point to a valid control block.
------------------------	---

MOM_INV_COMMAND	The command received is invalid.
------------------------	----------------------------------

MOM_INV_MOMMSG	The control message received is invalid.
-----------------------	--

MOM_SMALL_MAPSTG	The control block passed is too small.
-------------------------	--

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

@MOMVALIDRC

Checks the return code of a previous command. (C)

Invocation `CALL @MOMVALIDRC`

- Usage Notes**
- For successful return codes, no action is taken.
 - For warning return codes, the reason code is translated to the corresponding message string and appears on the message log.
 - For error return codes, the MOM_command_FAIL exception is raised.
 - @MOMVALIDRC is a sample error checking routine. For errors that you expect and need to handle, you should code your own routine.

Exceptions

INVALID_MOMMGR	The mom_type received is invalid.
MOM_command_FAIL	The indicated <i>command</i> failed.
MOM_INV_@MOMMAP	The address received does not point to a valid control block.
MOM_INV_COMMAND	The command received is invalid.
MOM_INV_MOMMSG	The control message received is invalid.
MOM_SMALL_MAPSTG	The control block passed is too small.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for additional information on using the MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMCLOSE](#), [@MOMDISCONN](#), [@MQSMAP](#) and [@MQSMAP_PORT](#).

\$MOVECONTAINER

Moves a container and its contents from one channel to another. Afterwards, the source container no longer exists. (C)

Invocation `CALL $MOVECONTAINER(frchannel, frcontainer, tochannel, tocontainer)`

<i>frchannel</i>	The name (1-16 characters) of the channel that owns the source container.
------------------	---

<i>frcontainer</i>	The name (1-16 characters) of the source container to be moved.
--------------------	---

<i>tochannel</i>	The name (1-16 characters) of the channel that owns the target container.
------------------	---

<i>tocontainer</i>	The name (1-16 characters) of the target container. If the latter already exists, \$MOVECONTAINER overwrites its contents.
--------------------	--

Usage Notes You can move a container in either of these ways:

- From one channel to another.
- Within the same channel, for example, from within the current channel. Doing so renames the container.

Instead of \$GETCONTAINER and \$PUTCONTAINER, you can use \$MOVECONTAINER as a more efficient way of transferring data between channels.

MOVTAB

Changes the segment number of a table. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	DT Define Table option	Press Enter. A listing of tables for your TIBCO Object Service Broker database appears. Type the line command M beside the table whose segment number is being changed and press Enter. A screen similar to the one below appears.
From a rule		Type CALL MOVTAB (<i>tablename</i> , <i>segmentid</i>)

Where:

<i>tablename</i>	The name of the table for which you want to change the segment number.
<i>segmentid</i>	The segment number of the Pagestore segment to which you want the number changed.

To complete this command:

	NAME	TYPE	CREATOR	CREATD	UNIT	MODIFIER	MODIFD	BI*
M	EMPLOYEE_EXPENSE	TDS	USR50	1999-12-13	JZH	USR50	2000-03-16	
Enter parameter(s): Destination segment number								
	SEGMENT#	==>						

PFKEYS: ENTER=PROCESS 3=PROCESS 12=CANCEL

In the field **SEGMENT #**, type the new segment number and press Enter or PF3 to process the change. Press PF12 to cancel the change.



MOVTAB does not take part in the TIBCO Object Service Broker two-phase commit/intent list protocol. We strongly recommend that you do not use this tool in a transaction that accesses or updates data within the same table. It should also normally be the only logical unit of work within a transaction.

Usage Notes

- MOVTAB is used to change the segment number of a table. This number is stored in the Resident Table Index (RTIX). The change occurs at the physical level and does not affect the logical view of that table.
- The input table must be a defined TDS table and it must be empty.
- MOVTAB ignores the location parameter of the input table. Minimal table definitions are not allowed.
- The input table cannot be a TIBCO Object Service Broker system table (for example, Table of Tables or Table of Fields).
- The target segment must be online.
- MOVTAB runs and updates are made even if it is called in a transaction that is running in browse mode.

Tables Containing Data

If the table contains data, complete the following steps:

1. Use **UNLOAD** to export a copy of only the data to a temporary file.



Do not UNLOAD the table definition as well. This would cause the LOAD in step #4. to fail, because the table already exists.

If the table is large, consider using one of the batch unload utilities to unload the data to a file. For more information on the batch unload, refer to *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities*.

2. Use **\$CLRTAB** to clear all data from the table.
3. Use MOVTAB to change the segment number of the table.
4. Use **LOAD** to reload the data.

To load large volumes of data quickly, consider using the batch load utilities. For more information on the batch load, refer to *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities*.

Exceptions

DEFINITIONFAIL	Raised if the definition of the table does not exist or is inconsistent.
SECURITYFAIL	Raised if the user is not authorized to change the definition of the table.
ROUTINEFAIL	Raised if the definition of the table is minimal or the table type is not TDS.

@MQSMAP and @MQSMAP_PORT

Registers and allocates storage for use with the MQSMAP table. @MQSMAP is for use on z/OS and @MQSMAP_PORT on Open Systems. (TBL)

Usage Notes When you want to make WebSphere MQ API calls to set WebSphere MQ options, you modify the values in this table as required.

Table Fields The fields of this table are mapped to the fields for the WebSphere MQ API. The API fields are listed in the IBM WebSphere MQ documentation, as well as in the cmqc.h WebSphere MQ file for Open Systems, written to your computer when you installed WebSphere MQ.

Here are some of the fields you must check or modify (to use one of the last three fields listed in this table, replace the ellipsis with the name of the option you want to check or change):

Field Name	Description
RETURN_CODE	The return code from WebSphere MQ.
REASON_CODE	The reason code from WebSphere MQ.
BUFFER_ADDRESS	The address of the message buffer used to contain WebSphere MQ messages.
BUFFER_LENGTH	The length of the message buffer.
DATA_LENGTH	The length of the message in the message buffer.
OPTIONS	The options to be used on an WebSphere MQ call.
QO_...	Queue options. Mapped by the CMQODA macro.
MO_...	Message options. Mapped by the CMQMDA macro.
GO_...	Get options. Mapped by the CMQGMOA macro.
PO_...	Put options. Mapped by the CMQPMOA macro.

Initialization This map table is initialized by @MOMINIT and contains reasonable defaults for most WebSphere MQ commands. You modify these defaults to set WebSphere MQ options.

Example Refer to the example in the section on the Service Gateway for WMQ in *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments*.

See Also *TIBCO Object Service Broker Managing Data* for details on defining and using MAP tables.

TIBCO Object Service Broker for z/OS External Environments or *TIBCO Object Service Broker for Open Systems External Environments* for information on using the various MOM tools and the Service Gateway for WMQ.

Related tools: [@MOMINIT](#), [@MOMCONNECT](#), [@MOMOPEN](#), [@MOMPUT](#), [@MOMGET](#), [@MOMCOMMIT](#), [@MOMROLLBACK](#), [@MOMSPECIALCMD](#), [@MOMOPTION](#), [@MOMSETOPT](#), [@MOMMAPLENGTH](#), [@MOMVALIDRC](#), [@MOMCLOSE](#), and [@MOMDISCONN](#).

MSGLOG

Inserts the specified string as a line in the TIBCO Object Service Broker message log. (C)

Invocation `CALL MSGLOG(string)`

<i>string</i>	A character string to insert in the user log. Its syntax can be C (fixed-length character string), UN (Unicode), V (variable-length character string), or W (double-byte character).
---------------	--

- Usage Notes**
- Although longer lengths could be passed in, only the first 256 characters appear.
 - If the value of *string* is of syntax UN and it cannot be coerced to syntax V, MSGLOG inserts a Unicode literal.

Exceptions

LOGLIMIT	The user log is full. LOGLIMIT is controlled by the MSGLOGMAX Execution Environment parameter.
-----------------	--

Example The STAFF Rule

The following rule finds all the employees of a particular manager and uses MSGLOG to list them in the message log.

```

RULE EDITOR ==>
STAFF(NUM);
--
-- -----
-- CALL MSGLOG('THE FOLLOWING EMPLOYEES WORK FOR MANAGER ' ||
--   NUM || ':' );
-- FORALL EMPLOYEE('MIDWEST') WHERE MGR# = NUM :
--   CALL MSGLOG(EMPLOYEE.LNAME);
--   END;
-- -----
--

```

Results of the STAFF Rule

Executing the rule for manager 79912 produces the following message log:

----- INFORMATIONAL MESSAGE LOG -----

COMMAND ==>

SCROLL ==> P

THE FOLLOWING EMPLOYEES WORK FOR MANAGER 79912:

SMITH

HRODEK

SCHULTZ

BOIVIN

\$NEWPAGE

Positions subsequent output to the top of a new page. (C)

Invocation CALL \$NEWPAGE

- Usage Notes**
- The print arguments must have been previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to \$NEWPAGE.
 - \$NEWPAGE has no effect if subsequent output begins on a new page anyway.

Exceptions

LOGLIMIT	Too much output is sent to the message log.
ROUTINEFAIL	\$NEWPAGE is not preceded by a call to \$SETPRINT or \$RESETPRINT.
STRINGSIZE	This exception is raised if the left, center, or right titles overlap, or if the combined length of character strings exceeds <i>width</i> (where <i>width</i> is the page width set by \$SETPRINT or \$RESETPRINT) or 132, whichever is less.

Example Displayed Output for NEWPAGE_1

The following rule prints two pages to the message log:

```

NEWPAGE_1;
-
- -----
-                                     +-----+
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N');           | 1
- CALL $PRINTLINE(                                | 2
-   'THIS IS THE FIRST LINE OF THE FIRST PAGE. '); |
- CALL $NEWPAGE;                                   | 3
- CALL $PRINTLINE(                                | 4
-   'THIS IS THE FIRST LINE OF THE SECOND PAGE. '); |
- -----

```

Rule Using \$NEWPAGE

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P

```

----- NEW PAGE -----

THIS IS THE FIRST LINE OF THE FIRST PAGE. Page 1

----- NEW PAGE -----

THIS IS THE FIRST LINE OF THE SECOND PAGE. Page 2

NLS

Enables the database administrator to set code page values in translation tables.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type NLS <Enter>
	COMMAND prompt	Type EX NLS<Enter>

Usage Notes You must execute NLS from a level-7 user ID.

Code Page Lookup

When determining if translation is to be performed for external DBMS data, TIBCO Object Service Broker looks at code pages in the following order:

1. Table name
2. Server ID (for a peer server or Service Gateway for Files)
3. Table type
4. If no code page is found, no conversion is performed

NLS Screen

On entering the NLS tool, the following screen appears:

```

National Language Support Manager

TDS and peer to peer Code Page Settings
  Locale.Codepage
SELF          ENGL.IBM-037
REMOTE        ENGL.IBM-037
Service Gateway for Files Code Page Settings (By Server ID)
Server ID     Locale.Codepage
* NO CODEPAGE *

External DBMS Code Page Settings (By Table Type)
Type          Locale.Codepage
* NO CODEPAGE *

External DBMS Code Page Settings (By Server ID)
Server ID     Locale.Codepage
* NO CODEPAGE *
```



```
External DBMS Code Page Settings (By Table Name)
Table Name      Type Locale.Codepage
#ACT1           DAT DVDFFGF
@IDMS_ELEMENTS      * SECURITYFAIL *
PFKEYS: 3=EXIT 5=EDIT 6=ADD 12=EXIT
Place cursor on appropriate entry then PF5 to edit or PF6 to add
```

There are five areas to set code pages, as shown in the following sections.

TDS and peer to peer Code Page Settings

In this area, you set the SELF and REMOTE code pages. SELF is the code page for user character data stored in user TDS tables in the local Data Object Broker and REMOTE is the code page for character data in peer-to-peer communication.

Service Gateway for Files Code Page Settings (By Server ID)

All accesses for a specific Server ID are translated. For example, you could have two different remote instances of the Service Gateway for Files with different code page settings.

External DBMS Code Page Settings

In the next three areas, you specify the code pages of your external DBMSs. You can do this at three levels of detail.

By Table Type	All accesses for the specified table type are translated (for example, DB2).
By Server ID	All accesses for a specific Server ID are translated. For example, you could have two different DB2 gateways with different code page settings.
By Table Name	All accesses to a specific table are translated.

Pressing PF5 on the appropriate section brings up a Table Editor window where you can edit the table. Here are some examples:

Service Gateway for Files

```
EDITING TABLE      :  @SERVERCONFIG(_FG)
COMMAND ==>
```

SCROLL: P

	NAME	TYPE	SYNTAX	LENGTH	DECIMAL	VALUE	
MODIFY							
—	-----	-	-	-----	-----	-----	-
—	SERVERLS	S	V	32		SVEN.IBM-278	N

By Table Type

EDITING TABLE : @SERVERCONFIG(_NT)						
COMMAND ==>						
NAME		TYPE	SYNTAX	LENGTH	DECIMAL	VALUE
MODIFY						
-----		-	-	-----	-----	-----
_ DB2		S	V	32	0	ENGL.IBM-500
_ IMS		S	V	32	0	ENGL.IBM-500

By Server ID

EDITING TABLE : @SERVERCONFIG(_NS)						
COMMAND ==>						
NAME		TYPE	SYNTAX	LENGTH	DECIMAL	VALUE
MODIFY						
-----		-	-	-----	-----	-----
_ DB2ITAL		S	V	32		ENGL.IBM-280
_ DB2ESPA		S	V	32		ENGL.IBM-284

By Table Name

EDITING TABLE : @SERVERPARMS(DB2CUST)										
COMMAND ==>										
NUMBER	NAME	TYPE	SYNTAX	LENGTH	DECIMAL	DEFAULT	USAGE	SCOPE		
MUSTINCLDEFN										
—	-----	-	-	-----	-----	-----	-	-	-	-
—	1	SERVERID	S	C	8	0	DEFAULT	C	Y	Y
—	2	SERVERTYPE	S	C	3	0		T	Y	Y
—	3	CODEPAGE	S	V	32	0	NORS.IBM-277	Y	Y	N

NOOP

Does nothing. (C)

Invocation

CALL NOOP

Usage Notes

NOOP can be used when you need to invoke a tool due to syntax or coding conventions.

NUM_CHK

Determines if a given string satisfies the TIBCO Object Service Broker definition of a numeric literal. (F)

Invocation `verify = NUM_CHK(val)`

<code>verify</code>	On return, contains the value Y if <i>val</i> is a numeric literal or N if <i>val</i> is not a numeric literal.
---------------------	---

<code>val</code>	A value.
------------------	----------

See Also *TIBCO Object Service Broker Programming in Rules* for a description of valid TIBCO Object Service Broker numeric literals.

Example Rule Using NUM_CHK

The following rule verifies that you provide a number for a rule:

```

VERIFY_INPUT(INPUT);
_ LOCAL NUMBER;
-----
_ NUM_CHK(INPUT) = 'Y';                                     | Y N
-----+-----
_ NUMBER = INPUT;                                           | 1
_ CALL CALCULATE;                                           | 2
_ CALL ENDMSG(INPUT || ' IS NOT A VALID NUMBER. ');        | 1
-----

```

- If you provide valid input, for example, `VERIFY_INPUT(56)`, the `CALCULATE` rule is called to perform arithmetic operations on the number.
- If the input is not valid, for example, `VERIFY_INPUT(AB)`, you receive the message:

AB IS NOT A VALID NUMBER

OBJECT_MGMT

Displays the contents of a table and enables a predefined set of commands that are unique to the table to operate on the display. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type OBJECT_MGMT (<i>tablespec</i>) <Enter>
	COMMAND prompt	Type EX OBJECT_MGMT (<i>tablespec</i>) <Enter>

Where:

<i>tablespec</i>	<p>A string specifying the table name (and parameters, if any). Selection and ordering can also be specified. If used:</p> <ul style="list-style-type: none"> • Selection must begin with the keyword WHERE. • Ordering must begin with the keyword ORDERED. • No abbreviations can be used.
------------------	---

- Usage notes**
- [OBJECTMGR](#) is the version of this tool called from within a rule, a version that does not trap exceptions. To trap exceptions, use OBJECT_MGMT.
 - If you do not supply a value for *tablespec*, executing OBJECT_MGMT displays a screen prompting for a value.
 - After executing OBJECT_MGMT, the specified table appears. This table is usually a subview of a master object table.
 - Further selecting and ordering can be specified through primary commands, selecting and ordering that is done only on the subset specified in *tablespec*.
 - OBJECT_MGMT (and its interactive version OBJECTMGR) uses [DEFINE_OBJLIST](#) when it is invoked. DEFINE_OBJLIST defines the appearance of the display and the commands to be used, and arranges up to two lines of titling. This is applied against the table given as an argument to OBJECT_MGMT.

- OBJECT_MGMT should run in BROWSE mode so that the commands can update the objects selected.
- The object commands operate on the primary key. The primary key field cannot be a composite primary key field.
- The valid object commands appear across the bottom of the screen.
- The following primary commands and function keys are predefined to OBJECT_MGMT:

Primary Commands	Keys	Purpose
	Enter	Executes command.
	PF3	Cancels and exits.
	PF12	Cancels and exits.
	PF7	Scrolls up.
	PF8	Scrolls down.
	PF9	Recalls last command.
SEL/SELECT		Selects specified object.
F/FIND		Finds specified object.
	PF5	Finds the next occurrence.
ORDERED		Orders occurrences by field.

Example This example uses both the table and the information presented during the description of the fields of the DEFINE_OBJLIST tool. Executing OBJECT_MGMT against the table EMPLOYEES_0DPARM displays the following screen. Refer to DEFINE_OBJLIST to see how this table is formatted:

Command ==>					Scroll P		
EMPNO	LNAME	POSITION	MGR#	DEPTNO	SALARY	ADDR *	
— 80000	SMYTHE	DIRECTOR	80002	20	986.73		
— 80002	ROTTERDAM	VP	99999	50	234.84		
— 80003	CHANG	ASSOC. ANALYST	83020	10	589.91		
— 80004	GARZA	ANALYST	80009	30	574.16		
— 80005	HANSON	MGR	83020	20	890.00		
— 80006	MILMAN	ANALYST	84021	10	699.49		
— 80008	HONCHRSKY	STAFF	84021	50	711.19		
— 80009	CHESTERTON	PROGRAMMER	80000	70	978.76		

—	80014	TOWNSEND	PRESIDENT & GM	84021	70	230.23
—	80019	PASTARINA	MGR	80033	50	100.00
—	80020	CHESSELL	SECRETARY	83020	30	301.00
—	80021	TOWNESEND	RECEPTIONIST	84021	50	600.01
—	80024	NAPIER	SALES PERSON	80020	20	340.40
—	80033	CANNON	PROGRAMMER	80020	40	0.00

PFKEYS: 12=EXIT 13=PRINT 3=END 5=FIND NEXT 9=RECALL

OBJECTMGR

Displays the contents of a table and enables a predefined set of commands that are unique to the table to operate on the display. (C)

Invocation `CALL OBJECTMGR(tablespec)`

<i>tablespec</i>	<p>The table name (and parameters, if any). Selection and ordering can also be specified. If used:</p> <ul style="list-style-type: none"> • Selection must begin with the keyword <code>WHERE</code>. • Ordering must begin with the keyword <code>ORDERED</code>. • No abbreviations can be used.
------------------	---

Usage Notes

- [OBJECT_MGMT](#) is the interactive version of this tool.
- Further selecting and ordering can be specified through primary commands, selecting and ordering that is done only on the subset specified in *tablespec*.
- After calling `OBJECTMGR` the specified table appears. This table is usually a subview of a source object table.
- `OBJECTMGR` and its related version `OBJECT_MGMT` use [DEFINE_OBJLIST](#) when they are invoked. `DEFINE_OBJLIST` defines the appearance of the display and the commands to be used, and arranges up to two lines of titling. This is applied against the table given as an argument to `OBJECTMGR`.
- Run `OBJECTMGR` in browse mode so that the commands can update the objects selected.
- The object commands operate only on the primary key. The primary key cannot be a composite primary key.
- The valid object commands appear across the bottom of the screen.
- The following primary commands and function keys are predefined to `OBJECTMGR`:

Primary Commands	Keys	Purpose
	Enter	Executes command.
	PF3	Cancels and exit.s
	PF12	Cancels and exits.

Primary Commands	Keys	Purpose
	PF7	Scrolls up.
	PF8	Scrolls down.
	PF9	Recalls last command.
SEL/SELECT		Selects specified object.
F/FIND		Finds specified object.
	PF5	Finds the next occurrence.
ORDERED		Orders occurrences by field.

See Also For a complete description of the primary commands, refer to the *TIBCO Object Service Broker Getting Started* manual.

Exceptions

PARSER_ERROR	Raised if there is a syntax error in the <i>tablespec</i> .
NO_OBJECTS	Raised if the list is empty.
SECURITYFAIL	Raised if the caller does not have security clearance to read either the list or some other table used by the Object Manager.
ERROR_IN_SYNTAX	Raised if there is an error in the definition of the table containing the list that prevents the Object Manager from building its display.
TABLE_NOT_FOUND	Raised if the table to contain the list does not exist.

Example This example uses both the table and the information presented during the description of the fields of the DEFINE_OBJLIST tool. Executing OBJECTMGR against the table EMPLOYEES_ODPARM displays the following screen. Refer to DEFINE_OBJLIST to see how this table is formatted.

Weekly Report						
Command ==>						Scroll P
EMPNO	LNAME	Employee Salaries POSITION	MGR#	DEPTNO	SALARY	ADDR*
— 80000	SMYTHE	DIRECTOR	80002	20	986.73	
— 80002	ROTTERDAM	VP	99999	50	234.84	

—	80003	CHANG	ASSOC.ANALYST	83020	10	589.91
—	80004	GARZA	ANALYST	80009	30	574.16
—	80005	HANSON	MGR	83020	20	890.00
—	80006	MILMAN	ANALYST	84021	10	699.49
—	80008	HONCHRSKY	STAFF	84021	50	711.19
—	80009	CHESTERTON	PROGRAMMER	80000	70	978.76
—	80014	TOWNSEND	PRESIDENT & GM	84021	70	230.23
—	80019	PASTARINA	MGR	80033	50	100.00
—	80020	CHESSELL	SECRETARY	83020	30	301.00
—	80021	TOWNESEND	RECEPTIONIST	84021	50	600.01
—	80024	NAPIER	SALES PERSON	80020	20	340.40
—	80033	CANNON	PROGRAMMER	80020	40	0.00

D-Delete S-Select
PFKEYS: 12=EXIT 13=PRINT 3=END 5=FIND NEXT 9=RECALL

@OPENDSN

Specifies the name of the file that is subsequently used by [@READDSN](#) or [@WRITEDSN](#). (C)

Invocation `CALL @OPENDSN(dsname)`

<i>dsname</i>	A character string specifying the file to open. Its syntax is V (variable-length character string). It has a length of 54 for z/OS, 512 for Windows, and 1023 for Solaris.
---------------	--

Usage Notes

If <i>dsname</i> is...	Do this
A z/OS data set	Specify a fully qualified data set name. It can include a member name. For example, a valid <i>dsname</i> is: AAAAAA.DATA(SOURCE)
A Windows or Solaris file	Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to <i>TIBCO Object Service Broker Parameters</i> for more information about this parameter.

- [@WRITEDSN](#) and [@READDSN](#) are used to write to and read from the file.
- [@OPENDSN](#) identifies the file. An attempt to open the file is made with the first read or write operation.
- [@OPENDSN](#) works only within the scope of the transaction, rather than of the session; it does not cross transaction boundaries.
- [@OPENDSN](#) accesses a z/OS file using the data set name. There is no provision for using a DDNAME with this tool instead of a data set name.

Example The following rule:

1. Specifies the name of an existing file
2. Writes data from the example table to it
3. Closes the file
4. Re-specifies it

- 5. Reads back the first record from it
- 6. Prints that record to the message log:

```
OPENDSN_1;  
  _ LOCAL RECORD;  
  _ -----  
  _ -----+-----  
  _ CALL @OPENDSN(TSOID || ' .EXAMPLES.DATA'); | 1  
  _ FORALL EMPLOYEE : | 2  
  _   CALL @WRITEDSN(EMPLOYEE.LNAME); |  
  _   END; |  
  _ CALL @CLOSEDSN; | 3  
  _ CALL @OPENDSN(TSOID || ' .EXAMPLES.DATA'); | 4  
  _ RECORD = @READDSN; | 5  
  _ CALL MSGLOG(RECORD); | 6  
  _ CALL @CLOSEDSN; | 7  
  _ -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following output:

```
----- INFORMATIONAL MESSAGE LOG  
-----  
COMMAND ==> SCROLL ==> P  
SMYTHE
```

OPSTATS

Returns statistical data collected by the Data Object Broker. (F)

Invocation `string = OPSTATS(request-value)`

`string` On return contains the statistical data.

`request-value` Defines the statistical data to be returned.

Usage Notes Only those request-numbers documented below are supported for end-user rule execution. The request-numbers are platform specific. A set of sample rules are supplied in library @SAMPLES. These rules retrieve the data from the Data Object Broker and use MAP and TEM table functionality to display the values using the Table Browser.

Request Value	z/OS	Win & UNIX	Function	Sample Rule Name	Associated Tables
8	Y	Y	Supplies cumulative statistics that form the basis of the SMF type 10 Data Object Broker record.	S6BSTAT08	S6BSTAT08_MAP S6BSTAT08_TEM
12	Y	Y	Provides segment status.	S6BSTAT12	S6BSTAT12_MAP S6BSTAT12_TEM S6BSTAT12NUM_MAP
24	Y	Y	Provides a list of active users.	S6BSTAT24	S6BSTAT24_MAP S6BSTAT24_TEM S6BSTAT24NUM_MAP
32	Y	N	Provides internal Data Object Broker buffer usage statistics.	S6BSTAT32	S6BSTAT32_MAP S6BSTAT32_TEM
36	Y	N	Provides Data Object Broker lock statistics.	S6BSTAT36	S6BSTAT36_MAP S6BSTAT36_TEM
64	Y	N	Provides Data Object Broker statistics for the current user.	S6BSTAT64	S6BSTAT64_MAP S6BSTAT64_TEM

OPTIONLISTER

Displays options in columns and returns the ones selected (C).

Invocation `CALL OPTIONLISTER(optionlistname)`

<i>optionlistname</i>	The name of the option as defined in the table @OPTIONS.
-----------------------	--

- Usage Notes**
- After calling OPTIONLISTER a screen appears that can be used to select specified options.
 - The selected options are stored in the temporary table @SELECTED_OPTION and they must be retrieved from the table for any additional action. The table @SELECTED_OPTION has the following fields: NAME, DESC1, DESC2.
 - The options must be specified in the table @OPTIONS as described in the section below.

Fields of the @OPTIONS Table

NAME	The name of the option.
OPTION_COLS	The number of option columns to appear. It can display a maximum of 4 columns, depending on the length of the source field and the number of description columns included. If description columns are included, only one option column can be defined. See FIELD and DESCRIPTION_COLS below for more information.
TABLE	The name of the source table. It contains the field that holds the values for the option listed and the fields used to describe the option values. The table cannot be parameterized. If the primary source table is parameterized, you can create a subview table based on the required table instance.

FIELD	<p>The name of the field that holds the values for the option listed.</p> <p>The values displayed for the field are truncated if the length of field is greater than space allocated due to the number of OPTION_COLS and DESCRIPTION_COLS.</p> <p>The option values for the field are truncated as follows:</p> <table><tr><td>Number of DESCRIPTION_COLS</td><td>Number of OPTION_COLS</td><td>Length of Option field value</td></tr><tr><td>0</td><td>1</td><td>76 characters</td></tr><tr><td>0</td><td>2</td><td>35 characters</td></tr><tr><td>0</td><td>3</td><td>22 characters</td></tr><tr><td>0</td><td>4</td><td>16 characters</td></tr><tr><td>1 or 2</td><td>1</td><td>16 characters</td></tr></table>			Number of DESCRIPTION_COLS	Number of OPTION_COLS	Length of Option field value	0	1	76 characters	0	2	35 characters	0	3	22 characters	0	4	16 characters	1 or 2	1	16 characters
Number of DESCRIPTION_COLS	Number of OPTION_COLS	Length of Option field value																			
0	1	76 characters																			
0	2	35 characters																			
0	3	22 characters																			
0	4	16 characters																			
1 or 2	1	16 characters																			
MAX_SELECT	<p>The maximum number of selections the user can make from the option list when it appears.</p> <p>Valid entries are a positive integer or asterisk (*). If asterisk (*) is entered, the users are not limited in their number of selections.</p>																				
TITLE	<p>The title to appear on the option lister screen.</p> <p>Up to 36 characters can be entered.</p>																				
DESCRIPTION_COLS	<p>The number of columns used to provide additional information about the option listed.</p> <p>There can be 0, 1 or 2 columns. If there is 1 column, a maximum of 58 characters of information can be entered. If there are 2 columns, the first column can hold up to 16 characters of information and the second column can hold up to 40 characters of information.</p>																				
DESCRIBE_FIELD1	<p>The field from the source table that appears for the first description column.</p>																				
DESCRIBE_FIELD2	<p>The field from the source table that appears for the second description column.</p>																				



@OPTIONS is a shared table, and therefore the data in it cannot be promoted.

Example The option SEL_DEPARTMENT is defined as follows:

```

        --- SINGLE OCCURRENCE EDITOR ---

EDITING TABLE      :  @OPTIONS
TABLE TYPE          :  TDS
COMMAND ==>
-----

NAME                :  SEL_DEPARTMENT
OPTION_COLS         :  1
TABLE               :  DEPARTMENT_SUB
FIELD               :  DEPTNO
MAX_SELECT          :  1
TITLE               :  LIST OF VALID DEPARTMENT NUMBERS
DESCRIPTION_COLS    :  1
DESCRIBE_FIELD1     :  DNAME
DESCRIBE_FIELD2     :
```

SELECT_OPTION Rule

The example SELECT_OPTION rule:

- 1. Clears the screen using DELETE_DATA
- 2. Calls OPTIONLISTER(SEL_DEPARTMENT) to display a listing of department numbers for selection
- 3. Retrieves the selected option from the @SELECTED_OPTION table and inserts it into the screen field **DEPTNO** of the screen table **EMPLOYEE_INFO**

```

SELECT_OPTION;
_ LOCAL MSG;
-----
_ CALL DELETE_DATA('EMPLOYEE_INFO(NEW_EMPLOYEE)', ' ', '');      | 1
_ CALL OPTIONLISTER('SEL_DEPARTMENT');                             | 2
_ FORALL @SELECTED_OPTION:                                         | 3
_     EMPLOYEE_INFO.DEPTNO = @SELECTED_OPTION.NAME;               |
_     INSERT EMPLOYEE_INFO('NEW_EMPLOYEE');                       |
_     END;                                                         |
-----
```

Selection Screen

The following screen appears for selection:

```

                                LIST OF VALID DEPARTMENT NUMBERS

COMMAND ==>                                SCROLL:  P
```


	DEPTNO	DNAME
—	10	ACCOUNTING
—	20	SALES
—	30	PRE-SALES
—	40	PRODUCT SUPPORT
—	50	RESEARCH
—	60	OPERATIONS
—	70	PUBLICATIONS

< Place "S" beside the option(s) you wish to have Selected on PF3 >
PFKEYS: 1=HELP 3=SELECT 13=PRINT 12=EXIT

\$OTMA

Invoke IMS OTMA Callable Interface calls. (C)

Invocation `CALL $OTMA(map_table)`

<code>map_table</code>	A pointer to the system map table @OTMA_MAP, which contains parameters for the different OTMA Callable Interface calls. This routine sets up the required input and output parameters for the respective OTMA interface calls.
------------------------	--

Exceptions

<code>VALIDATEFAIL</code>	One of the following: <ul style="list-style-type: none">- Invalid function request- OTMA not properly installed
---------------------------	--

<code>ROUTINEFAIL</code>	One of the following: <ul style="list-style-type: none">- OTMA open failed- OTMA send failed
--------------------------	---

Samples Sample rules and tables are available that you can use as a template:
The rule names start with @OTMA and are in the COMMON library.
The table names start with @OTMA, under UNIT=OTMA.

@OTMA_MAP

Register and allocate storage for use with the @OTMA_MAP table. (TBL)

Table Definition @OTMA_MAP has the following fields:

Table: @OTMA_MAP			Type: MAP			Unit: OTMA			IDgen: Y			
Parameter Name		Typ	Syn	Len	Dc	Cls	Reference		Event Rule		Typ	Acc
ADDRESS		B		4	0	A						
Field Name		Xsyn	Xlen	Xdec	Offset	Key	Typ	Syn	Len	Dec	Rqd	Default
KEY	B		4	0	0	P	I	B	4	0		
FUNCTION	C		4	0	0			C	4	0		
ANCHOR	B		8	0	4			B	8	0		
RETURNCODE	B		4	0	12			B	4	0		
REASON1	B		4	0	16			B	4	0		
REASON2	B		4	0	20			B	4	0		
REASON3	B		4	0	24			B	4	0		
REASON4	B		4	0	28			B	4	0		
GROUP_NAME	C		8	0	32			C	8	0		
MEMBER_NAME	C		16	0	40			C	16	0		
PARTNER_NAME	C		16	0	56			C	16	0		
SESSIONS	B		4	0	72			B	4	0		
TPIPE_PREFIX	C		4	0	76			C	4	0		
SESSION_HANDLE	B		8	0	80			B	8	0		
PROC_OPT	B		1	0	88			B	2	0		
DUMMY_FILLER	B		3	0	89			B	2	0		
TRANSACTION	C		8	0	92			C	8	0		
PRF_NAME	C		8	0	100			C	8	0		
LTERM	C		8	0	108			C	8	0		
MODNAME	C		8	0	116			C	8	0		
SEND_BUFFER@	B		4	0	124			B	4	0		
SEND_BUFFER_LEN	B		4	0	128			B	4	0		
SEND_SEG_LIST	B		4	0	132			B	4	0		
RECEIVE_BUFFER@	B		4	0	136			B	4	0		
RECV_BUFFER_LEN	B		4	0	140			B	4	0		
RECEIVED_LEN	B		4	0	144			B	4	0		
RECV_SEG_LIST	B		4	0	148			B	4	0		
CONTEXTID_PART1	B		8	0	152			B	8	0		
CONTEXTID_PART2	B		8	0	160			B	8	0		
ERROR_MESSAGE	V		120	0	168			V	120	0		

Fields

FUNCTION

The function code: OPEN, ALLO, SEND, FREE, or CLOS.

ANCHOR	A field initialized to zero before the OPEN call and filled in by the otma_open function call.
RETURNCODE, REASON1, REASON2, REASON3, REASON4	Status information for the transaction.
GROUP_NAME	The XCF group name.
MEMBER_NAME	The member name for this client.
PARTNER_NAME	The member name for the OTMA server (IMS).
SESSIONS	The number of parallel sessions to be supported with IMS. The only valid value is 1.
TPIPE_PREFIX	The TPIPE name prefix.
SESSION_HANDLE	The session identifier for the subsequent otma_send_receive. It is filled in by the otma_allocate function call.
PROC_OPT	<p>The processing options for the subsequent otma_send_receive call. The supported options are (can be NULL, to use the defaults):</p> <p>Bit 0 (SyncOnReturn): This option tells IMS to process the message without the RRS context token; in this case, the user ID is obtained when RRS CTXRDTA is invoked.</p> <p>Bit 1 (SyncLevel1): With this option, the OTMA send-then-commit (Commit Mode 1) SYNCLEVEL 1 is used instead of the default SYNCLEVEL 0.</p>
TRANSACTION	The name of the IMS transaction or command to be sent to IMS.
PRF_NAME	The name of the RACF group name for transactions and commands.
LTERM	The lterm name. On input it is passed to IMS. It is updated on output to the lterm field returned by IMS. Can be blank in both cases.

MODNAME	The modname. On input, it is passed to IMS. It is updated on output to the modname field returned by IMS. Can be blank in both cases. If the input modname is DFMSM01, DFMSM02, or DFMSM05, it is treated as blanks.
SEND_BUFFER@	The pointer to the data to be sent to IMS. When a NULL is specified for TRANSACTION, the transaction name or command must appear, followed by a blank, followed by the data, in this buffer when it is being sent to IMS.
SEND_BUFFER_LEN	The length of the send data.
SEND_SEG_LIST	The pointer to an array of lengths of message segments to be sent to IMS. The first element is the count of the following segment lengths. If a single segment is to be sent, either the first element or the address of the array can be zero.
RECEIVE_BUFFER@	The pointer to the buffer to receive the reply message from IMS.
RECV_BUFFER_LEN	The length of the buffer available to receive the message.
RECEIVED_LEN	The length of data received in the receive buffer. It should equal the sum of the segment lengths.
RECV_SEG_LIST	<p>The pointer to an array to hold the number of segments sent by IMS.</p> <p>On send, the first element must be set to the number of elements in the array. If a single segment is to be received, either the first element or the address of the array can be zero, in which case all segments are received contiguously without indication of segmentation boundaries.</p> <p>On receive, the first element is the count of the following segment lengths and must be set by the client to indicate the maximum length of the array. It is modified by the receive.</p>

CONTEXTID	The Distributed Sync Point Context ID from RRS or NULL. For an unauthorized caller (as is the case with the TIBCO Object Service Broker implementation), OTMA invokes the CTXSWCH call to disassociate the token and to validate if the token is current for a task. When OTMA receives a response from IMS, it switches the context back onto the task before returning control to the caller.
ERROR_MESSAGE	An error or information message from IMS.

Security Access to @OTMA_MAP is controlled using normal TIBCO Object Service Broker security manager tools. Refer to the *Security* manual.

Usage Notes The appropriate fields of the @OTMA_MAP map table must be set up before requesting an OTMA function. The following lists the required fields for each of the functions. After setting up the fields, you invoke the \$OTMA tool:

OTMA_OPEN

OTMA_OPEN invokes the otma_open interface call to connect with IMS. The required fields in the @OTMA MAP table are:

FUNCTION	OPEN
GROUP_NAME	The XCF group name.
MEMBER_NAME	The member name for this client.
PARTNER_NAME	The member name for the IMS OTMA server.
SESSIONS	The number of parallel sessions to be supported with IMS. The only valid value is 1.
TPIPE_PREFIX	The prefix for the TPIPE name.

Return values are in the RETURNCODE and REASONx fields:

0	XCF JOIN was successful, client-bid was sent, and acknowledgment received.
4	IMS is not ready. Try again later.

8	Your XCF group and member are already active.
12	A system error occurred.



For the complete description of each error, see the *IBM IMS Open Transaction Manager Access Guide and Reference* manual.

OTMA_ALLOC

OTMA_ALLOC invokes the `otma_allocate` interface call to create an independent session to exchange messages. The required fields in the @OTMA_MAP table are:

FUNCTION	ALLO
TRANSACTION	A name for the IMS transaction or command to be sent to IMS.
PROC_OPT	The SyncOnReturn and SyncLevel1 processing options you want for the subsequent <code>otma_send_receive</code> call, in bits 0 and 1. You select the option by turning on the bit.
PRF_NAME	The name of the RACF group name for OTMA transactions and commands.

Return values are in the RETURNCODE and REASONx fields:

0	Success.
4	Session limit reached.
8	Null anchor.

OTMA_SEND_RECEIVE

OTMA_SEND_RECEIVE invokes the `otma_send_receive` interface call to initiate a message exchange with IMS. The required fields in the @OTMA_MAP table are:

FUNCTION	SEND
LTERM	The lterm name.
MODNAME	The modname.

SEND_BUFFER@	A pointer to the buffer containing the IMS transaction or command data to be sent to IMS.
SEND_BUFFER_LEN	The size of the send buffer desired.
SEND_SEG_LIST	A pointer to an array of lengths of message segments to be sent to IMS.
RECEIVE_BUFFER@	A pointer to the buffer to receive the reply message from IMS.
RECV_BUFFER_LEN	The size of the receive buffer desired.
RECV_SEG_LIST	The pointer to an array to hold the number of segments sent by IMS.
CONTEXT_ID	A field containing NULL or the Distributed Sync Point Context ID from RRS.

Return values are in the RETURNCODE and REASON fields:

0	Normal completion.
8	No anchor, bad session handle, or segment too large.
12	Send failed.
16	Receive was cancelled.
20	Error from IMS.

OTMA_FREE

OTMA_FREE invokes the interface call otma_free to free an independent session created by otma_allocate. The required fields are already set in the @OTMA_MAP table.

Return values are in the RETURNCODE and REASON fields:

0	Success.
4	No allocated session.
8	Incorrect anchor.

OTMA_CLOSE

OTMA_CLOSE invokes the interface call otma_close to free storage for communication and leave the XCF group. The required fields are already set in the @OTMA_MAP table.

Return values are in the RETURNCODE and REASON fields:

0	Success.
4	Null anchor.
8	Cannot leave XCF group.

PAD

Returns a string padded to a specified length using a pad character, positioning the string to the left, right or center of the padding. (F)

Invocation `pad_string = PAD(string, length, padcharacter, just)`

<i>pad_string</i>	On return, contains the padded string. Its syntax can be UN (Unicode), V (for a <i>string</i> of C or V), or W (double-byte character).
<i>string</i>	The string to pad. Its syntax can be C (fixed-length character string), UN, V (variable-length character string), or W.
<i>length</i>	<p>An integer specifying the length, in characters, to which <i>pad_string</i> should be padded. Its syntax is B (binary) with length 2.</p> <p>The specified <i>length</i> cannot exceed the greater of 4096 or the value of the EXECSTACKSIZE session parameter minus 32768, to a maximum of 32,767.</p>
<i>padcharacter</i>	<p>The character with which to pad.</p> <p>Its syntax is C or UN. If it's of syntax C and alphabetic, it is treated as uppercase. If it's of syntax C and null, and <i>string</i> is of syntax C, V, or W, it is treated as a blank. If <i>string</i> is of syntax UN, a null pad character is not allowed.</p>
<i>just</i>	<p>One of:</p> <p>LEFT or L – Left justify the string.</p> <p>CENTER, CENTRE, or C – Center the string.</p> <p>RIGHT or R – Right justify the string.</p> <p>Its syntax is C with a length of 8.</p>

Usage Notes *string* is not altered.

Example Rule Using PAD

The following rule pads a string and prints both the original string and the padded string to the message log:

```

PAD_1;
_  LOCAL SOURCE_STRING, PADDED_STRING;
_  -----
_  -----+-----
_  SOURCE_STRING = 'THIS IS THE SOURCE STRING';           | 1
_  PADDED_STRING = PAD(SOURCE_STRING, 40, '*', 'C');       | 2
_  CALL MSGLOG('THE PADDED STRING IS:');                   | 3
_  CALL MSGLOG(PADEDDED_STRING);                           | 4
_  CALL MSGLOG('THE SOURCE STRING IS STILL:');             | 5
_  CALL MSGLOG(SOURCE_STRING);                             | 6
_  -----

```

Output for the PAD_1 Rule

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE PADDED STRING IS:
*****THIS IS THE SOURCE STRING*****
THE SOURCE STRING IS STILL:
THIS IS THE SOURCE STRING

```

PARMVALUE

Returns the value of the parameter from the table that was accessed when the trigger or validation rule was activated. (F)

Invocation `value = PARMVALUE(parmname)`

<i>value</i>	On return, contains the value of the parameter. Its syntax can be either C (fixed-length character string) or V (variable-length character string).
<i>parmname</i>	A string specifying the name of the parameter. Its syntax is C with length 16.

- Usage Notes**
- PARMVALUE is required because parameters are not stored as field occurrences.
 - PARMVALUE can be used only within an event rule.

See Also *TIBCO Object Service Broker Managing Data* for information about event rules.

Exceptions

ERROR	Raised if an event-driven rule is not being run or the parameter named is not defined as a parameter for the triggering table.
--------------	--

Example The following validation rule checks the updated values for the field **EMPNO** in the **EMPLOYEE_DEPT** parameterized table and returns the parameter value if the update fails:

```

PARMVALUE_1;
_ LOCAL VALUE;
-----
_ PARMVALUE ('DEPTNO') != 40;                                | Y N N
_ EMPLOYEE_DEPT.EMPNO <= 90000;                               |   Y N
-----+-----
_ VALUE = PARMVALUE ('DEPTNO');                               | 1 1 1
_ RETURN ('FOR DEPTNO ' || VALUE || ' EMPNO MUST            |      2
_ BE LESS THAN 90000');                                       |
_ RETURN ('Y');                                               | 2 2
-----

```

Output for the PARMVALUE_1 Rule

Pressing Enter after updating the table with an invalid value returns the following message to the screen:

```
EDITING TABLE      :   EMPLOYEE_DEPT(40)
COMMAND ==>
```

	EMPNO	LNAME	POSITION	MGR#	DEPTNO	SALARY
—	80033	CANON	Programmer	90020	40	567.09
—	81001	CAREY	Secretary	81092	40	565.89
—	81003	CHIU	VP	81033	40	865.70
—	91014	LYNGBAEK	Mgr	84021	40	780.67

```
PFKEYS: 1=HELP 3=END 12=CANCEL 13=PRINT
```

```
UPDATE FAILED:  FOR DEPTNO 40 EMPNO MUST BE LESS THAN 90000
```

PARSE

Breaks up an input string into tokens and applies grammar rules to the tokens. (C)

Invocation `CALL PARSE(grammar_usage, string)`

<i>grammar_usage</i>	<p>The <i>grammar</i> and <i>usage</i> parameters, for the SEMANTIC table, to be used for this invocation of PARSE. Type one of the following:</p> <ul style="list-style-type: none"> • If the <i>usage</i> parameter for SEMANTIC is STANDARD, type the value of the <i>grammar</i> parameter. • If you are using an instance of the SEMANTIC table where the <i>usage</i> parameter is other than STANDARD, type the value of the <i>grammar</i> parameter, followed by a space, followed by the value of the <i>usage</i> parameter. <p>For more information on the <i>usage</i> parameter, refer to Task B, Use the SEMANTIC table to associate actions with changes of state, on page 435.</p>
----------------------	---

<i>string</i>	The string of characters to be analyzed by the parser.
---------------	--

Overview

PARSE is a finite-state machine written in rules. It breaks up the input string into tokens (distinct elements) and applies grammar rules to each of the tokens. If the grammar rules for each token are met, the string's syntax is correct and each token can be passed to a specified rule or rules for further processing. If any of the tokens do not satisfy the grammar rules, the exception SYNTAX_ERROR is raised and PARSE fails.

This section provides an overview of the tasks you must complete to use PARSE:

- [Task A, Use the GRAMMARS table to specify tokens and states](#), page 433

Use the GRAMMARS table to do the following:

- a. Specify the type and number of tokens into which you want the input string to be broken.

For example, if the input string is a customer's name, you could specify three tokens of type ID to accommodate the first, middle, and last names. When PARSE analyses the input string, the string must satisfy these

grammar rules (that is, it must consist of three elements that PARSE recognizes as type ID) or PARSE fails.

- b. Specify the successive states into which PARSE changes as it processes each token.

PARSE checks that the tokens in the input string are of the correct type and in the correct order. It also keeps track of the tokens so it can apply the appropriate rule to each token that it parses successfully. To enable PARSE to do this, you specify a series of successive states through which PARSE passes as it successfully parses each token.

For example, you could specify that when PARSE processes the first name of the customer, its state changes from START to MNAME; when it processes the middle name, from MNAME to LNAME; and when it processes the last name, from LNAME to FINISH. When PARSE changes from one state to another, it can execute a rule associated with the specific change of state. In this way, rules can be applied to each token based on its position in the input string.

- [Task B, Use the SEMANTIC table to associate actions with changes of state, page 435](#)

Actions (rules) can be associated with each change of state. These rules are used to further process each token.

Refer to the following sections to learn how to complete these steps.

Task A Use the GRAMMARS table to specify tokens and states

Use of the GRAMMARS Table

Use the GRAMMARS table to specify the following:

- The number and types of tokens
- Transition states for each token

GRAMMARS is parameterized by GRAMMAR, a unique name you assign to the set of grammar rules you construct for PARSE.

Fields of the GRAMMARS Table

For each token, in order from first to last in the input string, enter values for the following fields:

Field	Description	Rules
INDEX	A number that uniquely identifies each token and its associated states.	Enter a unique number for each token.
STATE	The current (initial) state, from which the parser changes if it finds the specified token.	The first state for a series of tokens must be START, which PARSE begins from. The STATE can be up to 16 characters long.
TOKEN	<p>The token or type of token for which PARSE checks while it is in the specified STATE:</p> <p>If the parser finds the specified token or type of token while it is in STATE, it changes to NEW_STATE and can execute an assigned action on the token.</p> <p>If the parser does not find the specified token or type of token when it is in STATE, a SYNTAX_ERROR exception is raised and PARSE fails.</p>	<p>Specify a string up to 17 characters in length, or enter one of the following values to make PARSE try to match a particular type of token:</p> <p><i>%grammar</i> - A nested grammar</p> <p>A token that begins with a percent (%) sign represents a nested grammar. The parser tries to match the input with the tokens in the nested grammar before continuing in the current grammar.</p> <p>ANY - Matches any token</p> <p>ID - Matches a character or series of characters, excluding numbers, symbols or special characters</p> <p>NUM - Matches a number or series of numbers</p> <p>RLIT - Matches any raw-data literal</p> <p>STR - Matches any quoted string</p> <p>ULIT - Matches any Unicode literal</p> <p>XLIT - Matches any hexadecimal literal</p> <p>No token (empty field) - Matches null input</p> <p>PARSE always changes from STATE to NEW_STATE when this token type is specified.</p>

Field	Description	Rules
NEW_STATE	The new (subsequent) state to which the parser changes if it finds the TOKEN while in the specified STATE.	The final NEW_STATE for a series of tokens must be ACCEPT. NEW_STATE can be up to 16 characters long.

Example of the GRAMMARS(CUST_NAME) Table

The following example illustrates how the GRAMMARS table could be set up to parse a string consisting of a first, middle, and last name.

BROWSING TABLE : GRAMMARS(CUST_NAME)
COMMAND ==>

INDEX	STATE	TOKEN	NEW_STATE
1	START	ID	MNAME
2	MNAME	ID	LNAME
3	LNAME	ID	FINISH
4	LNAME		ACCEPT
5	FINISH		ACCEPT

Task B Use the SEMANTIC table to associate actions with changes of state

Use the SEMANTIC table to associate actions (rules) with changes of state identified in the GRAMMARS table. The action is invoked before PARSE makes the transition from STATE to NEW_STATE. You do not have to associate every change of state with an action.

SEMANTIC is parameterized by *grammar* and *usage*, which are described here:

grammar Parameter

The *grammar* parameter for the SEMANTIC table should match the *grammar* parameter you specified for the GRAMMARS table.

usage Parameter

Using the *usage* parameter, you can associate more than one set of rules with a given grammar. You could want a particular change of state identified in the GRAMMARS table to result in PARSE executing one rule in some instances and another in other instances. You accomplish this by specifying different usage values for the *grammar_usage* argument when you call PARSE.

Default Value for the *usage* Parameter

The default value for *usage* is STANDARD. If PARSE is called with only one value for *grammar_usage*, it assumes that the value is the name of the grammar and that the *usage* is STANDARD. If you have only one set of rules for a particular grammar, it is simplest to use STANDARD as a value for the *usage* parameter.

Specifying Multiple Usages

If you want to specify different sets of rules for the grammar, you can use other values for the *usage* parameter and then include them in the *grammar_usage* argument when you call PARSE. The following is an example of calling PARSE with a particular usage parameter:

```
CALL PARSE('GRAMMAR1 USAGE1', 'INPUT_STRING')
```

In this example, the input string is parsed with the grammar described in GRAMMARS(GRAMMAR1) and the rules in SEMANTIC(GRAMMAR1, USAGE1) are applied to the tokens.

Fields of the SEMANTIC Table

Supply values for the following fields:

Field	Description	Rules
INDEX	A unique number that identifies the action and associates it with a particular change of state in the GRAMMARS table.	Associate an action with a change of state by having them share the same INDEX number. You only have to create entries for those changes of state that require actions.
ACTION	The name of a rule that is executed before the transition from STATE to NEW_STATE.	Actions must be procedural rules with no arguments. Actions have access to two variables: INPUT_TOKEN, which contains the text of the current token, and MSG, which is used to pass a description of a semantic failure, if one arises. Actions can raise any exceptions that are necessary; PARSE's caller is responsible for handling any exceptions. This field can be null.

Example of
SEMANTIC
(CUST_NAME,
STANDARD)
Table

The following example shows how the changes of state in the GRAMMARS(CUST_NAME) table can be associated with particular actions.

BROWSING TABLE : SEMANTIC(CUST_NAME,STANDARD)		SCROLL: P
COMMAND ==>		
INDEX	ACTION	
-----	-----	
1	SAVE_FIRSTNAME	
2	SET_LASTNAME	
3	RESET_LASTNAME	

Usage Notes

You must declare the local variable MSG. It is used to pass a description of a semantic failure, should one arise.

Exceptions

SYNTAX_ERROR	Signaled if <i>string</i> does not follow the rules described by <i>grammar_usage</i> .
--------------	---

Examples

Parsing a Customer Name

The following example parses a customer name, breaking it into three tokens (first, middle, and last name), and printing the tokens to the message log. The example is composed of the following elements:

- The table GRAMMARS(CUST_NAME), shown in [Task A, Use the GRAMMARS table to specify tokens and states, on page 433](#)
- The table SEMANTIC(CUST_NAME, STANDARD), shown in [Task B, Use the SEMANTIC table to associate actions with changes of state, on page 435](#)
- The rules SAVE_FIRSTNAME, SET_LASTNAME, and RESET_LASTNAME, which are listed in the field ACTION in the table SEMANTIC(CUST_NAME, STANDARD)
- The TEST_CUSTNAME parent rule

Actions

The following are the rules that constitute the ACTIONS in the SEMANTIC(CUST_NAME,STANDARD) table:

RULE EDITOR ==>>	SCROLL: P
------------------	-----------

```
SAVE_FIRSTNAME;
-
- -----+-----
- FNAME = INPUT_TOKEN; | 1
- -----
```

```
    RULE EDITOR ==>                                SCROLL: P
SET_LASTNAME;
-
- -----+-----
- LNAME = INPUT_TOKEN; | 1
- -----
```

```
    RULE EDITOR ==>                                SCROLL: P
RESET_LASTNAME;
-
- -----+-----
- MNAME = LNAME; | 1
- LNAME = INPUT_TOKEN; | 2
- -----
```

The TEST_CUSTNAME Parent Rule

The TEST_CUSTNAME rule parses customer names using the grammar CUST_NAME:

```
    RULE EDITOR ==>                                SCROLL: P
TEST_CUSTNAME(NAME);
- LOCAL MSG, FNAME, MNAME, LNAME;
- -----
- CALL PARSE('CUST_NAME', NAME); | 1
- CALL MSGLOG('THE FIRST NAME IS ' || FNAME); | 2
- CALL MSGLOG('THE MIDDLE NAME IS ' || MNAME); | 3
- CALL MSGLOG('THE LAST NAME IS ' || LNAME); | 4
- -----
```

Result

When the TEST_CUSTNAME rule executes with the argument 'Margaret Alison Smith', the following message log is produced:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>
THE FIRST NAME IS MARGARET
THE MIDDLE NAME IS ALISON
THE LAST NAME IS SMITH
SCROLL ==> P

```

Example 2: Parsing an Address

The following example parses an address of the form:

ROBERT JONES, 31 HIGH ROAD, BUFFALO, NY

It breaks the address into tokens, removes the commas, and recombines the tokens and prints them to the message log with titles. The tokens could also be passed to other rules for further processing or storage in tables. The example consists of the following parts:

- Table GRAMMARS(ADDRESS)
- Table SEMANTIC(ADDRESS, STANDARD)
- Action rules ADDRESS1 to ADDRESS7, which are listed as actions in the table SEMANTIC(ADDRESS, STANDARD)
- The PARSE_ADDRESS parent rule

Table GRAMMARS(ADDRESS)

EDITING TABLE : GRAMMARS(ADDRESS)

COMMAND ==>

SCROLL: P

INDEX	STATE	TOKEN	NEW_STATE
1	START	ID	FNAME
2	FNAME	ID	LNAME
3	LNAME	,	COMMA1
4	COMMA1	NUM	NUMBER
5	NUMBER	ID	STREET
6	STREET	ID	STR_OR_RD
7	STR_OR_RD	,	COMMA2
8	COMMA2	ID	CITY
9	CITY	,	COMMA3
10	COMMA3	ID	STATE
11	STATE		FINISH
12	FINISH		ACCEPT

Table SEMANTIC(ADDRESS, STANDARD)

EDITING TABLE : SEMANTIC(ADDRESS, STANDARD)

COMMAND ==>		SCROLL: P
INDEX	ACTION	
-----	-----	
1	SET_FNAME	
2	SET_LNAME	
4	SET_NUMBER	
5	SET_STREET	
6	SET_STR_OR_RD	
8	SET_CITY	
10	SET_STATE	

Actions ADDRESS1 to ADDRESS 7

RULE EDITOR ==>		SCROLL: P
SET_FNAME;		
-----	-----	
FNAME = INPUT_TOKEN;	1	

RULE EDITOR ==>		SCROLL: P
SET_LNAME;		
-----	-----	
LNAME = INPUT_TOKEN;	1	

RULE EDITOR ==>		SCROLL: P
SET_NUMBER;		
-----	-----	
NUMBER = INPUT_TOKEN;	1	

RULE EDITOR ==>		SCROLL: P
SET_STREET;		
-----	-----	
STREET = INPUT_TOKEN;	1	

RULE EDITOR ==>		SCROLL: P
SET_STR_OR_RD;		

STR_OR_RD = INPUT_TOKEN;		1

RULE EDITOR ==>		SCROLL: P
SET_CITY;		

CITY = INPUT_TOKEN;		1

RULE EDITOR ==>		SCROLL: P
SET_STATE;		

STATE = INPUT_TOKEN;		1

The PARSE_ADDRESS Parent Rule

RULE EDITOR ==>		SCROLL: P
PARSE_ADDRESS(String);		
_ LOCAL MSG, FNAME, LNAME, NUMBER, STREET, STR_OR_RD, CITY, STATE;		

CALL PARSE('ADDRESS', String);		1
CALL MSGLOG('FIRST NAME: ' FNAME);		2
CALL MSGLOG('LAST NAME: ' LNAME);		3
CALL MSGLOG('ADDRESS: ' NUMBER ' ' STREET ' ' STR_OR_RD);		4
CALL MSGLOG('CITY: ' CITY);		5
CALL MSGLOG('STATE: ' STATE);		6

ON SYNTAX_ERROR :		
_ CALL MSGLOG('SYNTAX INCORRECT: ' String);		

Result

If PARSE_ADDRESS is executed with the argument

ROBERT JONES, 31 HIGH ROAD, BUFFALO, NY

the message log produced is:

```
----- INFORMATION LOG -----  
COMMAND ==>                                SCROLL ==> P  
FIRST NAME: ROBERT  
LAST NAME: JONES  
ADDRESS: 31 HIGH ROAD  
CITY: BUFFALO  
STATE: NY
```

PARSE_TAM

Breaks up an input string into a table specification, and optionally, the WHERE clause and the ORDERED clause of the corresponding table access statement. (C)

Invocation `CALL PARSE_TAM(string)`

<i>string</i>	The string of characters to be analyzed by the parser. Contains the table specification, optionally the WHERE clause, and optionally the ORDERED clause of the corresponding table access statement.
---------------	--

Usage Notes On successful return, the fields of the table buffer of the TEM table TAM are initialized.

Fields of the TEM Table Called TAM

Field	Description
TAM.TNAME	The table name of the table specification.
TAM.#PARMS	The number of parameters in the table specification.
TAM.PSTR	An encoding of the parameters suitable for passing to the FORALLA tool, or empty.
TAM.WSTR	An encoding of the WHERE clause suitable for passing to the FORALLA tool, or empty.
TAM.OSTR	An encoding of the ORDERED clause suitable for passing to the FORALLA tool, or empty.
TAM.MSG	An error message if <code>PARSER_ERROR</code> is signalled, or empty.
TAM.UNIQUE	Not used; always N.
TAM.TSPECS	The table specification.
TAM.P1	The first parameter of the table specification, or empty.
TAM.P2	The second parameter of the table specification, or empty.
TAM.P3	The third parameter of the table specification, or empty.

Field	Description
TAM.P4	The fourth parameter of the table specification, or empty.
TAM.P5	The fifth parameter of the table specification, or empty.
TAM.P6	The sixth parameter of the table specification, or empty.

Successive invocations of PARSE_TAM overwrite the values in the TAM table buffer left by previous invocations.

Exceptions

PARSER_ERROR	There is a syntax error in the table specification, WHERE clause and ORDERED clause passed in the argument <i>string</i> . The field TAM.MSG contains a message specifying the syntax error. The other fields of TAM are NULL.
--------------	--

Example Rule

<pre> RULE EDITOR ==> QUERY(String); LOCAL ROW; ----- CALL PARSE_TAM(String); CALL FORALLA(TAM.TNAME, TAM.PSTR, TAM.WSTR, TAM.OSTR); UNTIL TABLEEND : ROW = ROW + 1; CALL MSGLOG(TAM.TNAME ' ROW# ' ROW); CALL MSGLOG(''); FORALL FIELDS(TAM.TNAME) : CALL MSGLOG(PAD(FIELDS.NAME, 16, ' ', 'R') ': ' (TAM.TNAME).(FIELDS.NAME)); END; CALL MSGLOG(''); CALL FORALLB(TAM.TNAME); END; CALL FORALLE(TAM.TNAME); CALL ENDMSG(ROW ' ROWS FOUND IN ' TAM.TNAME); ----- ON PARSER_ERROR : CALL ENDMSG(TAM.MSG); ON TABLEEND : CALL FORALLE(TAM.TNAME); CALL ENDMSG('NO ROWS FOUND IN ' TAM.TNAME); </pre>	<pre> SCROLL: P 1 2 3 4 5 </pre>
--	----------------------------------

Explanation

The QUERY rule does the following:

1. Takes a string containing the table specification, plus optionally the WHERE clause and optionally the ORDERED clause of a table access statement
2. Dynamically accesses all rows of the specified table instance that match the WHERE clause (if there is a WHERE clause, otherwise all the rows)
3. Writes them to the message log in the specified order (if any, otherwise the default table order)

For example:

```
CALL QUERY('TABLES WHERE NAME >= ''Q'' ORDERED DESCENDING NAME');
```

lists the rows of the TABLES table where the field value of NAME is greater than or equal to the letter Q in reverse alphabetical order of the values of the field NAME.

```
CALL QUERY('FIELDS(TAM)');
```

lists the rows of the table instance with a parameter value of TAM of the FIELDS table.

PATTERN_MATCH

Determines whether a string matches a given pattern. (F)

Invocation `PATTERN_MATCH(string, pattern)`

<i>string</i>	The sequence of characters that is matched against the pattern.
<i>pattern</i>	The sequence of characters, which could contain the special characters asterisk (*) and question mark (?)

- Usage Notes**
- An asterisk (*) is used to match any sequence of zero or more characters.
 - A question mark (?) is used to match any single character.
 - PATTERN_MATCH returns either Y (in which case the string is a partial match of the pattern) or N.

Example The following rule illustrates the use of PATTERN_MATCH:

- The rule checks to see if the commands **SELECT** or **DELETE**, or any abbreviation of these commands, such as **SEL** or **DEL**, are valid.



Normally if you have several commands, you would store them in a table and use LIKE as your selection operator.

- If the command is a valid **SELECT** or **DELETE** command, the appropriate rule is called.
- If the command is not valid, the user gets a message on the screen (EMPSCREEN) explaining that the command is not acceptable. The following illustrates a rule using PATTERN_MATCH:

```
PROCESS_CMD(USER_COMMAND);
```

```

-----
- PATTERN_MATCH(USER_COMMAND, 'SEL*');           | Y  N  N
- PATTERN_MATCH(USER_COMMAND, 'DEL*');           |   Y  N
-----+-----
- CALL SELECT_COMMAND;                           | 1
- CALL DELETE_COMMAND;                           |   1
- CALL SCREENMSG('EMPSCREEN', USER_COMMAND ||    |   1
-   'IS AN INVALID COMMAND');
-----

```

PEEL

Returns the result of removing the specified leading and trailing characters from the specified string. (F)

Invocation `peeled_string = PEEL(peelchars, string)`

<i>peeled_string</i>	On return, contains the peeled string. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>peelchars</i>	The string of characters to be peeled. Its syntax can be C, V, or W.
<i>string</i>	The string from which to peel the characters. Its syntax can be either C, V, or W.

- Usage Notes**
- PEEL examines both the front and the back of *string*, and peels each character that matches one of the characters in *peelchars*. The peeling continues until a non-peel character is encountered.
 - PEEL does not alter the value of *string*.

Example The following rule peels characters from a string and prints both the original string and the peeled string to the message log:

```

PEEL_1;
_ LOCAL SOURCE_STRING, PEELED_STRING;
_ -----
_ SOURCE_STRING = '1231234561323213';
_ PEELED_STRING = PEEL('321', SOURCE_STRING);
_ CALL MSGLOG('THE PEELED STRING IS: ' || PEELED_STRING);
_ CALL MSGLOG('THE SOURCE STRING IS STILL: ' ||
_ SOURCE_STRING);
_ -----

```

Output for the PEEL_1 Rule

Pressing PF2 after executing this rule displays the following on the screen:

```

----- INFORMATION LOG -----
COMMAND ==>                                SCROLL ==> P
THE PEELED STRING IS: 456

```

THE SOURCE STRING IS STILL: 1231234561323213

PEEL_HEAD

Removes the specified leading characters from a given string. (F)

Invocation `endstring = PEEL_HEAD(char, string)`

<i>endstring</i>	The string returned after all occurrences of <i>char</i> are removed from the beginning of <i>string</i> .
<i>char</i>	A list of characters to be removed from the beginning of <i>string</i> .
<i>string</i>	The character string from which leading characters are removed.

Usage Notes *char* cannot contain the cent character.

Example This rule removes leading blanks from last names in the EMPLOYEE table:

```

RULE EDITOR ==>                                SCROLL: P
TRIM_NAME;
-
- -----
- -----
- FORALL EMPLOYEE :                               | 1
-     CALL MSGLOG(PEEL_HEAD(' ', EMPLOYEE.LNAME)); |
-     END;                                         |
- -----

```

Source Data

The last names in the table look like the following:

```

LNAME
-----
DRABEK
ROEDER
HOEGSON
GLADWELL
    URBANEK
        TERAMURA
    LEES

```

Resulting Data

After using the rule they appear in the message log like this:

DRABEK
ROEDER
HOEGSON
GLADWELL
URBANЕК
TERAMURA
LEES

PEEL_TAIL

Removes the specified trailing characters from a given string. (F)

Invocation `startstring = PEEL_TAIL(char, string)`

<code>startstring</code>	The string returned after all occurrences of <i>char</i> are removed from the end of <i>string</i> .
--------------------------	--

<i>char</i>	A list of characters to be removed from the end of <i>string</i> .
-------------	--

<i>string</i>	The character string from which trailing characters are removed.
---------------	--

Usage Notes *char* cannot contain the cent character.

Example The following rule removes periods and trailing blanks from the end of addresses in the EMPLOYEE table:

```

      RULE EDITOR ==>                                SCROLL: P
TRIM_ADDRESS;
-
- -----
- -----
- FORALL EMPLOYEE :                                | 1
-     CALL MSGLOG(PEEL_TAIL(' . ', EMPLOYEE.ADDRESS)); |
-     END;                                           |
- -----

```

Source Data

The addresses in the table look like the following:

```

ADDRESS
-----
23 Irvine Rd.
2076 Chappel Dr.
335 Princess St.
215 Rogers Ave.

```

Resultant Data

They appear in the message log like the following:

23 Irvine Rd
2076 Chappel Dr
335 Princess St
215 Rogers Ave

@PEERSERVERID

Directs remote TIBCO Object Service Broker table accesses to a particular peer server on a remote TIBCO Object Service Broker system. (TBL)

Table Definition These are the fields within the @PEERSERVERID table:

Field Name	Typ	Syn	Len	Dec	Key	Ord	Rq	Description
SERVER-LOCATION	I	C	16		P			Location of remote server.
SERVERID	I	C	8	0			Y	Name of remote server.

Usage Notes

- On every TIBCO Object Service Broker access to a remote table (that is, when the LOCATION parameter on the table does not match the name of the local TIBCO Object Service Broker system), TIBCO Object Service Broker examines the table @PEERSERVERID on the local system. If the **SERVERLOCATION** field in a row in @PEERSERVERID matches the name of the remote TIBCO Object Service Broker system, the server name in the **SERVERID** field in that occurrence is appended to the request before it is sent to the remote system. If no match is found for the name of the remote TIBCO Object Service Broker system, the request is sent with the server name DEFAULT0 appended.
- The @PEERSERVERID table is limited to an implementation-defined number of occurrences (currently 6). Attempting to insert more occurrences than supported raises the COMMITLIMIT exception; however, issuing a COMMIT statement does not permit more occurrences to be inserted into the table.
- Supported operations are INSERT, GET, FORALL, REPLACE, and DELETE.
- Selection and ordering are supported.
- Parameter value (PRM) tables on @PEERSERVERID are *not* supported.
- Subview (SUB) tables on @PEERSERVERID are *not* supported.
- Triggers on @PEERSERVERID are *not* supported.
- @PEERSERVERID behaves like a session table in that its content and effects are local to one TIBCO Object Service Broker session and are not seen by other users, even if they are sharing a single Execution Environment.

- It is not an error to have invalid or nonexistent system or server names in @PEERSERVERID.
- If a nonexistent or inactive server name is used, the error is discovered at the remote system, and a suitable message is returned from that system.
- Only one server name can be used at a given remote system within a single transaction.

Exceptions

COMMITLIMIT	Maximum number of commits reached.
-------------	------------------------------------

Example The following example illustrates the use of @PEERSERVERID:

```
PSIRULE_1;
--
-- -----
-- @PEERSERVERID.SERVERLOCATION = 'TORONTO';           | 1
-- @PEERSERVERID.SERVERID = 'SERVER1';                 | 2
-- INSERT @PEERSERVERID;                               | 3
-- @PEERSERVERID.SERVERLOCATION = 'DALLAS';              | 4
-- @PEERSERVERID.SERVERID = 'DALCICS1';                 | 5
-- INSERT @PEERSERVERID;                               | 6
-- GET EMPLOYEES('DALLAS') WHERE EMPNO = 12345;        | 7
-- GET MANAGERS('TORONTO') WHERE EMNO = EMPLOYEES.MANNO; | 8
-- -----
--
```

The PSIRULE_1 rule inserts two occurrences into @PEERSERVERID, with remote system names TORONTO and DALLAS. When the GET on table EMPLOYEES is executed (action 7), TIBCO Object Service Broker searches the @PEERSERVERID table and finds the occurrence matching remote system name DALLAS. Server name DALCICS1 is then appended to the request and the DALLAS system attempts to use this server to access the EMPLOYEES table.

When the GET on the MANAGERS table is executed (action 8), TIBCO Object Service Broker searches the @PEERSERVERID table and finds the occurrence matching the remote system name TORONTO. Server name SERVER1 is then appended to the request and the TORONTO system attempts to use this server to access the MANAGERS table.

\$PIC

Returns a number in a format specified by a mask. (F)

Invocation `string = $PIC(value, mask)`

<i>string</i>	On return, contains the masked string. Its syntax is V (variable length character) with a maximum length of 80.
<i>value</i>	Contains the number to be formatted. It cannot be F (float) syntax.
<i>mask</i>	Contains a string of digit placeholder characters, special characters, and message characters. It can be up to 78 characters long, but cannot contain more than 31 digit placeholders.

Elements of a Display Mask

A display mask can have the following elements, all of which are optional:

Element	Description
Basic string	A string containing placeholders that determine how the digits in value appear.
Unconditional left string	A string prepended to the basic string.
Conditional right string	A string that is appended to the basic string and appears only if <i>value</i> is negative.
Unconditional right string	A string appended to the basic or conditional right string.

In the following mask: US\$-NNN,NNNV.99CR%paid

US\$-	Is the unconditional left string.
NNN,NNNV.99	Is the basic string.
CR	Is the conditional right string.
%paid	Is the unconditional right string.

Using this mask, \$PIC returns values such as the following:

- `$PIC(1234, 'US$-NNN,NNNV.99CR%paid')` returns `US$ 1,234.00%paid`
- `$PIC(-12345.6, 'US$-NNN,NNNV.99CR%paid')` returns `US$-12,345.60CR%paid`

Within a particular mask, any of the four elements can be omitted. The following sections explain how to use each element.

Basic String

In the basic string, you use digit placeholders, message characters, and the decimal point placeholder to control how a given value appears. The basic string begins with the first placeholder and ends with the last placeholder.

Digit Placeholders

Digit placeholders determine how digits in the value are positioned in the masked string. \$PIC uses the following digit placeholders:

Placeholder	Behavior	Examples
9	Displays leading zeros.	<code>\$PIC(12, '999')</code> returns <code>012</code> <code>\$PIC(1234, '\$999,999')</code> returns <code>\$001,234</code>
*	Displays leading zeros as asterisks (*).	<code>\$PIC(12, '****')</code> returns <code>*12</code> <code>\$PIC(1234, '\$***,***')</code> returns <code>***1,234</code>
Z	Displays leading zeros as blanks.	<code>\$PIC(12, 'ZZZ')</code> returns <code>12</code> <code>\$PIC(1234, '\$ZZZ,ZZZ')</code> returns <code>\$ 1,234</code>
N	Displays leading zeros as nulls	<code>\$PIC(12, 'NNN')</code> returns <code>12</code> <code>\$PIC(1234, '\$NNN,NNN')</code> returns <code>\$1,234</code>

Message Characters

A display mask can optionally contain message characters that do not serve as placeholders and are output as ordinary characters. Characters other than the digit placeholders are always treated as message characters, with the following exceptions:

- The characters E and e are reserved characters and cannot be used as message characters between placeholders.

- The characters V and percent sign (%) have specific functions and cannot be used as message characters between placeholders (refer to the sections below for more information on their function).

The digit placeholder characters asterisk (*), Z, N, and 9 can, in the following instances, function as message characters:

Instance	Examples
They precede the first dollar sign (\$) in a mask.	<p>In the mask 'NZ\$AZZ9', the initial 'NZ' is an ordinary character string, and the 'ZZ9' after the \$ are digit placeholders.</p> <p>\$PIC(123, 'NZ\$AZZ9') returns NZ\$A123</p>
<p>An asterisk (*), Z, or N follows a different digit placeholder.</p> <p>Note 9 is always a digit placeholder even if it appears after another digit placeholder, so an asterisk (*), Z, or N following it is treated as a message character.</p>	<p>In the mask 'ABCZZNZ', all three Zs are placeholders, and the N is treated as an ordinary character.</p> <p>\$PIC(123, 'ABCZZNZ') returns ABC12N3</p> <p>Note Any characters after the last digit placeholder up to either the end of the string or a percent sign are conditional, and their appearance depends on the value of the sign holder in the unconditional left string. Therefore:</p> <p>\$PIC(123, 'ABCZZ9Z') returns ABC123, not ABC123Z.</p> <p>Refer to Conditional Right String on page 459.</p>
They follow a percent sign (%).	<p>In the mask 'Z99%9', the second 9 is the last digit placeholder, and the last 9 is an ordinary character.</p> <p>\$PIC(123, 'Z99%9') returns 123%9</p>

For more information on how the dollar sign (\$) and the percent sign (%) characters are used in other elements in the mask, refer to the sections on the unconditional left and right strings below.

Decimal Point Placeholder

- Use the character V to align decimal digits:

\$PIC(23.45, '\$NNNV.99') returns \$23.45

\$PIC(12, '\$NNNV,99') returns \$12,00

- If V is placed immediately after the last digit placeholder and *value* is positive, no decimal digits from *value* appear. If *value* is negative, only a decimal point appears:

```
$PIC(12.34, '9999V.') returns 0012
```

```
$PIC(-12.34, '9999V.') returns 0012.
```

- If *value* contains more digits before the decimal point than the display mask can accommodate, \$PIC fails, as in:

```
$PIC(123.4, 'NNV.NN')
```

```
$PIC(1234, 'ZZZV.99')
```

- If there are more decimal digits than the mask can accommodate, the decimal value is truncated:

```
$PIC(1.68, 'NV.N') returns 1.6
```

```
$PIC(1.234, 'NNV.NN') returns 1.23
```

- Conversely, if there are more placeholders for the decimal digits than there are decimal digits in *value*, the remaining placeholders are replaced with zeros, no matter what the regular replacement character for that placeholder are:

```
$PIC(3.4, '**V.***') returns *3.400
```

```
$PIC(1.23, 'NV.NNNN') returns 1.2300
```



Omitting the V can result in incorrect numerical values because the decimal digits are not aligned.

Basic String Omitted

If the basic string is omitted from a mask, a value of 0 or blank outputs the mask; any other value causes \$PIC to fail.

```
$PIC(0, 'US$-') returns US$
```

Unconditional Left String

Use the unconditional left string to prepend message characters to the basic string. The unconditional left string consists of all characters that precede the first placeholder, and has two parts, both of which are optional: a character string and a sign holder.

Character String

To include in the unconditional left string any characters that ordinarily serve as placeholders or have special functions, place them before the first placeholders in the mask. As well, any characters preceding the first dollar sign in a mask are treated as message characters, except for the percent sign (%) character, which cannot be used in the unconditional left string. Subsequent dollar signs (\$) in a mask are treated as message characters. For example:

- `$PIC(1234, '9V9$9,999V.99')` returns `9V9$1,234.00`
- `$PIC(1234, 'ZZ$Z$ZZZ')` returns `ZZ$1$234`

Sign Holder

Using the sign holder, you can display whether *value* is positive or negative. The first positive (+) or negative (-) sign before the first placeholder is the sign holder. Subsequent positive or negative signs are treated as message characters. The sign holder follows these rules:

- If the sign holder is positive (+) and the number is negative, the sign holder is changed to the negative (-).

`$PIC(6789, '+*****')` returns `+*6789`

`$PIC(-6789, '+*****')` returns `-*6789`

- If the sign holder is negative (-) and the number is positive, the sign holder is changed to a blank.

`$PIC(3456, 'NZ$-*****')` returns `NZ$ *3456`

`$PIC(-3456, 'NZ$-*****')` returns `NZ$-*3456`

Conditional Right String

Use a conditional right string to append a set of message characters to the basic string that appear only when *value* is negative. For example, this element can be used to append the characters CR when a credit is due. The conditional right string consists of all characters after the last digit placeholder, unless a percent sign (%) marks the start of an unconditional right string. If a percent (%) sign is present, the conditional right string consists of all characters between the last digit placeholder and the percent sign, which signals the beginning of the unconditional right string (refer to [Unconditional Right String](#) below). The conditional right string follows these rules:

- If *value* is negative, the characters in the conditional right string are appended to the basic string. The characters CR and AUS are conditional right strings in the examples below.

```
$PIC(-1234, 'CDN$-ZZZZV.ZZCR') returns CDN$-1234.00CR
$PIC (-2345, '-*****AUS') returns -*2345AUS
```

- If *value* is positive, the characters in the conditional right string are replaced with characters that depend on the first placeholder in the basic string:

If the first placeholder is...	The conditional right string is replaced by...
9 or Z	Blanks
N	Nulls
An asterisk (*)	Asterisks (*)

```
$PIC(1234, 'CDN$-ZZZZV.ZZCR') returns CDN$ 1234.00
$PIC(2345, '-*****AUS') returns *2345***
$PIC(2345, '-NNNNNAUS') returns 2345
```

Unconditional Right String

Use the unconditional right string to append characters to the basic string or conditional right string that always appear regardless of whether *value* is positive or negative. The unconditional right string begins with a percent sign (%) character. Characters that follow a percent sign (%) character appear as message characters, including all placeholders and characters with special functions.

- \$PIC(567.89, 'NNNV.NNN%VALUE') returns 567.890%VALUE
- \$PIC(-7890.3, 'AUS\$-ZZZ,ZZZV.ZZCR%PAID') returns
AUS\$-7,890.30CR%PAID

See Also The [\\$UNPIC](#) tool, which, given a masked value produced by \$PIC and the display mask that produced it, determines the original value submitted.

Example The following rule formats an input amount so that it appears with commas, a decimal marker, and a negative sign if necessary, and then displays the formatted amount in the end message.

RULE EDITOR ==>		SCROLL: P
DISP_DOLLAR(AMOUNT);		
_ LOCAL AMOUNT2;		
_ -----		
_ -----		
_ AMOUNT2 = \$PIC(AMOUNT, '-\$NNN,NNN,NNNV.99');		1
_ CALL ENDMSG(AMOUNT2);		2
_ -----		

-
- The input value 47569808 is returned as \$47,569,808.00.
 - The input value -78987.20 is returned as -\$78,987.20.

@PRESENTATIONENV

Returns the name of the presentation environment for the current session. (F)

Invocation `environment = @PRESENTATIONENV`

`environment` Returned value of the presentation environment. Its syntax is C with length 16.

Possible values are TEXT (if the rule is executing from the workbench) and NONE (if the rule is executing in batch).

Example The following rule returns the value of the presentation environment for the session and returns the value to the end message:

```

RULE EDITOR ==>                                SCROLL: P
PRESENT_1;
_ LOCAL ENVIRONMENT;
_ -----+-----
_ -----+-----
_ ENVIRONMENT = @PRESENTATIONENV;                | 1
_ CALL ENDMSG('THE PRESENTATION ENVIRONMENT IS ' || | 2
_ ENVIRONMENT);                                  |
_ -----+-----

```

The following message is returned, if the presentation environment is the workbench:

THE PRESENTATION ENVIRONMENT IS TEXT

PRINT_DATA

Prints the data of a TIBCO Object Service Broker table. (C)

Invocation `CALL PRINT_DATA(tablespec, select, sourceloc)`

<i>tablespec</i>	The name of the table or table instance.
<i>select</i>	The selection criteria to be used.
<i>sourceloc</i>	The name of the node where the data is located.

- Usage Notes**
- If the table specified in *tablespec* is parameterized, you specify only the data parameters, not the location parameters.
 - If you specify an empty string for *select*, all the occurrences of the table are printed.
 - The syntax for *select* is *<field name> <relational operator> <value>*.
 - Specify a value for *sourceloc* only if the data is located on a different node.
 - The output is sent to the printer specified in your user profile.

See Also *TIBCO Object Service Broker Managing Security* for information about your user profile.

Exceptions

NO_PRINT	The call to PRINT_DATA did not complete. The exception should be handled by the calling rule. Further information describing the circumstances of the failure is in the @OBJECTMSG.MSG field.
-----------------	--

Example The following rule prints selected data from the table instance EMPLOYEES(MIDWEST).

```

RULE EDITOR ==>
PRINT_DATA_1;
Library: DOCEXEMPL |
-----+-----
CALL PRINT_DATA( 'EMPLOYEES(MIDWEST)', 'MGR#=79912', '' );
-----|
ON NO_PRINT  :
```

```
CALL ENDMSG('TABLE NOT PRINTED BECAUSE ' || @OBJECTMSG.MSG);
```

PRINT_DEFN

Prints the definition of a TIBCO Object Service Broker object. (C)

Invocation `CALL PRINT_DEFN(object, instance, library, environment, srcloc, parentonly)`

<i>object</i>	The TIBCO Object Service Broker object type of the object that is to be printed. Valid object types are: <ul style="list-style-type: none"> • GLOBALFIELD • LIBRARY • MENU • OBJECTSET • REPORT • RULE • SCREEN • TABLE • WEBSERVICEPROD
<i>instance</i>	The name of the object that is to be printed.
<i>library</i>	If the object is a rule, the name of the rules library where the rule is stored.
<i>environment</i>	This argument, although not currently used, must be supplied. You can enter a null (") value.
<i>srcloc</i>	The name of the node where the object is located.
<i>parentonly</i>	Specifies if all the objects or only the parent object should be printed. Valid values are: Y – Print only the parent. N – Print the parent and child objects.

- Usage Notes**
- Specify a value for *srcloc* only if the object is located on a different node.
 - Specify a value for *parentonly* only if the object is composed of one or more other objects (for example, a report is composed of report tables).

- The results are printed in the message log and the output is sent to the printer specified in your User Profile.

Exceptions

PRINT_ERROR	Due to an error (for example, lack of security permissions), the definition cannot be printed. More information describing the circumstances of the failure is in the @OBJECTMSG.MSG field.
-------------	--

Example The following rule prints the definition of a menu that is used in a 3270 environment:

```
RULE EDITOR ==>                                SCROLL: P
PRINT_DEFN_1;
-
- -----
- |                                     | 1
- CALL PRINT_DEFN('MENU', 'SCR_EMPLOYEE', '', '3270', '', |
- 'N');                                         |
- -----
```

Results Sent to the Message Log

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
**** Begin to print all definitions for object "SCR_EMPLOYEE" ****
Begin to print definition for object "SCR_EMPLOYEE" type "MENU"
Table "SESSION_MENUS" printed
Table "MENU_ITEMS" printed
Print definitions for object "SCR_EMPLOYEE" type "MENU" completed

**** Print all definitions completed ****
```

\$PRINTFIELD

Writes the specified string into the current printline. (C)

Invocation

CALL \$PRINTFIELD(*string*, *pos*, *length*, *fill*, *just*)

<i>string</i>	The string to be inserted. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>pos</i>	An integer specifying the position at which to insert the string. Its syntax is B (binary) with length 2.
<i>length</i>	An integer specifying the length to which to pad the string. Its syntax is B with length 2.
<i>fill</i>	A printable string specifying the character with which to pad the string. Its syntax is C with length 1.
<i>just</i>	One of: LEFT or L – Specifies that the string should be left justified. RIGHT or R – Specifies that the string should be right justified. CENTER, CENTRE, or C – Specifies that the string should be centered. Its syntax is C with length 8.

Usage Notes

- The print arguments must be previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to \$PRINTFIELD.
- Use [\\$PUTLINE](#) to output the printline.

Exceptions

RANGERROR	This exception is raised for one of the following reasons: pos – is less than or equal to zero. length – is less than zero. fill – is not a printable character. just – is not one of C, CENTER, CENTRE, L, LEFT, R, or RIGHT.
------------------	--

ROUTINEFAIL	\$PRINTFIELD is not preceded by a call to \$RESETPRINT or \$SETPRINT.
STRINGSIZE	The length of <i>text</i> is greater than the print <i>width</i> (where <i>width</i> is the page width set by \$RESETPRINT or \$SETPRINT).

Example The following rule prints a string to the message log:

```
PRINTFIELD_1;
_ LOCAL SOURCE_STRING;
_ -----+-----
_ -----+-----
_ CALL $SETPRINT(10, 70, 1, 'SCR', 'N');           | 1
_ SOURCE_STRING = 'THIS IS THE SOURCE STRING';      | 2
_ CALL $PRINTFIELD(SOURCE_STRING, 1, 70, '*', 'L'); | 3
_ CALL $PRINTFIELD('OUTPUT', 13, 6, '', 'C');        | 3
_ CALL $PUTLINE;                                     | 4
_ -----+-----
```

Displayed Output for the PRINTFIELD_1 Rule

Using PF2 after executing this rule displays the following screen. The second call to \$PRINTFIELD replaced the word SOURCE from SOURCE_STRING with the word OUTPUT.

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P

----- NEW PAGE -----

                                     Page 1
THIS IS THE OUTPUT STRING*****
```

\$PRINTLINE

Prints a string. (C)

Invocation `CALL $PRINTLINE(text)`

<i>text</i>	The character string to print. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
-------------	---

- Usage Notes**
- The print arguments must be previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to [\\$PRINTLINE](#).
 - The length of text must be less than or equal to the width specified by the [\\$SETPRINT](#) or [\\$RESETPRINT](#).
 - Subsequent output is printed on the next line.

Exceptions

ROUTINEFAIL	\$PRINTLINE is not preceded by a call to \$RESETPRINT or \$SETPRINT .
STRINGSIZE	The text length exceeds the width specified by \$RESETPRINT or \$SETPRINT .

Example The following rule prints a line to the message log:

```
PRINTLINE_1;
```

```

-
- -----
-                                     +-----+
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N'); | 1
- CALL $PRINTLINE('THIS LINE IS PRINTED TO THE MESSAGE LOG'); | 2
- -----

```

Displayed Output from \$PRINTLINE_1

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
----- NEW PAGE -----

```

THIS LINE IS PRINTED TO THE MESSAGE LOG

PRINTTABLE

Prints a table. (C)

Invocation `CALL PRINTTABLE(tablespec, pagelength, pagewidth, media)`

<i>tablespec</i>	A string specifying the name of the table to be printed, including parameters (if any).
<i>pagelength</i>	An integer specifying the number of lines on a page, usually 25 for a screen or 60 for a page.
<i>pagewidth</i>	An integer specifying the number of columns to use on the page. This value is usually 80 for a screen; it can be greater, as some printers can handle 132. NOTE: Fields that do not fit within a page are not printed.
<i>media</i>	A string specifying the medium (screen or printer) to which output is directed. It must be one of: SCR – Sends output to the screen. PRT – Sends output to the printer.

- Usage Notes**
- You must declare the local variable MSG. It contains a message indicating if the table printed successfully or why it did not print successfully.
 - Fields that contain raw data (syntax RD) or Unicode data (syntax UN) are printed as strings of hexadecimal characters.

Exceptions

<code>NO_PRINT</code>	Raised if the table cannot be printed successfully.
-----------------------	---

Example The following rule prints the table instance 01 of the EMPLOYEE_EXPENSE table and displays the result in the message log:

```
PRINTTABLE_1;
LOCAL MSG;

- -----+-----
- -----+-----
- CALL PRINTTABLE('EMPLOYEE_EXPENSE(01)', 60, 132, 'SCR');      | 1
- CALL ENDMSG(MSG);                                           | 2
- -----+-----
- ON NO_PRINT :
-   CALL ENDMSG(MSG);
```

The end message contains the following:
12:43:23 printed EMPLOYEE_EXPENSE(01)

Output from the PRINTTABLE_1 Rule

Pressing PF2 after executing the rule displays the following screen:

Printing Table: EMPLOYEE_EXPENSE(01)				Page 1.1	
EMPNO	LNAME	POSITION	MGR#	DEPTNO	MON_EXP
80000	SMYTHE	Director	80002	20	2204.95
80002	ROTTERDAM	VP	99999	50	1411.90
80003	CHANG	Assoc. Analyst	83020	10	0.00
80004	GARZA	Analyst	80009	30	0.00
80014	TOWNSEND	President & GM	84021	70	2859.02
80019	PASTARINA	Mgr	80033	50	0.00
80020	CHESSEL	Secretary	83020	30	0.00
80021	TOWENSEND	Receptionist	84021	50	0.00
80024	NAPIER	Sales Person	80020	20	377.73
80033	CANON	Programmer	80020	40	0.00
81000	NELSON	Daycare Mgr	80002	60	0.00
81001	CAREY	Secretary	81092	40	0.00
81003	CHIU	VP	81033	40	960.26
81014	LYNGBAEK	Mgr	84021	40	93.45
81019	KINGSON	CEO	81092	50	1381.19

PROCESS_FCNKEY

Processes the function keys while a screen is being displayed. (C)

Invocation `CALL PROCESS_FCNKEY(screen)`

<i>screen</i>	The name of the screen being displayed.
---------------	---

Usage Notes The PF keys must be predefined to the table FCNKEYS.

Example The following rule displays a screen called NEW_EMPLOYEE and, using PROCESS_FCNKEY, processes the function keys while the screen is being displayed.

- Before you run this rule, you must define the NEW_EMPLOYEE screen and it must contain a screen table called FCNKEY_SPECS, which has a field called FCNKEYS. This is where the function keys appear.
- [FCNKEY_MSG](#) is a tool that uses the FCNKEYS table to create a string listing the function keys defined for the screen.
- One of the function keys must invoke the EXIT_DISPLAY rule, which stops displaying the screen NEW_EMPLOYEE.

```

NEW_EMPLOYEE;
_
_  -----
_  -----
_  FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');      | 1
_  INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                     | 2
_  UNTIL EXIT_DISPLAY DISPLAY NEW_EMPLOYEE :                 | 3
_    CALL PROCESS_FCNKEY('NEW_EMPLOYEE');                     |
_    END;                                                      |
_  -----

```

PROCESS_TABLE

Provides specific processing for every occurrence in a table that is selected, ordered, or both selected and ordered. (C)

Invocation `CALL PROCESS_TABLE(tablespec, selection, ordering, processrule)`

<i>tablespec</i>	A string specifying the table name and parameters, if any.
<i>selection</i>	A string specifying selection criteria.
<i>ordering</i>	A string specifying ordering criteria.
<i>processrule</i>	The name of the rule to be called for each occurrence in the table that satisfies the selection/ordering criteria.

Usage Notes

- PROCESS_TABLE sets up a FORALL on the table.
- Calls to PROCESS_TABLE can be nested (that is, *processrule* can call PROCESS_TABLE on a different table).
- The syntax for selection is <field name><relational operator><expression>. For valid expressions, refer to the *TIBCO Object Service Broker Programming in Rules* manual.
- The syntax for *ordering* is *direction* (that is, ascending or descending) and *field name*.
- *processrule* must not be a function and it must not have arguments.
- *processrule* and its descendant rules must not reference the following local variables: TABLE, SELECTION, ORDERING, PROCESSRULE, and GIVENTABLENAME.

Exceptions

NO_ENTRIES	Raised if no occurrences are selected.
PARSER_ERROR	Raised if invalid <i>tablespec</i> , <i>selection</i> , or <i>ordering</i> is specified.

Example

In the following examples, the process rule changes the salary of selected employees.

The first example shows the rule to be processed:

```

RULE_EDITOR ==>
PROC_EMPLOYEE_R;

-----
EMPLOYEE_DEPT.SALARY = EMPLOYEE_DEPT.SALARY * 1.05;      | 1
REPLACE EMPLOYEE_DEPT(10);                                | 2
-----

```

The second example calls in PROCESS_TABLE:

```

RULE_EDITOR ==>
PROC_EMPLOYEE(SELECTION);

-----
CALL PROCESS_TABLE('EMPLOYEE_DEPT(10)', SELECTION, '',    | 1
'PROC_EMPLOYEE_R');                                       |
-----

```

- Typing `PROC_EMPLOYEE('SALARY < 500.00')` at the EX Execute Rule option on the workbench increases the salaries of all the employees whose salary is less than \$500.00 in the table `EMPLOYEE_DEPT(10)`.
- Typing `PROC_EMPLOYEE('MGR#=79912')` at the EX Execute Rule option on the workbench increases the salaries of all the employees managed by MGR# 79912 in the table `EMPLOYEE_DEPT(10)`.
- Typing `PROC_EMPLOYEE('EMP_LNAME LIKE "F*" & MGR# = 80354')` at the EX Execute Rule option on the workbench, increases the salaries of all employees whose last name starts with F and who are managed by MGR# 80354 in the table `EMPLOYEE_DEPT(10)`.

@PROMBINDOBJS

Restores the bind flag settings for the objects updated by @PROMUNBINDOBJS.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	EX execute rule option	Type @PROMBINDOBJS <Enter>
	COMMAND prompt	Type EX @PROMBINDOBJS<Enter>

Usage Notes



In a multi-user environment, all users are impacted after the tool is executed. They could experience performance degradation since all bound data must be rebuilt.

- @PROMBINDOBJS must be run using a Level-7 userid.
- Any objects which could not be updated are reported in the information log and the session return code is set to 4.
- You must recycle the Execution Environment after successfully running @PROMBINDOBJS to pick up the new settings.

See Also *TIBCO Object Service Broker Managing Deployment* for information about using the Promotions System.

PROM_MAIN

Invokes directly the Promotion system. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	PM promotion option	Press Enter
	EX Execute Rule option	Type PROM_MAIN <Enter>
	COMMAND prompt	Type EX PROM_MAIN <Enter>

When you execute PROM_MAIN, the main menu of the Promotion system appears.

Usage Notes The Browse flag at the top of the screen should be set to Y to ensure the data is not locked.

See Also *TIBCO Object Service Broker Managing Deployment* for complete information about using the Promotion system.

@PROMUNBINDOBJS

Stores the current setting of the bind flag for a set of objects and resets the values to N in the metadata tables. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	EX Execute Rule option	Type @PROMUNBINDOBJS(<i>scope</i>) <Enter>
	COMMAND prompt	Type EX @PROMUNBINDOBJS(<i>scope</i>) <Enter>

where *scope* is the set of objects you want to unbind. Value values are ALL, INSTALLATION, and SYSTEM.

Usage Notes

If for scope you specify...	The following occurs
ALL	Resets the bind flag for all objects which have a current value of Y or B.
INSTALLATION	Reset the bind flag for installation objects only. These are objects with an AUTHOR not equal to HURON or HURON2.
SYSTEM	Resets the bind flag for system objects only. These are objects with an AUTHOR equal to HURON or HURON2.

- Run @PROMUNBINDOBJS prior to applying a change or series of changes to unbind a set of bound objects. Unbinding objects prior to applying a change prevents Promotions Systems warning messages associated with bound objects from being issued. Consequently you will not need to resubmit changes associated with these types of messages.
- @PROMUNBINDOBJS must be run using a Level-7 userid
- If @PROMUNBINDOBJS fails after only some objects have been updated, you must run @PROMBINDOBJS to restore the original bind information before re-running @PROMUNBINDOBJS.

- Any objects that could not be updated are reported in the information log and the session return code is set to 4.
- You must recycle the Execution Environment after successfully running @PROMUNBINDOBS to pick up the new settings.

See Also *TIBCO Object Service Broker Managing Deployment* for information about using the Promotions System.

PRT_VSCR

Prints the screen fields of a defined screen in a page format, with or without a mask. (C)

Invocation `CALL PRT_VSCR(vscr, page_length, page_width, page_start, media, mask)`

<i>vscr</i>	The name of a defined screen.
<i>page_length</i>	The length of the printed page.
<i>page_width</i>	The width of the printed page.
<i>page_start</i>	The page number that is on the first printed page. It is also the start number. If a value of " or N is entered, NEW PAGE prints across the top, rather than a page number.
<i>media</i>	One of: PRT – Send the print of the screen to the printer. SCR – Send the print of the screen to the message log.
<i>mask</i>	One of: Y – Print the default mask values, that is, AAAAs for alphabetic fields and 9999s for numeric fields. N – Print the screen table fields only.

Usage Notes The full screen is printed.

Example In this example, PRT_VSCR is called into the PRINT_EMP rule.

```
PRINT_EMP;
_
_ -----+-----
_ -----+-----
_ CALL PRT_VSCR('NEW_EMPLOYEE', 60, 80, '', 'PRT', 'Y');      | 1
_ CALL SCREENMSG('NEW_EMPLOYEE', 'EMPLOYEE SCREEN PRINTED'); | 2
_ -----+-----
```

FCNKEYS Table

The PRINT_SCREEN rule is defined to PF13 in the FCNKEYS(EMPLOYEE_EXP) table:

EDITING TABLE : FCNKEYS(NEW_EMPLOYEE)			
COMMAND ==>			
PF_KEY	NAME	COMMAND	ROUTINE

_ PF3	SAVE		SAVE_EMP
_ PF12	EXIT		EXIT_DISPLAY
_ PF13	PRINT		PRINT_EMP

PURGELOG_BATCH

Purges the audit log data from the TIBCO Object Service Broker audit log table and archives it to an external file. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type PURGELOG_BATCH <Enter>
	COMMAND prompt	Type EX PURGELOG_BATCH <Enter>
From a rule		Type CALL PURGELOG_BATCH (<i>fromdate</i> , <i>todate</i> , <i>file</i>)
From a batch file		Submit a batch file job that invokes the PURGELOG_BATCH tool.

Where:

<i>fromdate</i>	The start date of the range of audit log data to be purged (Specify in YYYYMMDD format, for example, 20000101.)
<i>todate</i>	The end date of the range of audit log data to be purged (Specify in YYYYMMDD format, for example, 20000131.)
<i>file</i>	The archive filename.

Prerequisites Two external resource rules must be set up so that the user can specify the archive filename and purge the audit log.

See Also *TIBCO Object Service Broker Managing Security* for more information on this tool.

Constraints

- The Execution Environment must be in single user mode if used on z/OS.
- The Data Object Broker must be authorized and be run using Cross Memory Services (XMS) if used on z/OS.
- External security must be established.

- Current (that is, today's) audit logs cannot be purged.

Example

Assume an installation is externally secured using ACF2. An authorized user would define two resource rules, for example, ABCS6B.SPECFILE and ABC2S6B.PURGELOG. For each of these resource files, a list of user IDs who can perform the file specification and the purging of the audit log is created.

According to whatever schedule is appropriate to that site, a JCL job that does a SCHEDULE PURGELOG_BATCH is submitted by users authorized through the resource rules.

PURGELOG_SCREEN

Specifies the archive file for the audit log data and purges the audit log data after writing it to the specified archive file. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type PURGELOG_SCREEN <Enter>
	COMMAND prompt	Type EX PURGELOG_SCREEN <Enter>
From a rule		Type CALL PURGELOG_SCREEN

Prerequisites @SEC_PURGELOG object set must be enabled in order for the user to be able to access the PURGELOG_SCREEN utility. Two external security resource rules must be set up to specify the archive file and purge the audit log successfully.

See Also *TIBCO Object Service Broker Managing Security* for information on the Audit Log facility.

Usage Notes

- Authorized users of the PURGELOG_SCREEN tool can archive and purge data from the audit log. Its screen takes a date range and an archive filename, both of which are required fields. The default date range is given by the date of last archive up to yesterday's date. The default filename is the name of the file used for the last archive.
- To review the portion of the audit log that reflects the date parameters established on the screen, press PF4, although in general the Audit Log facility should be used for this purpose.
- Archiving the audit log on a regular basis keeps it from growing to unmanageable sizes.

Constraints

- The Execution Environment must be in single user mode if used on z/OS.
- The Data Object Broker must be authorized and be run using Cross Memory Services (XMS) if used on z/OS.
- External security must be established.

- Current (that is, today’s) audit logs cannot be purged.

Example The following screen archives audit log data entered within the specified date range to the file LOG_ARCHIVE:

```
-----
                          Archive and Purge Audit Log
-----

Purge dates ranging from: 19991215 to: 20000115

Archive File: LOG_ARCHIVE_____

PFKEYS: 1=HELP 3=PURGE & EXIT 12=EXIT 4=VIEW ACCESSLOG
```

\$PUTCONTAINER

Places data in a container associated with the specified channel. (C)

Invocation `CALL $PUTCONTAINER(channel, container, area, length, fromccsid, datatype)`

<i>channel</i>	The name (1-16 characters) of the channel that owns the container.
<i>container</i>	The name (1-16 characters) of the container in which to place data.
<i>area</i>	The pointer to an area from which the data is written to the container.
<i>length</i>	A fullword binary value that represents the length of the area from which to read data.
<i>fromccsid</i>	A fullword binary number that represents the Coded Character Set Identifier (CCSID). The character data to be put into the container will be converted into this format. For an explanation of CCSIDs, see the section "Data Conversion With Channels" in the <i>CICS Application Programming Guide</i> .
<i>datatype</i>	The type of data to put into the container. This option applies only to new containers. The data type of an existing container was established at creation and cannot be changed. For more information on data conversion with channels, see the <i>CICS Application Programming Guide</i> .

Example Following is a sample rule:

```

RULE EDITOR ==>
C_$PUTCONTAINER(CHANNEL, CONTAINER, TNAME);
LOCAL AREA, LENGTH, FROMCCSID, DATATYPE;
-----
-----+-----
@MAP.ADDRESS = 0;                                | 1
@MAP.SIZE = 80;                                  | 2
INSERT @MAP('ENVIRONMENT');                      | 3
AREA = @MAP.ADDRESS;                             | 4
LENGTH = @MAP.SIZE;                              | 5
GET MAP_CONTAINER(AREA);                          | 6
MAP_CONTAINER.F10 = TNAME;                        | 7
REPLACE MAP_CONTAINER(AREA);                     | 8
CCSID = '';                                       | 9
DATATYPE = '';                                  | A
CALL MSGLOG(                                     | B
'$PUTCONTAINER(CHANNEL, CONTAINER, AREA, LENGTH, FROMCCSID, '||

```

```
- 'DATATYPE)');
- CALL MSGLOG('Channel name is: ==> ' || CHANNEL);
- CALL MSGLOG('Container name is: ==> ' || CONTAINER);
- CALL MSGLOG('Container content: ');
- CALL MSGLOG(MAP_CONTAINER.F10);
- CALL $PUTCONTAINER(CHANNEL, CONTAINER, AREA, LENGTH, FROMCCSID,
-   DATATYPE);
- -----
```

Following is the MAP table MAP_CONTAINER:

TABLE DEFINITION													
COMMAND==>													
Table: MAP_CONTAINER					Type: MAP		Unit: HZS80				IDgen: Y		
Parameter Name		Typ	Syn	Len	Dc	Cls	Reference			Event Rule		Typ	Acc
-----		-	-	-	-	-	-----		,	-----		-	-
ADDRESS		B		4	0	A			,				
LOCATION		I	C	16	0	L			,				
									,				
		EXTERNAL				----- -----		Metadata Definition -----					
Field Name		Xsyn	Xlen	Xdec	Offset	Key	Typ	Syn	Len	Dec	Rqd	Default	
-----		-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	
KEY		B		4	0	0	P	I	B	4	0		
F10		C		80	0	0			C	80	0		

\$PUTLINE

Prints the current line constructed by \$PRINTFIELD. (C)

Invocation CALL \$PUTLINE

- Usage Notes**
- Format the printline using \$PRINTFIELD.
 - When the line is printed, subsequent calls to \$PRINTFIELD construct a new line.

Exceptions	LOGLIMIT	Too much output is sent to the message log.
	ROUTINEFAIL	\$PUTLINE is not preceded by a call to \$RESETPRINT or \$SETPRINT.

Example The following rule constructs a string using \$PRINTFIELD and then prints it to the message log:

```
PUTLINE_1;
_
_
_-----+-----
_ CALL $SETPRINT(10, 70, 1, 'SCR', 'N'); | 1
_ CALL $PRINTFIELD( | 2
_ 'THIS LINE IS PRINTED TO THE MESSAGE LOG', 1, 70, '*',
_ 'C'); |
_ CALL $PUTLINE; | 3
_-----
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P

----- NEW PAGE -----

                                           Page 1
*****THIS LINE IS PRINTED TO THE MESSAGE LOG*****
```

QUOTE

Returns a string with single quotation marks around it and doubles any single quotation marks in the string. (F)

Invocation `quoted = QUOTE(string)`

<i>quoted</i>	String with single quotation marks at the beginning and end. Single quotation marks in <i>string</i> are changed to two single quotation marks.
<i>string</i>	The character string to which the single quotation marks are added.

Example The following rule adds quotation marks to a string that is passed to [MESSAGE](#). The quotation marks are needed for the third argument because MESSAGE uses [TOKEN](#) to separate the tokens in this argument. Each token is used to substitute for a percent sign (%) in the message in the MESSAGES table. Message number 2 in the TABLE table instance is: At %.

By putting quotation marks around the string, it becomes a quoted string. TOKEN treats a quoted string as one token, and so the entire string substitutes for the percent sign.

```

DISPLAY_MSG2(TABLENAME);
_  LOCAL TEXTSTRING;
_  -----
_  -----+-----
_  TEXTSTRING = 'the TOP of table "' || TABLENAME || '"';      | 1
_  RETURN(MESSAGE('TABLE', 2, QUOTE(TEXTSTRING)));              | 2
_  -----

```

If the table name passed to DISPLAY_MSG2 is EMPLOYEE, the message that is returned is:

At the TOP of table "EMPLOYEE".

RANDOM

Returns a random integer greater than or equal to 1 and less than or equal to the specified limit. (F)

Invocation `number = RANDOM(rangelimit)`

<i>number</i>	On return, contains the random number. Its syntax is B (binary) with length 4.
---------------	--

<i>rangelimit</i>	An integer specifying the largest allowable random number. The <i>rangelimit</i> can be positive or negative if its absolute value is less than or equal to $(2^{31})-1$. Its syntax is B (binary) with length 4.
-------------------	--

- Usage Notes**
- The random integer returned is always positive (that is, the absolute value of the random number is returned).
 - If [RANDOMSEED](#) is not called, an initial seed is randomly selected.
 - If 0 is the *rangelimit* for RANDOM, it substitutes 1.

Exceptions

OVERFLOW	Raised if the absolute value of <i>rangelimit</i> is greater than $(2^{31})-1$
-----------------	--

Example The following rule generates random salaries for a table of test data:

```
GENERATE_DATA;
```

```

- -----
- -----
- -----+-----
-  FORALL TEST_EMP_DATA :      | 1
-    TEST_EMP_DATA.SALARY = RANDOM(1000);
-    REPLACE TEST_EMP_DATA;    |
-    END;                      |
- -----
```

Sample Data

If you view the table through the Table Browser after executing this rule, you see a random number in the **SALARY** field. A sample of the table is shown as follows:

BROWSING TABLE : TEST_EMP_DATA
COMMAND ==>

	EMPNO	LNAME	SALARY
—	80000	SMYTHE	964.00
—	80002	ROTTERDAM	109.00
—	80003	CHANG	34.00
—	80004	GARZA	909.00
—	80005	HANSON	779.00
—	80006	MILMAN	426.00

PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 12=END

RANDOMSEED

Sets the starting seed for the random number generator. (C)

Invocation `CALL RANDOMSEED(seed)`

<i>seed</i>	An integer specifying the seed value. It can take on positive or negative values if the absolute value is less than or equal to $(2^{*}31)-1$. Its syntax is B (binary) with length 4.
-------------	---

Usage Notes *seed* can be 0.

Exceptions

OVERFLOW	Raised if the absolute value of <i>seed</i> is greater than $(2^{*}31)-1$.
-----------------	---

Example The following rule sets the seed for the random number generator and generates test data for the TEST_EMP_DATA table:

<pre> RULE EDITOR ==> GENERATE_DATA(SEED_NUMBER); -- -- ----- -- CALL RANDOMSEED(SEED_NUMBER); -- FORALL TEST_EMP_DATA : -- TEST_EMP_DATA.SALARY = RANDOM(1000); -- REPLACE TEST_EMP_DATA; -- END; -- ----- </pre>	<p>SCROLL: P</p> <table border="1"> <tr><td>1</td></tr> <tr><td>2</td></tr> </table>	1	2
1			
2			

Sample Data

If you execute this rule and give it a number as an argument, you see random numbers in the **SALARY** field when you browse the table with the Table Browser. You see the same random salaries if you execute the rule again with the same number as an argument. A sample of the table is shown as follows:

BROWSING TABLE : TEST_EMP_DATA		
COMMAND ==>		
EMPNO	LNAME	SALARY

—	80000	SMYTHE	156.00
—	80002	ROTTERDAM	750.00
—	80003	CHANG	47.00
—	80004	GARZA	60.00
—	80005	HANSON	648.00
—	80006	MILMAN	773.00

PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 12=END

@READDSN

Returns the next record from the current file. (F)

Invocation `record = @READDSN`

<code>record</code>	On return, contains the record that was read. Its syntax can be either C (fixed-length character string) or V (variable-length character string).
---------------------	---

- Usage Notes**
- The file must be previously specified using [@OPENDSN](#).
 - You must have read access to the file.
 - An attempt to open the file is made with the first read operation.
 - If the file specified in the @OPENDSN statement does not exist, @READDSN fails.
 - On the z/OS platform, @READDSN always reads in EBCDIC format.
 - On a TIBCO Object Service Broker for Windows system, when the data is written to the external file, it is subject to the type specification for the file as given in filespec.dsn or by the DSBIFTYPE Execution Environment parameter. If the file type is LENGTH_PREFIXED_EBCDIC, the data is left alone and read as EBCDIC. If the file type is LINE_SEPARATED_ASCII, the data is converted from ASCII to EBCDIC when read, and back from EBCDIC to ASCII when written (using @WRITEDSN).
 - @READDSN accesses a z/OS file using the data set name. There is no provision for using a DDNAME with this tool instead of a data set name.

Exceptions	ENDFILE	Raised if the rule attempts to read past the last record.
	ROUTINEFAIL	Raised if the file is not specified through @OPENDSN or if the file does not exist.

Example The following rule (READDSN_1) specifies an existing file, writes data from the example table to it, closes the file, re-specifies it, and calls another rule.

```
READDSN_1;  
_ LOCAL RECORD;  
_ -----
```

```
- -----+-----
- CALL @OPENDSN(TSOID || '.EXAMPLES.DATA');          | 1
- FORALL EMPLOYEE :                                  | 2
-     CALL @WRITEDSN(EMPLOYEE.LNAME);                |
-     END;                                           |
- CALL @CLOSEDSN;                                    | 3
- CALL @OPENDSN(TSOID || '.EXAMPLES.DATA');          | 4
- CALL READFILE;                                     | 5
- CALL @CLOSEDSN;                                    | 6
- CALL MSGLOG('END OF FILE');                        | 7
- -----
```

READFILE;

```
- -----+-----
- UNTIL ENDFILE:                                     | 1
-     RECORD = @READDSN;                             |
-     CALL MSGLOG(RECORD);                           |
-     END;                                           |
- -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following output:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
DRABEK
ROEDER
HOEGSON
TERAMURA
LEES
MORANG
CROFTON
SMITH
SOUZA
SAUNDERS
HRODEK
CANNON...
```


REALTIME

Returns a string containing the current time of day. (F)

Invocation `string = REALTIME`

<code>string</code>	The string containing the current time of day. Its syntax is C (fixed-length character string) with length 8.
---------------------	---

- Usage Notes**
- The time of day is returned in the format HH:MM:SS; for example, 23:59:59.
 - The current time is returned, not the time the transaction started.

Example The following rule determines the current time of day and prints it to the message log:

```

REALTIME_1;
_ LOCAL TIME;
_ -----
_ -----+-----
_ TIME = REALTIME;                               | 1
_ CALL MSGLOG('THE CURRENT TIME OF DAY IS: ' || TIME); | 2
_ -----

```

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
THE CURRENT TIME OF DAY IS: 07:42:20

```

\$REALTIMER

Returns the number of micro-seconds since 1 January 1980. (F)

Invocation microseconds = \$REALTIMER

microseconds	A number containing the number of micro-seconds. Its syntax is P (packed decimal) with length 16 and 0 decimals.
--------------	--

- Usage Notes**
- The value returned is based on the clock set for the CPU (which is generally set to GMT/UTC) and represents the time elapsed from 1 January 1980 without modification due to time zones. \$REALTIMER should therefore be used only for deltas.
 - The current number of microseconds is returned (not the microseconds when the transaction started).

Example The following rule returns the number of microseconds since 1980 and returns it to the message log:

```

REALTIMER_1;
_ LOCAL TIME;
_ -----
_                                     +-----
_ TIME = $REALTIMER;                  | 1
_ CALL MSGLOG('THE NUMBER OF MICROSECONDS IS: ' || TIME );    | 2
_ -----

```

Resulting Output

Pressing PF2 after executing this rule displays the following on the screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
THE NUMBER OF MICROSECONDS IS:  422291030198808

```

REFMAKER

Rebuilds the global cross reference index. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	XR Global CrossRef Build option	Type library <Enter>
Developer's workbench	EX Execute Rule option	Type REFMAKER(<i>library</i>) <Enter>
	COMMAND prompt	Type EX EFMAKER(<i>library</i>) <Enter>

Where:

<i>library</i>	The name of the library to be indexed.
----------------	--

Usage Notes

- REFMAKER rebuilds the global cross reference index that is used by [SEARCH](#) and [CROSSREFSEARCH](#) to answer queries.
- To keep the cross reference index current, REFMAKER should be run after every promotion. You should plan to run REFMAKER in batch mode with all users suspended.
- For more information about REFMAKER, refer to *TIBCO Object Service Broker Managing Deployment*.

Detranslation of Rules

REFMAKER makes one pass over the specified library, detranslates the rules in the library, and makes entries into the global cross reference index based on the following order of evaluation:

- A direct call
- Event rules associated with the specific access type (W, I, R, D, or F)
- Indirection through the field of a table

If the table has no data parameters, all the rules in the field are included. If the table has one data parameter, you must first specify the instance or instances required in the table @IND_CALLS.

- Functions used within the rule

A global variable on the right-hand side of a statement is included in the index as a function, unless the variable is also used further on in the same rule, on the left-hand side, in which case it is not included.

- Screen validation rules associated with a DISPLAY statement
- TIBCO Object Service Broker routines used within the rule

REMAINDER

Returns the remainder from dividing the dividend by the divisor. (F)

Invocation `number = REMAINDER(dividend, divisor)`

<i>number</i>	On return, contains the remainder. Its syntax depends on the syntax of the dividend and divisor.
<i>dividend</i>	The number to be divided. Its syntax can be any of the numeric syntaxes.
<i>divisor</i>	The number to divide by. Its syntax can be any of the numeric syntaxes.

- Usage Notes**
- The values for both *dividend* and *divisor* must be positive.
 - The returned value is the smallest, non-negative value R such that the term $(\text{dividend} - R) / \text{divisor}$ is an integer.
 - REMAINDER does not alter the values of either *dividend* or *divisor*.
 - The function **MOD** calls REMAINDER and handles negative dividends and divisors.

Exceptions

ZERODIVIDE	Raised if the second operand is zero.
-------------------	---------------------------------------

Example The following rule determines the remainder of a division operation and prints it to the message log:

```
REMAINDER_1;
_ LOCAL DIVIDEND, DIVISOR, NUMBER;
_ -----
_                                     + -----
_ DIVIDEND = 131313;                  | 1
_ DIVISOR = 67;                       | 2
_ NUMBER = REMAINDER(DIVIDEND, DIVISOR); | 3
_ CALL MSGLOG('THE REMAINDER IS: ' || NUMBER); | 4
_ -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ===>                                SCROLL ===> P  
THE REMAINDER IS: 60
```

REMOTELLOCATION

Returns the current value of the default remote location. (F)

Invocation `loc = REMOTELLOCATION`

<code>loc</code>	The current default remote location for distributed data processing.
------------------	--

Usage Notes Use the [SETREMOTELOC](#) shareable tool to reset the default remote location, if necessary.

See Also TIBCO Object Service Broker *Application Administration* for information on distributed data processing.

Example The following rule changes the default remote location, if it is not already the value required:

```

RULE EDITOR ==>                                SCROLL: P
CHANGE_LOCATION(VALUE);

-
- -----
- VALUE = REMOTELLOCATION;                        | Y N
- -----+-----
- CALL SETREMOTELOC(VALUE);                      | 1
- CALL ENDMSG('THE LOCATION IS ' || VALUE);      | 1 2
- -----

```

\$RESETPRINT

Resets the output arguments. (C)

Invocation `CALL $RESETPRINT(length, width, page_number, media)`

<i>length</i>	An integer specifying the number of lines per page or screen. Valid values are greater than or equal to 1 and less than or equal to 32767. Its syntax is B (binary) with length 2.
<i>width</i>	An integer specifying the number of columns per page or screen. Valid values are greater than or equal to 1 and less than or equal to 256. Its syntax is B (binary) with length 2. When <i>media</i> is set to SCR, the width must be less than or equal to 255.
<i>page_number</i>	An integer specifying the page number that appears on the subsequent page. Valid values are greater than or equal to 1 and less than or equal to 32767. If <i>page_number</i> is zero, no page number is printed. Its syntax is B (binary) with length 2.
<i>media</i>	One of: PRT – Specifies that output should be directed to the printer. SCR – Specifies that output should be directed to the screen. <i>filename</i> – Specifies that output is to be directed to the named file. " – Indicates that the media is to remain as previously set. Its syntax is C (fixed-length character string), with length 54.

- Usage Notes**
- Use [\\$SETPRINT](#) to initialize the print arguments.
 - If a data set name is specified for *media*, the data set must be pre-allocated.
 - Control characters are emitted only if *media* is set to PRT. Control characters can be included in the output to a data set if PRT is redirected to a data set (for example, by specifying a file under the Print Parameters section of the User Profile option on the workbench).

Exceptions

ROUTINEFAIL	This exception is raised for one of the following reasons: <i>length</i> – Is less than or equal to zero. <i>width</i> – Is less than or equal to zero, or is greater than 256. <i>page_number</i> – Is less than zero. <i>media</i> – Is a number or is a partitioned data set with no member specified.
--------------------	---

Example The following rule sets the print arguments using \$RESETPRINT and prints a line to the message log:

```
RESETPRINT_1;  
-  
- -----  
- CALL $RESETPRINT(5, 40, 10, 'SCR'); | 1  
- CALL $PRINTLINE('THIS LINE IS PRINTED AT THE NEW SETTINGS' | 2  
- );  
- -----  
-
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ==> SCROLL ==> P  
  
----- NEW PAGE -----  
  
Page 10  
THIS LINE IS PRINTED AT THE NEW SETTINGS
```

RESETXPARAM

Resets overrides on server parameters or on default field values set in the Table Definer. (C)

Invocation `CALL RESETXPARAM(component, entity, parm name, location)`

<i>component</i>	<p>The scope of the reset, either TABLETYPE or TABLENAME.</p> <p>TABLETYPE – indicates that the values are reset for all tables of this type.</p> <p>TABLENAME – indicates that the values are reset only for this table name.</p>
<i>entity</i>	The table type or table name, depending on component.
<i>parm name</i>	A valid parameter name as defined in @@SERVERPARMS(<i>tabletype</i>) and listed in Server Parameters and Fields below, or a valid field name defined in the Table Definer and listed in Server Parameters and Fields below.
<i>location</i>	The physical location of the table; the Data Object Broker where the external DBMS resides.

Usage Notes This function is valid only for external DBMS table types.

Server Parameters and Fields

The following are the server parameters or field values that can be reset:

Table Type	Name	Parameter or Field	Default Value	Maximum Value Length
IDM	SERVERID	Parameter	DEFAULT	8
	DBNAME	Field		8
	READY_MODE	Field	SR	2
	OPTIMIZEUPDATE	Field	N	1
	USERSUBSCHEMA	Field		8

Table Type	Name	Parameter or Field	Default Value	Maximum Value Length
IMS	SERVERID	Parameter	DEFAULT	8
	SERVERTYPE	Parameter	IMS	8
	PSBNAME	Parameter		8
	DBNAME	Field		8
	OPTIMIZEUPDATE	Field	N	1
ADA	SERVERID	Parameter	DEFAULT	8
DAT	SERVERID	Parameter	DEFAULT	8
DB2	SERVERID	Parameter	DEFAULT	8
	SERVERTYPE	Parameter	DB2	8
204	SERVERID	Parameter	DEFAULT	8
SLK	SERVERID	Parameter	DEFAULT	8

Example

The following statement resets the server ID for all tables of type IDM (IDMS/DB) to the Table Definer default value:

```
CALL RESETXPARM('TABLETYPE', 'IDM', 'SERVERID', '');
```

RETURN_CODE

Returns the return code from the last call of a TIBCO Object Service Broker external routine. (F)

Invocation `code = RETURN_CODE`

<code>code</code>	On return, contains the return code. Its syntax is B (binary) with length 4.
-------------------	--

- Usage Notes**
- RETURN_CODE should be called immediately after the routine under consideration.
 - A returned value of 0 always indicates a successful operation.

See Also *TIBCO Object Service Broker for z/OS External Environments* for information about using external routines.

RETURN_MESSAGE

Returns the system error message whenever an exception is raised. RETURN_MESSAGE is a low-level tool that must be called immediately after an exception is trapped. (F)

Invocation `string = RETURN_MESSAGE`

<code>string</code>	On return, contains the return message. Its syntax is V (variable-length character string) with length 258.
---------------------	---

- Usage Notes**
- RETURN_MESSAGE returns the system error message that is set whenever an exception is raised. It is used to obtain the system error message whenever a table access or a routine raises an exception. A routine is a low-level shareable tool that is not written in the rules language.
 - RETURN_MESSAGE should not be used to obtain a message when a tool written in rules raises an exception.
 - The system message is set to the empty string (") if the operation is successful.

Example The following rule returns the system message from the system exception GETFAIL. Notice the use of the local variable *string*. This code ensures that RETURN_MESSAGE is called immediately after the exception is trapped.

```

RETURN_MESSAGE_1;
  _ LOCAL STRING;
  _ -----
  _ GET EMPLOYEE WHERE EMPNO > 90000;
  _ -----
  _ ON ACCESSFAIL:
  _   STRING=RETURN_MESSAGE;
  _   CALL ENDMSG(STRING);

```

RETURN_SYSMMSG

Returns the last \$SYSCALL system error message when an exception is raised. RETURN_SYSMMSG is a low-level tool that must be called immediately after an exception is trapped. (F)

Invocation `string = RETURN_SYSMMSG`

<code>string</code>	On return, contains the return message. Its syntax is V (variable-length character string) with length 128.
---------------------	---

- Usage Notes**
- RETURN_SYSMMSG returns the last \$SYSCALL system error message that is set when an exception is raised. It is used to obtain the system error message when a routine raises an exception. A routine is a low-level shareable tool that is not written in the rules language.
 - RETURN_SYSMMSG should not be used to obtain a message when a tool written in rules raises an exception.
 - If the operation is successful, the system message is set to an empty string (").

Example The following rule returns the system message from the system exception ROUTINEFAIL. Notice the use of the local variable *string*. This code ensures that RETURN_SYSMMSG is called immediately after the exception is trapped.

```

RETURN_MESSAGE_1;
_  LOCAL STRING;
_  -----
_  -----+-----
_  CALL $LISTPDS(PDS, PDSBUFF, MEMBER);          | 1
_  -----
_  ON ROUTINEFAIL:
_  STRING=RETURN_SYSMMSG;
_  CALL ENDMSG(STRING);

```

RMANAGE_REQUESTS

Manages change requests for systems where the source system is remote to the target system. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	PR Remote Promotion Admin option	Press Enter
	EX Execute Rule option	Type RMANAGE_REQUEST <Enter>
	COMMAND prompt	Type EX RMANAGE_REQUEST <Enter>

When you execute the RMANAGE_REQUESTS rule, the first panel for RMANAGE_REQUESTS appears.

- Usage Notes**
- If RMANAGE_REQUESTS is executed, it must be executed with the BROWSE option set to Y.
 - From the target system you can apply change requests that are extracted from your source system. You can also back out change requests that were applied previously.

See Also *TIBCO Object Service Broker Managing Deployment* for complete information about administering the Promotion system.

ROUND

Returns the specified value rounded to the nearest integer. (F)

Invocation `number = ROUND(value)`

<i>number</i>	On return, contains the number rounded to the nearest integer. Its syntax is B (binary) with length either 4 or 12.
<i>value</i>	The number to be rounded. Its syntax can be any of the numeric syntaxes.

Exceptions

RANGERROR	Raised if the result is not between $-(2^{95})$ and $(2^{95})-1$.
------------------	--

Example The following rule rounds a number and prints the original number and the rounded number to the message log:

```
ROUND_1;
_ LOCAL SOURCE_NUM, ROUNDED_NUM;
_ -----
_                                     +-----
_ SOURCE_NUM = 1234.5678;                                     | 1
_ ROUNDED_NUM = ROUND(SOURCE_NUM);                             | 2
_ CALL MSGLOG('THE ROUNDED NUMBER IS: ' || ROUNDED_NUM);      | 3
_ CALL MSGLOG('THE SOURCE NUMBER IS STILL: ' || SOURCE_NUM);  | 4
_ -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P
THE ROUNDED NUMBER IS: 1235
THE SOURCE NUMBER IS STILL: 1234.5678
```


\$RPTIMMEDIATE

Sends the records to the output as they are read, without sorting. (C)

Invocation `CALL $RPTIMMEDIATE(reportname, media)`

<i>reportname</i>	The name of the report to be printed. Its syntax is C (fixed-length character string) with length 16.
<i>media</i>	One of: PRT – Direct output to the printer. SCR – Direct output to the screen. <i>filename</i> – Direct output to the named file. Its syntax is V (variable-length character string) with length 56.

- Usage Notes**
- The call must be placed before the first insert to the body report table.
 - Only one immediate report can be sent to an output medium at a time.
 - \$RPTIMMEDIATE is ignored if the report definition requires more than one pass over the report tables, that is, if sorting is required or if there are derived fields.
 - The call to \$RPTIMMEDIATE prepares the report for printing. It must be issued before each call to [\\$RPTPRINT](#) for which it is to apply.
 - [\\$RPTPRINT](#) must be called to send the prepared output to be printed. It turns off \$RPTIMMEDIATE processing after it completes.

Example The following rule sends a report to the message log:

```
RPTIMMEDIATE_1;
```

```

--
-- -----
-- CALL $RPTIMMEDIATE('EMPLOYEE_RPT', 'SCR');
-- FORALL EMPLOYEE:
--     EMPL_RPT.* = EMPLOYEE.*;
--     INSERT EMPL_RPT('EMPLOYEE_RPT');
--     END;
-- CALL $RPTPRINT('EMPLOYEE_RPT', 'SCR');
-- -----

```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

----- INFORMATIONAL MESSAGE LOG		

COMMAND ==>		SCROLL ==> P
Employees by Department		Page 1
LNAME	EMPNO	DEPTNO
-----	-----	-----
SMYTHE	80000	20
ROTTERDAM	80002	50
CHANG	80003	10
GARZA	80004	30
TOWNSEND	80014	70
PASTARINA	80019	50
CHESSEL	80020	30
TOWENSEND	80021	50
NAPIER	80024	20
CANON	80033	40
NELSON	81000	60
CAREY	81001	40
CHIU	81003	40
LYNGBAEK	81014	40

\$RPTOCCLIMIT

Limits the number of occurrences used to generate the report. (C)

Invocation `CALL $RPTOCCLIMIT(reportname, occlimit)`

<i>reportname</i>	The name of the report to be printed. Its syntax is C (fixed-length character string) with length 16.
-------------------	---

<i>occlimit</i>	The number of occurrences to be used by the report. Its syntax is B (binary) with length 4.
-----------------	---

- Usage Notes**
- Only the specified number of occurrences are used and the output for the report is based on this limited number of occurrences.
 - You can specify selection. The report then contains only the specified number of occurrences that meet the selection criteria.

Example The following rule uses a limited number of occurrences and sends the results to the message log:

<pre> RULE EDITOR ==> RPTOCCLIMIT_1; -- -- ----- -- CALL \$RPTOCCLIMIT('EMPLOYEE_RPT', 4); -- FORALL EMPLOYEE WHERE DEPTNO >= 30 : -- EMPL_RPT.* = EMPLOYEE.*; -- INSERT EMPL_RPT('EMPLOYEE_RPT'); -- END; -- CALL \$RPTPRINT('EMPLOYEE_RPT', 'SCR'); -- ----- </pre>	<pre> SCROLL: P 1 2 3 </pre>
--	-------------------------------

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
====NEW PAGE FOR REPORT "EMPLOYEE_RPT"=====
                                Employees by Department                                Page 1

LNAME                                EMPNO                                DEPTNO
-----

```

GARZA	80004	30
PASTARINA	80019	50
ROTTERDAM	80002	
TOWNSEND	80014	70

\$RPTOVERLAP

Designates the report tables or report fields that are not to be printed on the overlapping page of a merged report. (C)

Invocation `CALL $RPTOVERLAP(report, reporttable, reportfield, blankoverlap)`

<i>report</i>	The name of the report to be printed. Its syntax is C (fixed-length character string) with length 16.
<i>reporttable</i>	The name of the report table to which the attribute defined in the <i>blankoverlap</i> argument applies. Its syntax is C with length 16.
<i>reportfield</i>	The name of the report field on a title row to which the attribute defined in the <i>blankoverlap</i> argument applies. Its syntax is C with length 16.
<i>blankoverlap</i>	<p>This argument is used to override a <i>blankoverlap</i> attribute defined in a report table definition. It is also used to override a previous call to \$RPTOVERLAP within the existing transaction. Valid values are:</p> <p>Y – Blanks out the titles of the report table or blank out the report field on the overlap page.</p> <p>N – Keeps the report table or report field on the overlap page.</p>

- Usage Notes**
- If a report table is specified, and not a report field, the *blankoverlap* argument applies to the title rows of the entire report table.
 - Multiple calls to \$RPTOVERLAP can be made in each transaction.

Example The following example rule merges two reports, defines [\\$RPTPARMS](#) to provide an overlapping page, and deletes the report fields **\$RPTDATE** and **\$PAGE** from the resulting overlapping page:

```
RPTOVERLAP_1;
--
-- -----+-----
-- FORALL EMPLOYEE_USER('USR40'):          | 1
--   SAL_SUM2_RT.* = EMPLOYEE_USER.*;      |
--   INSERT SAL_SUM2_RT('SALARY_SUM2');    |
--   CALL COMMIT_TEST;                     |
--   END;                                   |
-- FORALL EMPLOY_DEPT_USER(10, 'USR40'):    | 2
```

```

-   DEPTNO_SALARY_RT.* = EMPLOY_DEPT_USER.*;
-   INSERT DEPTNO_SALARY_RT('DEPTNO_SALARY');
-   CALL COMMIT_TEST;
-   END;
-   CALL $RPTOVERLAP('DEPT_SALARY','DEPT_SALARY_RT',      | 3
-   '$RPTDATE', 'Y');
-   CALL $RPTOVERLAP('DEPT_SALARY','DEPT_SALARY_RT',      | 4
-   '$PAGE', 'Y');
-   CALL $RPTPRINT('SALARY_SUM2', 'SCR');
-   CALL $RPTPARMS('DEPT_SALARY', '', '', 'N', '');
-   CALL $RPTPRINT('DEPT_SALARY','SCR');
-   -----
-   ON ERROR :
-       CALL ENDMSG(GET_ERRMSG(RETURN_MESSAGE));

```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

Employee/Salary Report by Department			1
As of MAY 24/00			
Department #	Department Name	Total Salary	
50	EDUCATION	1385.00	
70	PUBLICATIONS	1370.00	
MANAGER:JOHN DUBINSKY			
DEPTNAME:EDUCATION			
NAME		SALARY	
-----		-----	
STEVENSON		700.00	
DHILLON		685.00	
DEPTNAME:PUBLICATIONS			
CROFTON		675.00	
POIRIER		695.00	
GRAND TOTAL 2765.00			
Personnel Department			
Internal Confidential			

\$RPTPARMS

Controls explicitly the physical output of a report. (C)

Invocation `CALL $RPTPARMS(reportname, length, width, eject, pagenumber)`

<i>reportname</i>	The name of the report to be printed. Its syntax is C (fixed-length character string), with length 16.
<i>length</i>	An integer specifying the physical page length, in number of lines.
<i>width</i>	An integer specifying the physical page width, in number of characters.
<i>eject</i>	Enter one of the following values: Y – Specifies that a new page should be started for this report. This applies when more than one report is printed in the same transaction. N – Specifies that a new page should not be started for this report. This applies when more than one report is printed in the same transaction.
<i>pagenumber</i>	Enter one of the following values: Positive integer – Specifies the start page number to be used if more than one report is printed in a transaction. " – Two single quotes indicate that the page numbers are to run consecutively through all the reports.

- Usage Notes**
- The output contains only the fields that fall within the parameters given. The rest of the data is truncated.
 - A number specified in *pagenumber* overrides the default page numbering.

Example The following rule prints the report as specified:

```
RPTPARMS_1;
-
- -----
- CALL $RPTPARMS('EMPLOYEE_RPT',10, 25, 'Y', '');
- FORALL EMPLOYEE:
-   EMPL_RPT.* = EMPLOYEE.*;
```

```

--      INSERT  EMPL_RPT('EMPLOYEE_RPT');
--      END;
--      CALL  $RPTPRINT('EMPLOYEE_RPT','SCR');
--      -----

```

Resulting Output

Pressing PF2 after executing the rule displays the following screen:

LNAME

CHANG
NAPIER
SMYTHE
CHESSEL
GARZA
CANON
CAREY
CHIU
LYNGBAEK
KINGSTON
PASTARINA
ROTTERDAM
TOWENSEND
NELSON
TOWNSSEND

\$RPTPRINT

Prints a report to the medium specified. (C)

Invocation `CALL $RPTPRINT(reportname, media)`

<i>reportname</i>	The name of the report to be printed. Its syntax is C (fixed-length character string) with length 16.
<i>media</i>	One of: PRT – Direct output to the printer. SCR – Direct output to the screen. <i>filename</i> – Direct output to the named file. Its syntax is V (variable-length character string) with length 56.

- Usage Notes**
- Output is printed after the transaction is completed.
 - \$RPTPRINT clears the data from the report tables used in a report after the report is printed. You must insert data into the report tables before every call to \$RPTPRINT.

Example The following rule sends a report to the message log:

```
RPTPRINT_1;
-
- -----
- FORALL EMPLOYEE:                                + 1
-   EMPL_RPT.* = EMPLOYEE.*;                        |
-   INSERT EMPL_RPT('EMPLOYEE_RPT');                |
-   END;                                              |
- CALL $RPTPRINT('EMPLOYEE_RPT','SCR');              | 2
- -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
====NEW PAGE FOR REPORT "EMPLOYEE_RPT"=====
Employees by Department                                Page 1
```

LNAME	EMPNO	DEPTNO
-----	-----	-----
CHANG	80003	10
NAPIER	80024	20
SMYTHE	80000	
CHESSEL	80020	30
GARZA	80004	
CANON	80033	40
CAREY	81001	
CHIU	81003	
LYNGBAEK	81014	

\$RPTSKIPLINES

Controls explicitly the spacing of a report. (C)

Invocation `CALL $RPTSKIPLINES(reportname, reporttable, element, linesbefore, linesafter)`

<i>reportname</i>	The name of the report that is to be printed. Its syntax is C (fixed-length character string) with length 16.
<i>reporttable</i>	The name of the report table or break table to which the spacing applies. An asterisk (*) means all report tables. Its syntax is C with length 16.
<i>element</i>	<p>The type of element to which the spacing applies. Valid values are:</p> <p>T – All title rows.</p> <p>H – All heading rows.</p> <p>R – Each occurrence.</p> <p>* – All elements in the report or report table.</p> <p>Its syntax is C with length 1.</p>
<i>linesbefore</i>	The number of lines to insert before the element. Its syntax is B (binary) with length 2.
<i>linesafter</i>	The number of lines to insert after the element. Its syntax is B with length 2.

- Usage Notes**
- If *reporttable* is specified as an empty string ("), the spacing applies to the value specified for *element*.
 - If *element* is specified as an empty string ("), the spacing applies to either before or after the entire *reporttable*. If the *element* that is specified does not exist in the report, the specification is ignored.
 - If *reporttable* and *element* are both specified as an empty string ("), the spacing applies either before or after the entire report.

Example The following rule adds spaces before and after the title and heading elements of the report table DEPT_EXPENSE_BD. A call to \$RPTSKIPLINES is made for each explicit modification to the report table.

```

      RULE EDITOR ==>>
DEPT_EXPENSE_SKP;

--
-- -----
--
-- FORALL EMPLOYEE_EXPENSE WHERE MONTH = '2000-05-01' :
--     DEPT_EXPENSE_BD.* = EMPLOYEE_EXPENSE.*;
--     DEPT_EXPENSE_BD.MONTH = '2000-05-01';
--     INSERT DEPT_EXPENSE_BD('DEPT_EXPENSE_RP');
--     END;
-- CALL $RPTSKIPLINES('DEPT_EXPENSE_RP', 'DEPT_EXPENSE_BD',
--     'T', 3, '');
-- CALL $RPTSKIPLINES('DEPT_EXPENSE_RP', 'DEPT_EXPENSE_BD',
--     'T', '', 3);
-- CALL $RPTSKIPLINES('DEPT_EXPENSE_RP', 'DEPT_EXPENSE_BD',
--     'H', '', 2);
-- CALL $RPTPRINT('DEPT_EXPENSE_RP', 'SCR');
--
-- -----

```

SCROLL: P

\$RULE_EXISTS

Checks whether a rule with the given name would be a candidate for execution. The rule can be a rule in the current search path, a TIBCO Object Service Broker routine, or an external routine with an available and executable load module. (F)

Invocation `result = $RULE_EXISTS(rulename)`

result The value returned is either:

Y – The rule exists.

N – The rule does not exist.

rulename The name of the rule or routine that you want to check.

Exceptions None.

Example The following example uses \$RULE_EXISTS to check whether a rule or routine exists and put out an appropriate message.

```
EXISTS(NAME);
- LOCAL T;
```

- \$RULE_EXISTS(NAME) = 'Y';		Y	N
- T = '';		1	
- T = 'not ';			1
- T = 'Rule or routine "' NAME '" is currently ' T		2	2
- 'available for execution.';			
- CALL ENDMSG(T);		3	3

\$RULENAME

Retrieves the name of a rule from the current execution stack. (F)

Invocation `result = $RULENAME(level, transactioncount)`

<code>result</code>	The name of a rule. This is a 16-byte identifier of type character.
<i>level</i>	<p>The rule nesting level in relation to the current rule. For example, the rule that calls \$RULENAME is at level 0. In other words, if RuleA calls RuleB and RuleB calls \$RULENAME, then, in RuleB, to learn the name of the rule that called \$RULENAME (RuleB), use a <i>level</i> value of 0; and to learn the name of the rule that called RuleB (RuleA), use a <i>level</i> value of 1 because it is 1 level higher than RuleB. A value of -1 indicates the maximum value.</p> <p>This argument is binary, two bytes long, with no decimal places.</p>
<i>transactioncount</i>	<p>The transaction nesting level in relation to the current transaction. For example, the transaction that contains the rule that calls \$RULENAME is at transaction level 0, for the purposes of this tool. In other words, if RuleA of TransactionA starts TransactionB and RuleB of TransactionB calls \$RULENAME, then, in RuleB, to learn the name of the rule that called \$RULENAME (RuleB), use a <i>transaction</i> value of 0; and to learn the name of the rule that started TransactionB (RuleA), use a <i>transaction</i> value of 1 because it is 1 level higher than TransactionB. A value of -1 indicates the maximum value.</p> <p>This argument is binary, two bytes long, with no decimal places.</p>

Usage Note \$RULENAME first locates the transaction level specified by *transactioncount* and then locates the rule specified by *level*.

Exceptions

RANGERROR	Signaled if the rule stack is exhausted before the desired <i>level</i> or <i>transactioncount</i> is reached. In our example below, a <i>transactioncount</i> of 2 with a <i>level</i> of 2 would signal a RANGERROR exception because there are not two rules in the transaction two up from the one where \$RULENAME is called.
------------------	---

Examples Rules

A new transaction starts and this is the first rule:

<div><div>RULE EDITOR ==></div><div>TOP_TOP_RULE;</div><div>—</div><div>— -----</div><div>— -----</div><div>— ...</div><div>— EXECUTE PREV_TOP_RULE;</div><div>— ...</div></div>	<div>SCROLL: P</div> <div>— -----</div> <div>— -----</div> <div>— -----+-----</div> <div>— 1</div> <div>— </div>
---	---

That first rule starts a new transaction and this is its first rule:

<div><div>RULE EDITOR ==></div><div>PREV_TOP_RULE;</div><div>—</div><div>— -----</div><div>— -----</div><div>— ...</div><div>— CALL PREV_RULE1;</div><div>— ...</div></div>	<div>SCROLL: P</div> <div>— -----</div> <div>— -----</div> <div>— -----+-----</div> <div>— 1</div> <div>— </div>
--	---

These are the second and third rules in this transaction:

<div><div>RULE EDITOR ==></div><div>PREV_RULE1;</div><div>—</div><div>— -----</div><div>— -----</div><div>— ...</div><div>— CALL PREV_RULE2;</div><div>— ...</div></div>	<div>SCROLL: P</div> <div>— -----</div> <div>— -----</div> <div>— -----+-----</div> <div>— 1</div> <div>— </div>
---	---

<div><div>RULE EDITOR ==></div><div>PREV_RULE2;</div></div>	<div>SCROLL: P</div>
--	----------------------

```
-
- -----
- -----+-----
- ... |
- EXECUTE TOP_RULE; | 1
- ... |
```

That third rule starts a third transaction and this is its first rule:

```

RULE EDITOR ==>                                SCROLL: P
TOP_RULE;

-
- -----
- -----+-----
- ... |
- CALL RNAME_CALLER; | 1
- ... |
```

These are the second and third rules of this last transaction:

```

RULE EDITOR ==>                                SCROLL: P
RNAME_CALLER;

LOCAL LEVEL, TRAN, DESC;

-
- -----
- -----+-----
- DESC = ' IS THE NAME OF THIS RULE'; | 1
- CALL RNAME(0, 0, DESC); | 2
- DESC = ' IS WHO CALLED THIS RULE'; | 3
- CALL RNAME(1, 0, DESC); | 4
- DESC = ' IS THE TOP RULE FOR THIS TRANSACTION'; | 5
- CALL RNAME(-1, 0, DESC); | 6
- DESC = ' IS THE TOP RULE FOR THE PREVIOUS TRANSACTION'; | 7
- CALL RNAME(-1, 1, DESC); | 8
- DESC = ' IS THE SECOND LAST RULE FOR THE PREV. TRANS.'; | 9
- CALL RNAME(1, 1, DESC); | A
- DESC = ' IS THE TOP RULE FOR THE TOP TRANSACTION'; | B
- CALL RNAME(-1, -1, DESC); | C
- -----
```

```

RULE EDITOR ==>                                SCROLL: P
RNAME(LEVEL, TRAN, DESC);

-
- -----
- -----+-----
- CALL MSGLOG($RULENAME(LEVEL, TRAN) || DESC); | 1
-
- ON RANGERROR :
-   CALL MSGLOG('RANGERROR LEVEL ' || LEVEL || ' ' || TRAN
-   || ' IS INVALID');
- 
```


Output

```
----- INFORMATION LOG -----
COMMAND ==>
RNAME IS THE NAME OF THIS RULE
RNAME_CALLER IS WHO CALLED THIS RULE
TOP_RULE IS THE TOP RULE FOR THIS TRANSACTION
PREV_TOP_RULE IS THE TOP RULE FOR THE PREVIOUS TRANSACTION
PREV_RULE1 IS THE SECOND LAST RULE FOR THE PREV. TRANS.
SESSMGR IS THE TOP RULE FOR THE TOP TRANSACTION
SCROLL ==> P
```

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT FIND 12=EXIT 13=PRINT 9=RECALL

Explanation

`$RULENAME(0,0)` returns the name of the rule invoking `$RULENAME`.

`$RULENAME(1,0)` returns the name of the rule calling the rule invoking `$RULENAME`. If this is issued in the first rule in a trigger, it returns the name of the rule causing the trigger to be entered.

`$RULENAME(-1,0)` returns the name of the top rule at the current transaction level.

`$RULENAME(-1,1)` returns the top rule at the previous transaction level.

`$RULENAME(1,1)` returns the first rule up the call stack for the previous transaction.

`$RULENAME(-1,-1)` returns the name of the top rule at the top transaction level.



The result of calls to `$RULENAME(-1,1)` and `$RULENAME(-1,-1)` is affected by:

- Whether the application is being run from the workbench or from a batch job
- The value of the ACTION parameter (C, E, or T)

See Also *TIBCO Object Service Broker Programming in Rules* for information about transaction processing. (Levels as used in *TIBCO Object Service Broker Programming in Rules* are different from the use of the *levels* argument for this tool.)

RULEPRINTER

Prints a rule or prints an application structure using the root rule as the base. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	PR print rules option	Type <i>rule</i> <Enter>
	EX Execute Rule option	Type RULEPRINTER <Enter>
	COMMAND prompt	Type PR <Enter>

Where:

<i>rule</i>	The name of the rule to be printed.
-------------	-------------------------------------

- Usage Notes**
- If you supply a value for the rule, the rule is sent to the printer. If you do not supply a value, the Rules Printer screen appears.
 - A print tree does not extend to the descendants of entry-level or validation rules; you must produce a separate print tree for every rule that is listed as entry-level or validation.

The Rules Printer Screen

The following sections describe the Rules Printer screen.

Rules Printer Screen Illustrated

```

                                Print Rule Utility
Library: DOCMSG2                                Wednesday, Mar 15, 2000
      Print Destination:  Hardcopy Y else Message Log
      or File:
      Print Rule List
Sel:
      Name          Unit      Modified      Print Application Structure
-----
_ BROWSE_TBL       USR40      2000-06-11      Root rule:

                                Full Doc with tree (Y/N): N
                                Detailed prt options (Y/N): N

                                Specify parameterized tables
  
```

used in indirect references
in the area at the bottom.
Use the form t(a,b)
Scroll to use as many lines
as needed.

----- Tables for Indirect References -----

PFKEYS: 2=LOGS ENTER=SEL LST 12=EXIT 3=END 4=PRT RULES 5=PRT TREE 6=PRT XREF

Header Fields

The following attributes are defined to the fields in the top portion of the screen:

LIBRARY	The library you are presently using appears here by default. To view or access another library, enter a new value to this field. To view the rules in the specified library, press Enter.
PRINT DESTINATION	<p>The destination is one of:</p> <p>Hardcopy – Y (yes) is entered by default. It can be changed to N (no) or blank if the information is to be sent to the message log or a file.</p> <p>Message Log – If hardcopy is set to N or blank and no file is provided, the output is sent to the message log.</p> <p>File – Enter a filename where the rules are to be printed. If the name in the filename field is a z/OS data set, it must be pre-allocated with the Record Format VB and LRECL of 80 or greater. The existing contents of a sequential data set are overwritten. A new member can be created for a partitioned data set.</p> <p>For Open Systems, in the filename field specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to <i>TIBCO Object Service Broker Parameters</i> for more information about this parameter.</p>

Print Rule List Section

The following attributes are defined to the fields used to print a selected listing of rules:

SEL	You can type a selection string using the field names NAME, UNIT, AUTHOR, MODIFIED, MODIFIER, UNIT and ENTRY . Pressing Enter displays the selected rules.
Rule List	You can also use the line commands D and S to delete or select the required rules.

Print Application Structure Section

The following attributes are defined to the fields used to print an application structure:

ROOT RULE	The name of the base rule for the application.
FULL DOC WITH TREE	One of: Y – Print the full documentation of the component parts of the application. N – Print only the summary documentation of the component parts.
DETAILED PRT OPTIONS	Choose N for a more concise printout or Y for a complete printout. If you choose N, the following print options are selected for you: Print Base and Descendent Rules – prints the base rule and all its descendent rules (other than entry rules). Rule Cross Reference – prints a list of the rules and the rules that call them. Local Declaration – prints a list of the variables declared and the rules that declare them. Local Cross Reference – prints a list of the variables declared and all the rules that use them. Signal Cross Reference – prints a list of the signals raised and the rules that raise them. Exception Handler – prints a list of the rules that have handlers for exceptions.

Additional Options	<p>If you choose Y, the following options are also selected for you in addition to the above options:</p> <p>Screen Definition – prints screen definitions.</p> <p>Screen Cross Reference – prints a list of the rules that display each screen.</p> <p>Table Definition – prints table definitions.</p> <p>Table Cross Reference – prints a list of the rules that access each table.</p> <p>Print Entry & Validation Rules – prints definitions of entry and validation rules (but not their descendents).</p> <p>Validation Rule Cross Reference – prints a list of rules that call the validation rules.</p> <p>List of Unresolved Names – prints a list of elements within the rules that RULEPRINTER is unable to identify as a particular object type (for example, rule, screen, or table).</p> <p>List Entry & Validation Rules – prints a list of entry and validation rules.</p> <p>Rule Calling Structure – prints a list of rules in alphabetical order with all the rules that call them.</p>
---------------------------	---

Tables for Indirect References Section

If tables are used as indirect references (that is, rules are called by placing their names in a table), type the name of the tables and their parameters in the bottom portion of the screen in the form *Table(parm1, parm2, and so on)*. The percent sign (%) can be substituted for a table name if more than one table instance is being referenced, as shown here:

```
FCNKEYS(NEW_EMPLOYEE) %(DELETE_EMPLOYEE) .
```

PF Keys

To process your selections, use the following PF keys:

PF4	Send the selected rules to the specified output medium.
PF5	<p>Produce a print tree of the rule named in ROOT RULE and send it to the specified output medium.</p> <p>A print tree is a listing of a rule and its descendant rules, along with cross-reference information on local variables, screens and tables used, exception handlers, and so on.</p>

PF6 Produce a cross-reference of the rule named in ROOT RULE and send it to the specified output medium.

A cross-reference, like a print tree, is a listing of a rule and its descendant rules, along with cross-reference information on local variables, screens and tables used, exception handlers, and so on.

Note: The cross reference lists only the rule names and does not print the rules.

S6BCALL

Invokes a TIBCO-supplied callable routine that requires a specialized environment. (C)

Invocation `CALL S6BCALL(routine, argument1, argument2, ..., argumentN)`

<i>routine</i>	The name of the TIBCO-supplied routine to be invoked. Its syntax is V (variable-length character string).
----------------	---

<i>argument1</i> , <i>argument2</i> , ..., <i>argumentN</i>	A variable number of arguments to be supplied to the routine. The number of arguments and their format is determined by the invoking routine.
---	---

Usage Notes The TIBCO Object Service Broker interface with TIBCO Enterprise Management Service (EMS) makes use of this tool. Refer to *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for more details.

Exceptions

ROUTINEFAIL	Raised if the S6BCALL fails. Refer to the associated message log to determine the cause.
CONVERSION	Raised if any argument conversion fails.
NULLVALUE	Raised if a numeric null is passed as an argument value when a numeric value is required.
STRINGSIZE	Raised if a string argument is not large enough to hold the returned data.
OVERFLOW	Raised if a numeric argument is not large enough to hold the returned data.

Example This rule invokes the TIBCO-supplied EMS routine `tibems_Sleep`.

```
CALL S6BCALL('tibems_Sleep',10);
```


S6BFUNCTION

Invokes a TIBCO supplied function that requires a specialized environment. (F)

Invocation `result = S6BFUNCTION(function, argument1, argument2, ..., argumentN)`

<code>result</code>	The value returned by the function.
<i>function</i>	A character string of syntax V containing the name of TIBCO-supplied function to be invoked.
<i>argument1</i> , <i>argument2</i> , ..., <i>argumentN</i>	A variable number of arguments to be supplied to the routine. The number of arguments and their format is determined by the invoking routine.

Usage Notes The TIBCO Object Service Broker interface with TIBCO Enterprise Management Service (EMS) makes use of this tool. Refer to *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for more details.

Exceptions

ROUTINEFAIL	Raised if the S6BFUNCTION fails. Refer to the associated message log to determine the cause.
CONVERSION	Raised if any argument conversion fails.
NULLVALUE	Raised if a numeric null is passed as an argument value when a numeric value is required.
STRINGSIZE	Raised if a string argument is not large enough to hold the returned data.
OVERFLOW	Raised if a numeric argument is not large enough to hold the returned data.

Example This rule invokes a TIBCO-supplied EMS function to an EMS server located on the specified URL. The rule creates a connection to the EMS server on `server1.yourcompany.com`.

```
RESULT = S6BFUNCTION('tibemsConnection_create', 'server1.yourcompany.com',  
CLIENTID, USER, PASSWORD);
```

S6BNOTIFY

Sends a Notification message to TIBCO Hawk. (C)

Invocation CALL S6BNOTIFY(*msgnum*, *severity*, *action*, *source*, *subsource*, *correlation*, *msgtext*)

<i>msgnum</i>	A message identifier. Its syntax is C with length 10.
<i>severity</i>	A character indicating the message severity. Its syntax is C with length 1.
<i>action</i>	A support action. Its syntax is V with length 8.
<i>source</i>	The source of the message. Its syntax is V with length 16.
<i>subsource</i>	The sub-source for the message. Its syntax is V with length 64.
<i>correlation</i>	Additional information about the message origin. Its syntax is V with length 32.
<i>msgtext</i>	Message text. Its syntax is V with length 256.

Usage Notes

- Use of this tool within TIBCO Object Service Broker for z/OS results in data being passed to a TIBCO Mainframe Service Tracker™ subsystem, which in turn makes the data available to TIBCO Hawk® through the Notification method.
- Use of this tool within TIBCO Object Service Broker for Open Systems results in the data being made available to TIBCO Hawk through the Notification method of the Hawk microagent embedded in the TIBCO Object Service Broker monitor process (osMon) or TIBCO Object Service Broker batch client (osBatch).
- NULL or a zero-length string may be specified for any parameter, which in most cases will result in the corresponding Notification method parameter being empty.
- If the source parameter is NULL or a zero-length string, then the Issuer Source parameter of the Notification message will default to the name of the Data Object Broker or Execution Environment on Open Systems, or the jobname on z/OS.
- This tool does not provide the ability to specify a value for the Host Name parameter of the Notification message. On Open Systems, it will always be set

- to the hostname for the machine on which the Execution Environment is running; on z/OS, it will always be set to the name of the LPAR.
- This tool does not provide the ability to specify a value for the Host Platform parameter of the Notification message. On all platforms it will be set to "OSB".

See Also *TIBCO Object Service Broker for z/OS Installing and Operating* for additional information on interfacing with TIBCO Mainframe Service Tracker.

TIBCO Object Service Broker for Open Systems Installing and Operating for additional information on interfacing with TIBCO Hawk.

Exceptions

ROUTINEFAIL	Raised if there is an error creating the message to be sent. Refer to the associated message log to determine the cause.
--------------------	---

Example

This rule sends a Notification message to TIBCO Hawk that a rules-based application has been successfully started.

```
CALL S6BNOTIFY('TSYSHK001', 'I', NULL, 'TSYSAPP', 'FRAMEWORK',  
$RULENAME(0,0), 'STARTED');
```

S6BTROFF

Terminates tracing initiated by the complementary shareable tool [S6BTRON](#). (C)

Invocation `CALL S6BTROFF`

Usage Notes Calling S6BTROFF without a preceding call to S6BTRON has no effect and causes no exception to be raised.

Exceptions None

Example This rule will execute the rule specified by the parameter RULE in a nested transaction, while outputting trace records to the data set or file specified by TRACEFILE.

```
TRACE(TRACEFILE, RULE);
-----
- CALL S6BTRON(TRACEFILE);                               | 1
- CALL $EXECUTE(RULE, 'N', 'N', 'L');                       | 2
- CALL S6BTROFF;                                           | 3
- -----
```

S6BTRON

Initiates tracing of rules execution in the current session. (C)

Invocation `CALL S6BTRON(data_set_or_file_name)`

<i>data_set_or_file_name</i>	A character string of syntax V (variable-length character string) containing the name of the data set or file to which trace data is to be output.
------------------------------	--

Usage Notes

- The tracing will capture the entry into, and exit from, each rule or shareable tool into a data set or file, until the invocation of the complementary shareable tool [S6BTROFF](#).
- Each entry or exit will be captured in a trace record with a record identifier, an identification of the rule or shareable tool, the library the rule was invoked from, a time-of-day, and an accumulated CPU time value of the entry or exit, and the stream level at which the entry or exit occurred.
- The value of *data_set_or_file_name* is specified as follows:
 - **z/OS data set** – Specify a fully qualified sequential data set name, for example, AAAAAA.DATA. **Note:** PDS members are not supported.
 - **Open Systems file** – Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to TIBCO Object Service Broker Parameters for more information about this parameter.
- On z/OS, the data set must be pre-allocated with record format FB (fixed-block) and logical record length of 44 bytes. The data set must be allocated with one extent and sufficient size to accommodate the expected trace output.
- On z/OS, tracing will be suspended, but not fail, if the data set fills up.
- On Open Systems, the file will be created if it does not exist. Tracing will fail if the file cannot be written to or expanded.
- The unit of time on all platforms is that of bit 0 through bit 63 of the z/OS Time-Of-Day (TOD) clock; specifically, a 64 bit integer value where conceptually a 1 is added to bit position 51 every microsecond.

- On all platforms, the trace data is output in records of 44 bytes, laid out as follows:

Offset	Size	Type	Value
0	16	Blank-padded EBCDIC character string	The name of a rule or shareable tool.
16	8	8 byte unsigned big-endian integer	The time of the rule/shareable tool entry or exit, measured from an arbitrary point in the past.
24	8	8 byte unsigned big-endian integer	On z/OS, the accumulated CPU time in the session at the time of the rule/shareable tool entry or exit. On Open Systems, the same as the preceding value.
32	2	2 byte big-endian integer	Record identifier: <ul style="list-style-type: none">254: Rule entry.255: Rule exit.252: Shareable tool entry.253: Shareable tool exit.251: TRANSFERCALL.
34	1	1 byte integer	Stream level.
35	1		Not used.
36	8	Blank-padded EBCDIC character string	The name of the library from which the rule was invoked, or blank if the record refers to a shareable tool.

Exceptions

ROUTINEFAIL	Raised if the trace data set does not exist, if tracing is already in progress, or if the trace file cannot be written to or expanded.
-------------	--

Example This rule will execute the rule specified by the parameter RULE in a nested transaction, while outputting trace records to the data set or file specified by TRACEFILE.

```
TRACE(TRACEFILE, RULE);
-----
- CALL S6BTRON(TRACEFILE);                               | 1
- CALL $EXECUTE(RULE, 'N', 'N', 'L');                     | 2
- CALL S6BTROFF;                                           | 3
- -----
```

SCREENCOL

Returns the number of columns on the user’s physical screen. (F)

Invocation number = SCREENCOL

number	On return, contains the number of columns. Its syntax is binary with length 2. For SDK sessions, the number returned is for internal use only.
--------	--

Example The following rule determines the number of columns on the screen and prints it to the message log:

```
RULE EDITOR ==>
SCREENCOL_1;
_ LOCAL NUM_COLS;
-----
_
_
_ NUM_COLS = SCREENCOL;
_ CALL MSGLOG('THE NUMBER OF COLUMNS ON THE SCREEN IS: ' ||
_   NUM_COLS);
_
_
```

Resulting Output

Pressing PF2 after executing this rule displays the following result:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>
THE NUMBER OF COLUMNS ON THE SCREEN IS: 80
SCROLL ==> P
```

SCREENMSG

Displays the given message in the message area of the specified screen. (C)

Invocation `CALL SCREENMSG(name, msg)`

<i>name</i>	The name of the screen or window. Its syntax is C (fixed-length character string) with length 16.
<i>msg</i>	A character string specifying the message. Its syntax can be C, V (variable-length character string), or W (double-byte character).

- Usage Notes**
- The message is visible the next time the screen appears.
 - The message field is located on the last row of the physical screen.
 - The message is truncated to fit the width of a screen.
 - There is no effect if the screen does not exist.
 - To set attributes for the message area of a screen, use the [\\$SETATTRIBUTE](#) tool, leaving the *table* argument blank and setting the *field* argument to @MESSAGE.
 - To set the color for the message area of a screen, use the [\\$SETCOLOUR](#) tool, leaving the *table* argument blank and setting the *field* argument to @MESSAGE.

Example The following rule sets a new screen message for the example screen and displays the screen with the new screen message:

```
SCREENMSG_1;
_
_  -----
_  -----
_  -----
_  CALL SCREENMSG('EMPLOYEE_SCR',                               | 1
_  'THIS IS THE NEW SCREEN MESSAGE');                             |
_  DISPLAY EMPLOYEE_SCR;                                           | 2
_  -----
```

Resulting Output

Executing this rule displays the following message at the bottom of the screen:

Employee Name	Employee#
-----	-----

THIS IS THE NEW SCREEN MESSAGE

SCREENROW

Returns the number of rows on the user's physical screen. (F)

Invocation `number = SCREENROW`

<code>number</code>	On return, contains the number of rows. Its syntax is B (binary) with length 2. For SDK sessions, the number returned is for internal use only.
---------------------	---

Example The following rule determines the number of rows on the screen and prints it to the message log:

```
SCREENROW_1;
_  LOCAL NUM_ROWS;
_  -----
_  -----+-----
_  NUM_ROWS = SCREENROW;                      | 1
_  CALL MSGLOG('THE NUMBER OF ROWS ON THE SCREEN IS: ' ||  | 2
_  NUM_ROWS);                                |
_  -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following message log:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE NUMBER OF ROWS ON THE SCREEN IS: 24
```

SCRIPT

Uses commands to format text from a table and store the formatted text in another table. (C)

Invocation `CALL SCRIPT(source, dest)`

<i>source</i>	The name of the table that contains the text and Script commands.
---------------	---

<i>dest</i>	The name of the table in which to write the formatted text.
-------------	---

Usage Notes Both the *source* and the *dest* tables must be defined as text tables (that is, a table that you can edit with the Text Editor, [TED](#)). The text table can be parameterized, and it must contain the following two fields:

NUMBER	A primary key field with a length of 4, a syntax of B (binary), and a semantic type of I (identifier).
---------------	--

LINE	A field with a syntax of V (variable-length character string) and a semantic type of S (string).
-------------	--

SCRIPT Commands This section lists and describes the commands you can use to control the presentation of your text. The commands are divided into three categories:

Formatting commands	Controls the format of the text.
---------------------	----------------------------------

Page layout commands	Specifies the layout of the material on the page.
----------------------	---

Convenience commands	Provides services that would require several commands to accomplish or adds extra features to the formatter.
----------------------	--



In TIBCO Object Service Broker, Script commands can be preceded either with a period or a colon.

Formatting Commands

The following commands provide basic formatting functions:

Command	Description
.br or .break	Forces a break in the formatting. A break forces whatever is currently in the line to be emitted and a new line to start immediately. Several other commands (for example, .skip) cause breaks to occur.
.bc chars	Characters specified with the .bc command become the bullet characters. You can specify up to two characters.
.bu text	Places a bullet before the text. The default bullet is ->. Specify a space (or a period) after the command and then the text that you want to appear after the bullet.
.co {on off}	Uses the option on to turn concatenation on; use the option off to specify no concatenation. Normally, concatenation is on, which specifies whether to join input lines to fit the specified line length. When concatenation is off, the input lines are placed unchanged, starting at the left margin. Input lines too long to fit are truncated. This command is useful for creating diagrams, for example, where you want the text to appear as you enter it.
.format {on off} or .fo {on off}	Uses the option on to format the input text; uses the option off to specify a pass-through mode that copies the source exactly.
.in # or .indent #	Indents the text # spaces from the left margin. An indent of 0 causes the text to begin at the left margin.
.p text or .pp text	Starts a new paragraph. The command .p is the Waterloo GML tag, and .pp is the Waterloo script primitive. To use with GML, a period '.', not a blank, must delimit the text.
.pa or .page	Places the input on a new page if the current page is not empty.
.pn {on off #}	Uses the on option to number the current page, the off option to turn page numbering off, or a number for the # variable to specify a page number.

Command	Description
<code>.para text</code>	Begins a new paragraph. At least one line is left blank and the text is indented five spaces on the first line of the new paragraph. Specify a space (or a period (.)) after the command and then the text that you want in the paragraph.
<code>.skip #</code> or <code>.sk #</code> or <code>.sp #</code>	A break occurs and # lines are skipped. If you are using double spacing, twice # lines are skipped.

Page Layout
Commands

The following commands specify the layout of material on the page. You can use them repeatedly in a document, although it is unlikely you would use the `.setup` command more than once.

Command	Description
<code>.ad #</code>	Adjusts the left margin # spaces from the edge of the page.
<code>.tm #</code>	Sets the top margin to have # lines.
<code>.bm #</code>	Sets the bottom margin to have # lines.
<code>.date 'str'</code>	Displays the specified date format. String <i>str</i> is any date format valid for \$DATE_PIC .
<code>.double</code>	Uses double spacing for the output lines.
<code>.imbed tablename</code>	Causes the specified table to be used as the source of text. When all the text is formatted, SCRIPT continues formatting with the line immediately after this command. Embedded tables can also contain <code>imbed</code> commands. The tablename can contain parameters or can be the second parameter of the <code>@TEXT</code> table.
<code>.justify {on off} or .ju {on off}</code>	Right justifies to the right margin if the option <code>on</code> is set; turns right justification off if the option <code>off</code> is specified.
<code>.ll #</code>	Sets the line length to # characters. To use this command after an initial line length is set up, it is first necessary to give the command <code>.br</code> and then the new <code>.ll</code> command. The right margin is the <i>adjust</i> plus the <i>line length</i> characters from the edge of the page.

Command	Description
.ls #	Sets the line spacing to #. The command .ls 1 is equivalent to .single and .ls 2 is equivalent to .double ; however, you can use any positive integer.
.pl #	Sets the page length to # lines. The page length must include the lines used for the top and bottom margins.
.pn {on off #}	<p>Sets the page number.</p> <p><i>on</i> – Turns page numbering on</p> <p><i>off</i> – Turns page numbering off. The special character percent sign (%) in title strings, which the setup determines, positions page numbers in the margins. Title strings that contain this character do not print if page numbering is off. This differs from Waterloo Script, which suppresses only the particular string “page %”.</p> <p><i>#</i> – Sets the number of the current page to whatever you specify for #. When the number is at the top of the page, it must be reset before the start of the new page</p>
.setup setupname	<p>Invokes a setup that specifies page details and other options. Refer to TEXTSETUP for more information on creating custom setups. TIBCO Object Service Broker provides the following setups:</p> <p><i>Default</i> is in effect whenever formatting begins and if no setup is specified. It turns formatting off and sets the page setup to suit the document handler. Therefore, SCRIPT passes through the old documentation and new documentation containing the .format command with the option on is formatted to fit the document handler screen.</p> <p><i>Screen</i> sets a larger page size, intended to fill a screen.</p> <p><i>Help</i> sets a page without a page number and is used by the Screen Definer to display both field and screen-level help.</p> <p><i>Print</i> sets a page that can print in portrait form through the use of a print function that you provide later.</p>
.single	Uses single spacing.

**Convenience
Commands**

The following commands provide services that would require several commands to accomplish or add extra features to the formatter. There are five types of convenience commands:

- Heading commands
- List commands
- The table command
- Box commands
- Text shifting commands

**Heading
Commands**

Heading commands produce consistent headings. Type a space (or a period) after any of these commands and then any text that you want in the heading. The heading commands are:

Command	Description
<code>.heading1 text</code> <code>.h1 text</code>	Makes the following text into the highest-level heading.
<code>.heading2 text</code> <code>.h2 text</code>	Makes the following text into a second-level heading.
<code>.heading3 text</code> <code>.h3 text</code>	Makes the following text into a third-level heading.

List Commands

These commands generate ordered, unordered, and definition lists. These lists can be nested.

Command	Description
<code>.list char</code>	<p>Begins a list and sets the type of the list. Depending on which variable you specify for <i>char</i>, the following lists occur:</p> <p># - Ordered list</p> <p>t - Definition list. The terms of the definition list are specified by <code>.term</code> commands and the definitions of the terms are specified by the items following each term.</p> <p>Any other characters – An unordered list using the specified characters as the bullet that marks each item</p>

Command	Description
.item text	<p>Identifies an item placed in a list. The text of the item is always indented and separated from other items by at least one blank line. Type a space (or a period) after the .item command and then the text of the item.</p> <p>After scripting, items appear as follows:</p> <p>In an ordered list, a number precedes each item.</p> <p>In an unordered list, the specified bullet character precedes each item.</p> <p>In a definition list, an item provides the definition for a preceding term, which is identified by a .term command. The item appears below and to the right of the term it defines.</p>
.term text	<p>Identifies a term In a definition list. The term cannot extend more than one line. Type a space or a period after the .term command and then the text of the term. The term is defined by the text associated with the .item command that follows it.</p>
.listend	Signifies the end of the list.

The following commands also produce ordered, unordered, and definition lists, which can be nested, and are compatible with GML if they are preceded by a colon (which is interchangeable with a period in SCRIPT), or if a period precedes the text.

Command	Description
.ol	Beginning of an ordered list.
.li text	An item for an ordered list.
.eol	End of an ordered list.
.ul	Beginning of an unordered list
.li text	An item for an unordered list
.eul	End of an unordered list

Command	Description
<code>.dl</code>	Beginning of a definition list
<code>.dt text</code>	A term for a definition list
<code>.dd text</code>	A definition of a term
<code>.edl</code>	End of a definition list

Table Command This command causes a table to print in the output.

Command	Description
<code>.table tablename field1, field2,...fieldn</code>	<p>The named table is printed into the output. The tablename can include parameters and it must be a table, not only a parameter of the @TEXT table. A list of fields of the table can follow the tablename. If this list is present, only the fields in the list are printed and appear in the order they are named. Otherwise, all the fields, up to one line, are printed. Fields that cannot fit into the line are left out.</p> <p>Note Printing a table is not the same as embedding it. This command invokes rules from the Table Printer tools (PRINTTABLE and TABLEPRINT). The table prints into the output; SCRIPT does not format it.</p>

Box Commands You use these commands to place text within a box. The box commands are often used with the concatenate commands, `.co{on|off}`, to give the user more control over the layout of text within the box.

Command	Description
<code>.box</code>	<p>This command begins a box to enclose the text that follows. This box differs from either the Waterloo Script or the GML box commands. The box is drawn as wide as the existing margins and then the margins inside the box are moved closer together so that all the text fits within it. After the completion of the box, the margins are restored. Either the colon or the period is acceptable as the first character of the command.</p> <p>Note A box can continue from one page to the next. If necessary, you can use commands, such as <code>.sk</code> (skip) and <code>.p</code> (page), to make the box fit on one page. The text within the box can contain almost any other command.</p>

Command	Description
<code>.ebox</code>	End the box. The end of the box is drawn and the margins restored to their values before the box was started. Either the colon or the period is acceptable as the first character of the command.

Text Shifting Commands

These commands specify where the text should appear on the page.

Command	Description
<code>.ce text</code>	Centers the text on the line. The text must not extend more than the line length. Type a period instead of a space before the text to use this command with Waterloo Script.
<code>.cc text</code>	Prints text centered and in uppercase. The text must not extend more than the line length. Type a period instead of a space before the text to use this command with Waterloo Script.

Example

The following rule formats text and Script commands in the @TEXT(USERNAME,SCRIPTINPUT) table and puts the formatted text in the SCRIPTOUTPUT table:

```

FORMAT_TEXT;
_
_
_ -----+-----
_ CALL SCRIPT('@TEXT(USR10,SCRIPTINPUT)', 'SCRIPTOUTPUT'); | 1
_ -----

```

Unformatted Text

The unformatted text in the @TEXT table is shown here:

```

_ :h1.Introduction to Script
_ :p.
_ You can use Script to format documents in OSB. You can access
_ Script through the SCRIPT tool or through the
_ text editor, TED.

```

Formatted Text

The formatted text in the SCRIPTOUTPUT table is shown here:

Introduction to Script

You can use Script to format documents in OSB. You can access Script through the SCRIPT tool or through the text editor, TED.

SEARCH

Searches the keyword or cross reference indexes to answer a query. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	SR search utility option	Press Enter
	EX execute rule option	Type SEARCH <Enter>
	COMMAND prompt	Type SR <Enter>

Usage Notes

- **CROSSREFSEARCH** is the callable version of this tool to be used from within a rule for a cross reference search.
- **KEYWORDSEARCH** is the callable version of this tool to be used from within a rule for a keyword search.
- Before using SEARCH, you must first build the global cross reference index. To build the index for a local library from within the SEARCH tool screen, type the name of the local library in the library field and press PF5. Your system administrator must run the REFMAKER tool in batch to build an index on the installation system libraries.
- In addition, before using SEARCH you must also use the KEYWORDMGR tool to build a keyword index. This tool indexes only keywords in the installation library.

Specifying Your Query String

You can enter more than one object when specifying your query string. Construct your strings using the following:

- Names or keywords
The wild card characters asterisk (*) and question mark (?) can be used if they are enclosed in single quotation marks.
- The AND (&) or OR (|) logical operators
- The NOT operator or the not sign
- The parentheses symbols "(" and ")"

Search Global Cross Reference Screen

Executing SEARCH displays the following screen:

Search Global Cross Reference

Feb 10, 2000

Index updated: Feb 10, 2000.

Library:

Find objects which refer to:

Rules ->

Tables(t,t.f,*.f)->

Screens ->

Reports ->

Global fields ->

Object sets ->

Libraries -> COMMON

Menus ->

Find rules which:

Raise exceptions ->

Trap exceptions(e,e t,* t)->

Declare locals ->

Search Keywords

Object type:

Keywords:

Names:

Unit:

Operators for any query are: and, &, or, x, not, ¬.

PFKEYS: ENTER=SEARCH 12=CANCEL 3=END 4=LIST KEYWORDS 9=REPEAT

This section lists the parts of the Search Global Cross Reference screen and describes the fields in each part.

Global Cross Reference

Index updated	The date when the index of the library was last updated.
Library	The library from which the global cross reference is built.

Find objects that refer to

Rules	Enter the name of one or more rules. The global cross reference searches for objects that call this rule.
-------	---

Tables	<p>Enter the name of one or more tables. The global cross reference searches for objects that refer to these tables. A special naming convention is used for tables. Tables can be named using a table name (for example, EMPLOYEE), a table name and a field name (for example, EMPLOYEE.DEPTNO), or a wild card and a field name (for example, *.DEPTNO).</p> <p>For screen tables, the search finds the screens that contain the table, as well as the objects that refer to it. For report tables, the search finds the reports that contain the table, as well as the objects that refer to it.</p> <p>If only a field name is specified, the search cross references objects that refer to the field from any table.</p>
Screens	Enter the name of one or more screens. The global cross reference searches for objects that refer to the named screens.
Reports	Enter the name of one or more reports. The global cross reference searches for objects that refer to the named reports.
Global fields	Enter the name of one or more global fields. The global cross reference searches for objects that refer to the named global fields.
Object sets	Enter the name of one or more object sets. The global cross reference searches for objects that refer to the named object sets.
Libraries	Enter the name of one or more libraries. The global cross reference searches for objects that refer to the named library.
Menus	Enter the name of one or more menus. The global cross reference searches for objects that refer to the named menus.

Find rules which

Raise exceptions	Enter the name of one or more valid exceptions. The global cross reference searches for rules that raise this exceptions.
-------------------------	---

Trap exceptions	Enter the name of one or more valid exceptions. The global cross reference searches for rules that trap it. Exceptions can be named as an exception name (for example, SYNC_ERROR), an exception name and a table (for example, GETFAIL EMPLOYEE), or a table name (for example, *EMPLOYEE). If only a table name is given, the search cross references trapped exceptions in the table.
Declare locals	<p>Enter the name of one or more local variables. The global cross reference searches for rules that declare the variables named in the query.</p> <p>If more than one of the above fields is filled in, the search is done on the first query encountered.</p>

Search Keywords

Object type	<p>Identify the type of object being sought in a keywords, names, or unit search. You can search for one object type at a time, or for ALL. Enter one of:</p> <ul style="list-style-type: none">• Global_field• Library• ObjectSet• Report• Rule• Screen• Table• All
Keywords	Enter the name of one or more keywords that you want to search for in the keyword index. The resulting display contains objects that use the specified keywords.
Names	Enter the name of the objects for which you want to search.
Unit	Enter the units of the objects for which you want to search. Menus and windows do not have a unit attribute. Therefore they do not appear on the results of an ALL search display.

The following PF keys and function keys are recognized while the Search screen appears:

Enter	Executes the SEARCH commands.
PF3	Exits the Search tool.
PF4	Lists the keywords for the library being searched. You can select from the displayed listing.
PF5	Builds an index on the specified library.
PF12	Cancels the search and exits.
PF22	Deletes an index on the specified library.

- The wild card characters asterisk (*) and question mark (?) can be used with the names or keywords being searched if they are enclosed in single quotation marks.
- After running the **SEARCH** command, you can edit or define a displayed object by entering **E** (edit) or **D** (define) on the line command and pressing Enter.
- You can create a new query by placing **Q** on the line command. This query is for all objects that reference the object on the line. After 1 or more **Q** commands, you can use PF16 to go back down the tree to the results of the previous query. After using PF16 one or more times, you can use PF14 to move up the tree again. If you use the **Q** command again, it changes the tree and all the results above are removed and replaced by the results of the most recent **Q** command.

PF3	Returns to the search panel.
PF5	Finds next occurrence.
PF9	Recalls the last primary command.
PF12	Returns to the search panel.
PF13	Prints the list.
PF14	Moves up the tree.
PF15	Returns to the workbench.
PF16	Moves back down the tree.

Example This example queries for all the objects that refer to the [REFMAKER](#) rule. The following is the result of the query:

RESULT OF QUERY			SCROLL: P
COMMAND==>			
QUERY: RULE: REFMAKER			
TYPE	NAME	UNIT	DESCRIPTION
-----	-----	-----	-----
— MENU	@ADMIN		
— MENU	@CRAMENU		
— RULE	KEYWORDMGR3	FIND	EXECUTE EACH OF THE KEYWORD UTIL

E-Edit D-Define

SEARCH_REPLACE

Replaces all occurrences of a pattern with specified characters. (F)

Invocation `new_string = SEARCH_REPLACE(input_string, replace_this, with_these, else_with_this)`

<code>new_string</code>	The character string that is returned after all the occurrences of a pattern are replaced.
<code>input_string</code>	The character string containing characters that you want to change.
<code>replace_this</code>	The pattern of characters that you want to replace.
<code>with_these</code>	A list of tokens that you want to use as replacements.
<code>else_with_this</code>	A character string that you can use to replace any other occurrences of the pattern, after the list in <i>with_these</i> is exhausted.

- Usage Notes**
- If you do not specify a value for *replace_this*, the original string (*input_string*) is returned.
 - If you do not specify a value for *with_these*, supplying only a value for *else_with_this*, all occurrences of the pattern are replaced with the *else_with_this* value.

Example The following rule removes the underscores and truncates a TIBCO Object Service Broker name to eight characters so that it is a valid name for an object outside TIBCO Object Service Broker.

```

CONVERT_NAME(NAME);
--
-- -----
--                                     +-----
-- RETURN(HEADSTRING(SEARCH_REPLACE(NAME, '_', '', ''), 8));           | 1
-- -----

```

If you execute:

```
CONVERT_NAME(EMP_INT_TABLE)
```

the returned string is:

```
EMPINTTA
```

SEARCHLIB

Searches all rules or specified rules in a library for a given string. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type SEARCHLIB <Enter>
	COMMAND prompt	Type EX SEARCHLIB <Enter>
From a rule		Type CALL SEARCHLIB

Usage Notes SEARCHLIB searches all the rules or a set of specified rules in a library for a search string and then returns the names of rules containing the string. To use SEARCHLIB, complete the following steps:

1. Enter the name of the library you are searching in.
2. Enter the token or string you are searching for.
The type of search determines what kind of search string you can enter. Refer to step 5 below for more information on search types.
You can search for an exact token or string or use the wildcard characters "?" (single character) or "*" (multiple characters) in your search string.
3. To restrict the search to a specific rule or set of rules, enter the rule name with the appropriate wildcard characters.
You can search for a specific rule or use the wildcard characters "?" (single character) or "*" (multiple characters) in your rule name.
4. To restrict the search to a specific unit or set of units, enter the unit name with the appropriate wildcard characters.
You can search for a specific unit or use the wildcard characters "?" (single character) or "*" (multiple characters) in your unit name.
5. Use the PF keys to run the appropriate search.

There are two types of searches available:

Type	PF Key	Details
Fast	PF6 or Enter	<p>Searches for a given string anywhere within a rule. A fast search works as though you entered <i>*searchstring*</i>, and therefore can return false positives (for example, entering MESSAGE as the search string could return a rule containing the token MESSAGES).</p> <p>Because a fast search searches for a string rather than a token, it can find any string, including quoted strings that include spaces. However, it cannot find any of the following:</p> <ul style="list-style-type: none">• <i>TABLENAME.FIELDNAME</i> constructions (though it can find either <i>TABLENAME</i> or <i>FIELDNAME</i> individually)• Reserved words such as CALL or EXECUTE• Numeric values
Slow	PF5	<p>Searches for the exact, complete token specified, unless wildcard characters are used.</p> <p>Because a slow search searches for a token rather than a string, you cannot use it to find extended strings containing spaces or <i>TABLENAME.FIELDNAME</i> constructions. A slow search does find reserved words such as CALL or EXECUTE, and numeric values. For large libraries, a slow search can take significantly longer than a fast search.</p> <p>Note The numeric value must be entered exactly as it appears in the Rule Editor.</p>

6. If necessary, edit or view the displayed rules.
- From the screen containing the list of returned rules, you can invoke the Rule Editor to display or edit the rules by placing your cursor on the row containing the rule name and pressing Enter or by typing any character in the command line to the left of the rule name and pressing Enter.

Example The following screen shows a slow search performed for the search string “messages” on rules named “TEST*” in library STANDARD:

```
Rule Analyser and Search Utility

Search Library: STANDARD_____ For Token/String Like: messages_____ (*/? )
In Rules Like: TEST*_____ And With A Unit Like: *_____
Case Sensitive Search N (Y/N)
```

	NAME	DATE	TIME	UNIT	MODIFIER	CREATEDATE	CREATOR
—	TEST2	2000-03-26	0859	BAD00	USR40	2000-03-26	BAD00
—	TEST3	2000-03-26	0859	BAD00	USR40	2000-03-26	BAD00

PFKEYS: 5=SLOW-SEARCH 6=FAST-SEARCH ENTER=EDIT 3=END 12=EXIT
Found 2 rules in a SLOW search

SEC_REBIND

Rebinds all security data previously bound in the Execution Environment storage.
(C)

Invocation `CALL SEC_REBIND(object, parmcatt, name)`

The three arguments are not currently used and are reserved for future enhancements. Use an empty string (' ') for the values.



If you access the Execution Environment in question through an OSB UI session, in order to pick up the rebound security data, be sure to also close and restart that OSB project after SEC_REBIND has been run.



In a multi-user environment, all users are impacted after the tool is executed. They could experience performance degradation since all bound data must be rebuilt.

Usage Notes This tool is invoked by a level 7 user to make the latest security changes effective.

Example The following example illustrates a situation where SEC_REBIND is used.

A user is initially prevented from accessing a particular table. When the user performs a GET on the table, a security fail exception results. A security administrator is required to grant the permissions to the table to the user. Since the old security data (the user being prevented from accessing the table) was bound, the user must execute a rule that calls `SEC_REBIND(' ', ' ', ' ')` to remove the old bound security data.

SECOND

Returns the second within the minute that the transaction started based on the local machine’s time zone in which the Execution Environment is running. (F)

Invocation string = SECOND

string	On return, contains the number of seconds. Its syntax is C (fixed-length character string) with length 2.
--------	---

Usage Notes The returned value is a string representing the second ("00", "01", "02", ..., "59").

Example The following rule determines the second when the transaction started and prints it to the message log:

```
SECOND_1;
_ LOCAL TIME;
_ -----
_ -----+-----
_ TIME = SECOND;                               | 1
_ CALL MSGLOG('THIS TRANSACTION WAS STARTED ' || TIME      | 2
_ || 'SECONDS AFTER THE MINUTE.');
```

Pressing PF2 after executing this rule displays the following on the screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THIS TRANSACTION WAS STARTED 25 SECONDS AFTER THE MINUTE.
```

SECURITY

Invokes the TIBCO Object Service Broker Security Manager main menu. (E)

Invocation

From the...	Move the cursor to the...	And...
Administrator's workbench	SE Security Administration option	Press Enter
	EX Execute Rule option	Type SECURITY <Enter>
	COMMAND prompt	Type SE <Enter>
Developer's workbench	SE security mgr option	Press Enter
	EX execute rule option	Type SECURITY <Enter>
	COMMAND prompt	Type SE <Enter>

Executing SECURITY displays the Security Manager main menu. You use this menu to manage objects, applications, security groups, and user IDs.

Usage Notes

You must be in browse mode to use the SECURITY tool.

See Also

TIBCO Object Service Broker Managing Security for information about the TIBCO Object Service Broker Security Manager.

SELECT_OBJ

Provides a screen that can be used to list and select objects that meet specified criteria. (C)

Invocation EXECUTE IN BROWSE SELECT_OBJ(*name*, *type*, *unit*, *author*, *library*, *location*, *children*, *subtype*)

<i>name</i>	Name of the desired object.
<i>type</i>	The object type. Valid values are: <ul style="list-style-type: none">• GLOBALFIELD• LIBRARY• MENU• OBJECTSET• REPORT• RULE• SCREEN• TABLE• WEBSERVICEPROD
<i>unit</i>	The unit to which the object belongs.
<i>author</i>	The object author.
<i>library</i>	If the object is a rule, the name of the rules library where the rule is stored.
<i>location</i>	The name of the node where the object is located.
<i>children</i>	Specifies if all objects or only the parent object should be listed. Valid values are Y (include children of objects) or N (include only parent objects).
<i>subtype</i>	Specifies if all subtype objects such as report tables should be listed and returned as subtype table (such as RPT_TABLE) or as TABLE Valid values are Y (include the subtype of the object) or N (do not include the subtype of the object).

- Executing SELECT_OBJ with null arguments displays the following screen. If values are provided for any of the arguments, they appear in the appropriate fields in the Object Selection screen and the selected objects are entered into the bottom portion of the screen. If required, you can edit the provided values in the Object Selection screen.

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

- Prompt values are predefined for the **Op** field and the **Type** field. Press PF1 to display the values for selection.
- After you specify the selection criteria and press Enter, the selected objects appear in the bottom portion of the screen. You can do a further selection on the items that you require from this screen, using the line command **s**.
- **UNLOAD** shows an example of using SELECT_OBJ from within another tool.

```

RULE EDITOR ==>
SELECT_OBJ_1;

-----
EXECUTE IN BROWSE SELECT_OBJ(' ', ' ', ' ', ' ', 'USR40',      | 1
'NODE3 ', 'Y', 'N');
-----

```

Object Selection Screen

After executing the rule, the following screen appears:

Object Selection

COMMAND ==>

Location: NODE3

Library (for RULES): USR40

Presentation Environment:

Select All: N

List Children: Y

Selection Specification

Attr	Op	Value
NAME		
TYPE	=	
UNIT		
AUTHOR		

Scroll:

Name	Type	Library	Environment	Unit
------	------	---------	-------------	------

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

@SERVERERROR

Invokes special parsing and handling of the last message, which resulted from a request to the TIBCO Object Service Broker external DBMS server. (C)

Invocation `CALL @SERVERERROR(return_message)`

<i>return_message</i>	The last message from the message stack.
-----------------------	--

Prerequisites @SERVERERROR can be invoked only in handling the SERVERERROR exception.

Usage Notes All TIBCO Service Gateway error messages have the following format:

`S6Bss###E serverid serveruid source: msg text`

<i>ss</i>	The server component (for example, IM for IMS/DB, D2 for DB2, ID for IDMS/DB).
-----------	--

<i>###</i>	The external message number.
------------	------------------------------

<i>serverid</i>	The server ID of the TIBCO Service Gateway that returned the error.
-----------------	---

<i>serveruid</i>	The server user ID (IDPREFIX + ###) of the TIBCO Service Gateway that returned the error.
------------------	---

<i>source</i>	The code portion of the server trapped the error and returned the message (for example, CSECT, rule, or function).
---------------	--

<i>msg text</i>	The actual error message.
-----------------	---------------------------

If a message from a server has some information that is required to process the error, the table-driven approach to the execution of @SERVERERROR causes a rule (specified for that error by the developer using @SERVERERROR) to execute. The error message is interpreted in the @SERVERERROR processing and put into a temporary table until required. For information on server startup parameters, refer to the *Service Gateway* manuals.

Updating Control Tables

To customize error handling, data in specific control tables must be updated:

CA-Datacom	@DATERRORCODES
CA-IDMS	@IDMSTATUSCODES
DB2	@DB2SQLMSGCNTL, @DB2CAFMSGCNTL
IMS/DB	@IMSMSGSTCNTL, @IMSMSGRCNTL

The definition of these tables is owned by TIBCO Object Service Broker and must not be modified. The data is owned by the users. The tables are used in the following manner:

1. The message identifier handlers from @SERVERMSGCNTL do a lookup in the server control tables for the external error codes.
2. If any codes are found, they call the associated user-written handler.
3. The user-written handler can use other functions and data stored in other tables to handle any external error/status code.

@SERVERERROR can be called at any time, although it is useful only for parsing external server messages generated due to external DBMS errors.

The original message can always be retrieved using @SE_MSG after @SERVERERROR is called.

The information parsed by @SERVERERROR has transaction scope.

Functions for Parsing Messages

Along with @SERVERERROR, you can use the following functions to parse the message:

@SE_MSG	Returns the entire message without modifications.
@SE_MSGEXT	Returns the external portion of the message (msg text).
@SE_MSGID	Returns the message identifier or UNKNOWN if message identifier or its prefix is not in @SERVERMSGCNTL.
@SE_MSGSOURCE	Returns the source of the message (DB2, DAT, IDM, IMS and so on) or indicates if the source is unknown.
@SE_MSGTABLE	Returns the table name where the message text was parsed (for example, @SERVERERRORDB2). Refer to the appropriate <i>TIBCO Service Gateway</i> manual for more information.
@SE_MSGHEADER (MSGID)	Returns the message identifier from the header (same as @SE_MSGID).
@SE_MSGHEADER ('MSGIDPREFIX')	Returns the message identifier prefix (first 5 characters S6Bss).
@SE_MSGHEADER (SERVERID)	Returns the server ID of the TIBCO Service Gateway that returned the message.
@SE_MSGHEADER (SERVERUSERID)	Returns the server user ID of the TIBCO Service Gateway that returned the message.
@SE_MSGHEADER (SOURCE)	Returns the code source that issued the message (CSECT, RULE, PROCEDURE, and so on).



Do not parse the non-standard part of the message format because it can change.

Exceptions

None are raised unless so arranged in the processing associated with that error for particular error situations. This error-specific processing is activated at pre-specified exit points.

Examples Handling an Error Message

The following example shows the GET_EMPLOYEE rule:

```
RULE EDITOR ==>                                SCROLL: P
GET_EMPLOYEE;

-
- -----
- -----+-----
- GET IDMS_EMPLOYEE WHERE EMP_ID = 467;          | 1
- -----
- ON SERVERERROR :
-   CALL @SERVERERROR(RETURN_MESSAGE);
- ON IDMS0966 :
-   CALL SCREENMSG(SCREEN, IDMSUSERMSG);
- ON IDMSGLOBALERROR :
-   CALL SCREENMSG(SCREEN, @SE_MSG);
```

Suppose a GET on the table IDMS_EMPLOYEE returns the following message:

S6BID034E serverid IDMS01: IDMS ERROR STATUS 0966 RECORD
ORG-DEMO-AREA

The SERVERERROR exception is raised and the @SERVERERROR rule is called. @SERVERERROR reads the following @SERVERMSGCNTL table looking for a matching entry. It scans any user-defined tables by using the \$@SERVERMSGCNTL parameter value(PRM) table first, and if no match is found it scans the @HURON parameter value table.

@SERVERMSGCNTL(@HURON)			
MSG_ID	HANDLER	MSG_TABLE	MSG_SOURCE
AD	@SERVERERRORADA	@SERVERERRORADA	ADA
AD010E	@SERVERERRORADAS	@SERVERERRORADA	ADA
AD011E	@SERVERERRORADAS	@SERVERERRORADA	ADA
DM016E	@SERVERERRORDAT	@SERVERERRORDAT	DAT
D2			DB2
D2002E	@SERVERERRRDB2CAF	@SERVERERRORDB2	DB2
D2033E	@SERVERERRRDB2SQO	@SERVERERRORDB2	DB2
D2034E	@SERVERERRRDB2SQO	@SERVERERRORDB2	DB2
D2036E	@SERVERERRRDB2SQN	@SERVERERRORDB2	DB2
IM211I	@SERVERERRORIMS2	@SERVERERRORIMS	IMS



The data for owner @HURON is owned by TIBCO Object Service Broker and must not be updated. Customers can add their own instances in @SERVERMSGCNTL provided that the OWNER specified begins with letters A to Z. The key values in their instance are message identifiers in the form ss####E mentioned above.

The ability to add user-specific entries in @SERVERMSGCNTL is provided, with the understanding that TIBCO Object Service Broker can modify the text of any message and that it is the responsibility of users to update their own handlers. It is recommended that you customize using only the specific server control tables.

Saving the Original Message

When the @SERVERERROR rule is called, it saves the original message in its own storage. This message can be retrieved at any time using function @SE_MSG when @SERVERERROR is done or within any handlers that @SERVERERROR calls through the control tables. Refer to [Functions for Parsing Messages on page 579](#).

Example 2: Parsing a Message

If an entry is found, the HANDLER associated with it is executed to parse the external message. Information from the parsed message is stored in the temporary (TEM) table specified in MSG_TABLE. The developer can use the information placed in this table to determine further action.

=====

BROWSING TABLE : @IDMSTATUSCODES

COMMAND ==>

SCROLL: P

ERROR_STATUS	SIGNALRULE	MEANINGFUL_MSG
----	-----	-----
0308	SIGNAL_IDMSERR1	Invalid record name or set name.
0966	SIGNAL_IDMSERR1	Area not available for update.
1410	SIGNAL_IDMSERR1	Attempted privacy breach. User not authorized.
1469	SIGNAL_IDMSERR1	Run unit not connected or connection broken.
1474	SIGNAL_IDMSERR1	Invalid DMCL, subschema or procedure name.

=====

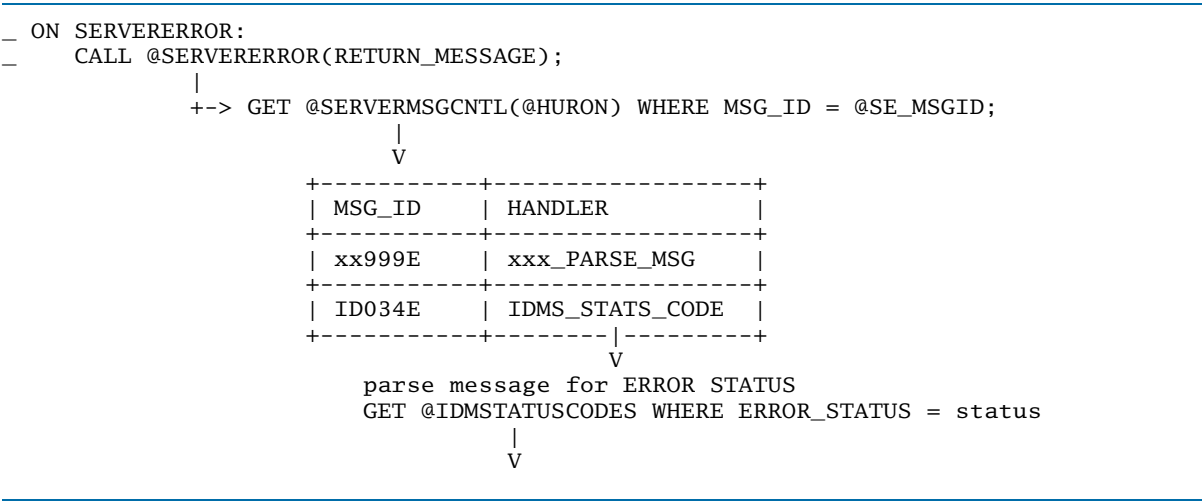
In this example, the message ID034E is found and the IDMS_STATUS_CODE rule is executed to parse the message looking for the CA-IDMS status code. Upon finding the status code, the @IDMSTATUSCODES table is searched using the status code. If an entry is found the SIGNALRULE is executed. The SIGNALRULE, SIGNAL_IDMSERR1, sets the appropriate signal depending on the status code:

=====	
RULE EDITOR ==>	SCROLL: P
SIGNAL_IDMSERR1;	

IDMSSTATUSCODE = 966;	Y N N N N
IDMSSTATUSCODE = 1469;	Y N N N
IDMSSTATUSCODE = 1474;	Y N N
IDMSSTATUSCODE = 1410;	Y N

SIGNAL IDMS0966;	1
SIGNAL IDMS1469;	1
SIGNAL IDMS1474;	1
SIGNAL IDMS1410;	1
SIGNAL IDMSGLOBALERROR;	1

Overview of Process Flow



@SESSION

Alters session-related items maintained by this table. (TBL)

Table Definition

Parameters This table has two parameters: INSTANCE and LOCATION.

Parameters	Typ	Syn	Len	Description
INSTANCE		B	4	Must be 0.
LOCATION	I	C	16	Location of @SESSION table (remote system).



The INSTANCE parameter must be set to 0.

Fields @SESSION has only a single row. Fields marked with an asterisk (*) can be changed subject to validity checking. All other fields, including those marked RESERVED, must not be changed.

Field Name	Typ	Syn	Len	Dec	Key	Ord	Rq	Description
ADDRESS	I	B	4	0	P			Reserved.
USERID	I	C	8	0				The current user ID.
SEARCH	S	C	1	0				The library search path for the current transaction.
GROUP *	I	C	16	0				The current security group.
REMOTELOCATION *	I	C	16	0				The default remote location.
HOMELOCATION	I	C	16	0				The name of this TIBCO Object Service Broker system.
TERMINAL	I	C	16	0				The user's terminal name.
STORE	I	C	16	0				Reserved.
TESTLOCATION	I	C	16	0				Reserved.

Field Name	Typ	Syn	Len	Dec	Key	Ord	Rq	Description
COMMHANDLE *	I	B	4	0				Address of a communications area.
COMMLENGTH *	C	B	2	0				Length of the communications area.
COMMFREEHANDLE	I	B	4	0				Address of free space in the communications area.
COMMFREENLENGTH	C	B	2	0				Length of free space in the communications area.
SEG0INHANDLE	I	B	4	0				Address of IMS input segment 0.
SEG0INLENGTH	C	B	2	0				Length of IMS input segment 0.
SEG1INHANDLE	I	B	4	0				Address of IMS input segment 1.
SEG1INLENGTH	C	B	2	0				Length of IMS input segment 1.
SEG2INHANDLE	I	B	4	0				Address of IMS input segment 2.
SEG2INLENGTH	C	B	2	0				Length of IMS output segment 2.
SEG0OUTHANDLE *	I	B	4	0				Address of IMS output segment 0.
SEG0OUTLENGTH *	C	B	2	0				Length of IMS output segment 0.
SEG1OUTHANDLE *	I	B	4	0				Address of IMS output segment 1.
SEG1OUTLENGTH *	C	B	2	0				Length of IMS output segment 1.
SEG2OUTHANDLE *	I	B	4	0				Address of IMS output segment 2.
SEG2OUTLENGTH *	C	B	2	0				Length of IMS output segment 2.

Field Name	Typ	Syn	Len	Dec	Key	Ord	Rq	Description
APIINHANDLE	I	B	4	0				Address of dataIn commarea (Call Level Interface, SDK (C/C++), and SDK (Java) only).
APIOUTHANDLE	I	B	4	0				Address of dataOut commarea (Call Level Interface, SDK (C/C++), and SDK (Java) only).
PLATFORM	S	C	1	0				The platform where your TIBCO Object Service Broker system is running: <ul style="list-style-type: none"> • N-Windows • S-Solaris • V-z/OS
RELEASE	S	C	16	0				The release number of the TIBCO Object Service Broker system you are running.

See Also *TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for information about the Call Level Interface, the SDK (C/C++), and the SDK (Java).

- Usage Notes**
- Supported operations are: GET and FORALL (which both retrieve the single row), and REPLACE (which replaces the single row, subject to a number of validity tests).
 - Not supported: the selection operation, and parameter value (PRM) tables and subview (SUB) tables.
 - @SESSION behaves like a session table in that its content and effects are local to one TIBCO Object Service Broker session and are not seen by other users, even if they are sharing a single Execution Environment.
 - Fields **COMMHANDLE** and **COMMLENGTH** are primarily for use in the z/OS CICS environment; however, they could be provided in other environments as well. These fields can be replaced by the user (subject to the restriction that the **COMMHANDLE** value must be the value of field **ADDRESS** of a row in System Interpreted Table @MAP) with **SCOPE** of **ENVIRONMENT**, and **COMMLENGTH** must not exceed the value of **SIZE** in the same row of @MAP.

- The IMS-specific fields contain meaningful information in a z/OS IMS environment only.

Example @SESSION_SAMPLE1 Rule

The @SESSION_SAMPLE1 rule changes the TIBCO Object Service Broker security group for the current session to NEWGROUP. The user must have security authorization to make this change; otherwise, the SECURITYFAIL exception is raised.

```
@SESSION_SAMPLE1;
-
- -----
- -----+-----
- GET @SESSION(0);                                | 1
- @SESSION.GROUP='NEWGROUP';                      | 2
- REPLACE @SESSION(0);                            | 3
- -----
```

@SESSION_SAMPLE2 Rule

The @SESSION_SAMPLE2 rule uses [@MAP](#) to allocate and register environment storage for a new commarea of 1000 bytes. The address of the new commarea is then placed in the **COMMHANDLE** field of @SESSION and the length of the storage area is placed in field **COMMLENGTH**.

If an external program is subsequently called, the address and length of the COMMAREA are passed to it. The COMMAREA can be manipulated using MAP tables. If the address placed in @SESSION.COMMHANDLE is not that of a valid block of environment storage, the DATAREFERENCE exception is raised.

```
@SESSION_SAMPLE2;
-
- -----
- -----+-----
- @MAP.ADDRESS = 0;                                | 1
- @MAP.SIZE = 1000;                                | 2
- INSERT @MAP('ENVIRONMENT');                      | 3
- GET @SESSION(0);                                | 4
- @SESSION.COMMHANDLE = @MAP.ADDRESS;                | 5
- @SESSION.COMMLENGTH = @MAP.SIZE;                  | 6
- REPLACE @SESSION(0);                            | 7
- -----
```

@SESSIONCOUNTS

Obtains information on events occurring within the Execution Environment during a session. (TBL)

Table Definition

Parameters This table requires a numeric value to be specified as a parameter. The value can be any valid number between 0 and 999999999.

Fields Fields in the @SESSIONCOUNTS table include:

ADDRESS	Reserved field.
USERID	The session user ID.
LOCALMESSAGES	Number of messages sent to the local node.
LOCALDELTA	Increment in local message traffic since the last access to @SESSIONCOUNTS.
REMOTEMESSAGES	Number of messages sent to a remote node.
REMOTEDELTA	Increment in remote message traffic since the last access to @SESSIONCOUNTS.
TABLECALLDELTA	Increment in the number of table calls since the last access to @SESSIONCOUNTS.
CPUSECONDS	Address space (TCB) time. This value is obtained from the z/OS ASCB structure, and is generally reliable within a range of plus/minus 3 percent.
CPUDELT	Increment in address space time since the last access to @SESSIONCOUNTS.
SYSTEMSECONDS	SRB time. This value is obtained from the z/OS ASCB structure and is generally reliable within a range of plus/minus 3 percent.

SYSTEMDELTA	Increment in SYSTEMSECONDS time since the last access to @SESSIONCOUNTS.
RESERVEDON	Reserved field.
RESERVEDONDELTA	Reserved field.
RESERVEDOFF	Reserved field.
RESERVEDOFFDELTA	Reserved field.
STORAGENOW	Reserved field.
STORAGENOWDELTA	Reserved field.
STORAGEHWM	Reserved field.
STORAGEHWMDELTA	Reserved field.
STORAGEOCCUPANCY	Reserved field.
STORAGEOCCDELTA	Reserved field.



CPUSECONDS & CPUDELTA represent address space time information and should not be referenced in a shared space environment such as z/OS CICS.

Usage Notes

- This facility is available only on a z/OS system.
- All the values are SESSION related, not transaction related.
- To view data in @SESSIONCOUNTS use the Table Browser, or to process data in @SESSIONCOUNTS, do a GET from a rule and use the fields accordingly.
- @SESSIONCOUNTS can be used to view CPU usage and message traffic.
- It can be used to examine the effects of rule changes when optimizing applications.
- It is usually browsed with the Table Browser.
- It can also be useful for performance tests in a rule.

Example Fields whose names end in DELTA are the difference between the value of the correspondingly named field now, and its value the last time the table was read.

Suppose a user is interested in how many table accesses are done in the course of creating a new employee record. The transaction they would normally run would start by calling CREATE_EMPLOYEE from a menu. To determine our count, the user could write the following rule:

```
CHECK_#ACCESSES;
```

```
-----+-----  
GET @SESSIONCOUNTS(0); | 1  
CALL CREATE_EMPLOYEE('111'); | 2  
GET @SESSIONCOUNTS(0); | 3  
TABLECALLS = @SESSIONCOUNTS.TABLECALLDELTA; | 4
```

This obtains the count of table calls done between the GETs on @SESSIONCOUNTS. Since CREATE_EMPLOYEE is the only process called in between, this gives the count.

SESSMGR

Displays the login interface to the Session Manager (workbench).

Invocation If a customized workbench is not provided, the STANDARD screen appears when you first log in. Refer to the *TIBCO Object Service Broker Defining Screens and Menus* manual for information on customizing a session manager (workbench) screen.

The Session Manager

The following two illustrations show this screen:

```

DOCMSG TEST: N BROWSE: N 1:33 AM THURSDAY MAR 15 2007

ER edit rule      ==>
EX execute rule   ==>
DB debug rule     ==>
BR browse table   ==>
ED edit table     ==>

OS object set     ==>
DS define screen  ==>
DR define report  ==>
DT define table   ==>
DL define library ==>
GR generate rpt   ==>

COMMAND ==>  __

PFKEYS: 2=LOGS 3=EXIT 12=EXIT

Top of Window

```

Press PF8 to scroll down the screen:

DOCMSG TEST: N BROWSE: N 1:33 AM THURSDAY MAR 15 2007									
CD copy defn	==>	SU MO TU WE TH FR SA							
CT copy table	==>	1 2 3							
CL clear table	==>	4 5 6 7 8 9 10							
DD diff defn	==>	11 12 13 14 15 16 17							
PR print rules	==>	18 19 20 21 22 23 24							
PT print table	==>	25 26 27 28 29 30 31							

```
SR search utility ==>
SE security mgr    ==>
PB problem rpt    ==>
PM promotion       ==>
MR manage rights   ==>
UP user profile    ==>

COMMAND ==>  ____
```

```
PFKEYS: 2=LOGS 3=EXIT 12=EXIT
```

```
1:33:51 Engine promotion 3217 (V500E040) 2007-03-14 at 20:44:18
```

Fields The screen contain the following fields:

Library	The Library field is on the title line for the workbench, and is not labeled. It displays the name of your local library. You can change this to another local library name. In the example above, it displays USR40.
TEST	<p>Displays Y or N. If you change this to Y, a rule executed from the EX execute rule option of the workbench is run in test mode, unless the mode is changed within the rule. When the transaction ends or a COMMIT is issued, the updates to tables are discarded.</p> <p>Only the execute rule workbench tool uses the test mode set on the workbench. When the TEST field is set to Y, you can insert occurrences in a table using the Table Editor. If you insert occurrences using a rule, the rule could run successfully, occurrences no longer existing in the table after the rule ends.</p>

BROWSE	<p>Displays Y or N. If this is changed to Y, a rule executed from the EX execute rule field of the workbench is run in browse mode and you <i>cannot</i> make updates to tables through the execution of rules, unless the mode is changed within the rule. You can change the mode within the rule by starting a new transaction and specifying a different mode.</p> <p>Only the execute rule workbench tool uses the browse mode set on the workbench. When the BROWSE field is set to Y, you can insert occurrences in a table using the Table Editor. If you insert occurrences through a rule, the rule fails.</p> <p>Other tools, such as the Menu Definer, give you the option of specifying the mode in which your rule executes. This specification overrides the workbench specification.</p>
DATE	<p>This field displays the current date.</p>
COMMAND	<p>Enter any command abbreviation and optionally, the parameters.</p>
Command History	<p>The command history area is scrollable and appears below the COMMAND field. The list can display the last 50 commands issued. To scroll the command history area, move the cursor into the field and press PF7 and PF8.</p>
AppointmentBook /Tickler	<p>The Appointment Book/Tickler is invoked by placing the cursor on any day in the calendar and pressing Enter. It provides an hourly appointment schedule, a to-do list, and reminders. The appointment book can also be entered for the current day by placing the cursor on any field in the reminder area and pressing Enter.</p>
Broadcast Message	<p>The Broadcast Message appears when you log in to TIBCO Object Service Broker. It appears at the bottom of the workbench next to the time. The message is the first occurrence of the BROADCAST table. You can view the rest of the BROADCAST table by pressing PF2 before you start any other transactions in this TIBCO Object Service Broker session. If the table is empty, the message "No broadcast available" appears.</p>

Accessing the Workbench Tools

The TIBCO Object Service Broker tools listed in the menu can be accessed in three ways:

- From the menu
- From the command line in the middle of the screen
- From the command history in the lower portion of the screen

To issue a command, place the cursor on the corresponding line for the command. The menu can be scrolled to display all the commands. Most commands have a required or optional argument that should be typed in the corresponding field. If the name of an object (for example, rule, table, or screen) is not typed, a list of the available objects appears.

Rule arguments or table parameters can be supplied with the rule or table names. If the necessary arguments or parameters are not supplied, a prompt screen appears.

The pair of letters to the left of each command line is an abbreviation for that command. They are used with the **COMMAND** and **Command History** fields.

A previously issued command can be executed again by placing the cursor on it in the command history area and pressing Enter. The abbreviation or the argument can be changed by typing over it before execution.

See Also For information about:

This option...	Refer to this manual...
ER edit rule	<i>TIBCO Object Service Broker Programming in Rules.</i>
EX execute rule	<i>TIBCO Object Service Broker Programming in Rules.</i>
DB debug rule	<i>TIBCO Object Service Broker Programming in Rules.</i>
BR browse table	<i>TIBCO Object Service Broker Managing Data.</i>
ED edit table	<i>TIBCO Object Service Broker Managing Data.</i>
DS define screen	<i>TIBCO Object Service Broker Defining Screens and Menus.</i>
DR define report	<i>TIBCO Object Service Broker Defining Reports.</i>
DT define table	<i>TIBCO Object Service Broker Managing Data.</i>
DL define library	<i>TIBCO Object Service Broker Programming in Rules.</i>

This option...	Refer to this manual...
GR generate report	<i>TIBCO Object Service Broker Defining Reports.</i>
PM promotion	<i>TIBCO Object Service Broker Managing Deployment.</i>
UP user profile	<i>TIBCO Object Service Broker Managing Security.</i>

The rest of the TIBCO Object Service Broker tools are documented in this manual.

PF Keys

The following function keys are recognized while the screen appears:

PF1	Provides online help related to the session menu.
PF2	Displays a history of broadcast messages when you first log in to TIBCO Object Service Broker. After you use any of the workbench tools, this PF key displays the message log if any information is there.
PF3	Terminates the current TIBCO Object Service Broker session.
PF7	Scrolls the command history or command menu up.
PF8	Scrolls the command history or command menu down.

Appointment Book

Pressing Enter while the cursor is positioned on a day number in the calendar displays a screen similar to the following screen:

APRIL		MAY		JUNE
SU MO TU WE TH FR SA	SU MO TU WE TH FR SA	SU MO TU WE TH FR SA	SU MO TU WE TH FR SA	
2 3 4 5 6 7 8	7 8 9 10 11 12 13	4 5 6 7 8 9 10		
9 10 11 12 13 14 15	14 15 16 17 18 19 20	11 12 13 14 15 16 17		
16 17 18 19 20 21 22	21 22 23 24 25 26 27	18 19 20 21 22 23 24		
23 24 25 26 27 28 29	28 29 30 31	25 26 27 28 29 30		
30 31				
APPOINTMENTS:	FRIDAY	MAY	19	2000
8:00				TO DO:
9:00				
10:00				
11:00				
12:00				
1:00				

2:00
3:00
4:00
5:00
EVE:
EVE:

|
|
|
|
|
|

COMMAND ==>
Go to month: __ day#: __ year: ____

Fields	<p>The appointment calendar contains the following fields:</p> <table> <tr> <td>APPOINTMENTS</td><td>Type appointments in the appropriate time slot.</td></tr> <tr> <td>TO DO</td><td>Type things to do.</td></tr> </table>	APPOINTMENTS	Type appointments in the appropriate time slot.	TO DO	Type things to do.										
APPOINTMENTS	Type appointments in the appropriate time slot.														
TO DO	Type things to do.														
Primary Commands	<p>The following primary commands are recognized while the screen appears:</p> <table> <tr> <td>N or Next</td><td>Saves the present page and displays the next page.</td></tr> <tr> <td>P or Previous</td><td>Saves the present page and displays the previous page.</td></tr> <tr> <td>N # or Next #</td><td>Saves the current page and moves ahead # pages.</td></tr> <tr> <td>P # or Previous #</td><td>Saves the current page and moves back # pages.</td></tr> <tr> <td>N ? or Next ?</td><td>Saves the current page and moves ahead to the next non-blank page.</td></tr> <tr> <td>P ? or Previous ?</td><td>Saves the current page and moves back to the previous non blank page.</td></tr> <tr> <td>N s string or Next s string</td><td> <p>Saves the current page and moves ahead to the next page that contains the string <i>string</i>.</p> <p>For example:</p> <p>COMMAND==> n s meeting</p> <p>The search is sensitive to case and detects the string when it is a substring.</p> </td></tr> </table>	N or Next	Saves the present page and displays the next page.	P or Previous	Saves the present page and displays the previous page.	N # or Next #	Saves the current page and moves ahead # pages.	P # or Previous #	Saves the current page and moves back # pages.	N ? or Next ?	Saves the current page and moves ahead to the next non-blank page.	P ? or Previous ?	Saves the current page and moves back to the previous non blank page.	N s string or Next s string	<p>Saves the current page and moves ahead to the next page that contains the string <i>string</i>.</p> <p>For example:</p> <p>COMMAND==> n s meeting</p> <p>The search is sensitive to case and detects the string when it is a substring.</p>
N or Next	Saves the present page and displays the next page.														
P or Previous	Saves the present page and displays the previous page.														
N # or Next #	Saves the current page and moves ahead # pages.														
P # or Previous #	Saves the current page and moves back # pages.														
N ? or Next ?	Saves the current page and moves ahead to the next non-blank page.														
P ? or Previous ?	Saves the current page and moves back to the previous non blank page.														
N s string or Next s string	<p>Saves the current page and moves ahead to the next page that contains the string <i>string</i>.</p> <p>For example:</p> <p>COMMAND==> n s meeting</p> <p>The search is sensitive to case and detects the string when it is a substring.</p>														

P s string or Previous s string	<p>Saves the current page and moves back to the previous page that contains the string <i>string</i>.</p> <p>For example:</p> <p>COMMAND==> p s meeting</p> <p>The search is sensitive to case and detects the string when it is a substring.</p>
--	--

\$SETATTRIBUTE

Sets attributes for the field of the screen table, in the specified screen. (C)

Invocation CALL \$SETATTRIBUTE(*screen*, *table*, *field*, *attribute*, *flag*)

<i>screen</i>	A character string specifying the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	A character string specifying the screen table. Its syntax is C with length 16. To set attributes for the message area of the specified screen (where the SCREENMSG tool writes), leave <i>table</i> blank.
<i>field</i>	A character string specifying the screen field. Its syntax is C with length 16. To set attributes for the message area of the specified screen (where the SCREENMSG tool writes), set <i>field</i> to @MESSAGE.
<i>attribute</i>	<p>The following attributes are available:</p> <p>P – Protects the field.</p> <p>H – Highlights the field.</p> <p>V – Makes the field visible.</p> <p>D – Makes the field light pen detectable. Makes the data mapped graphical object detectable by cursor (allow focus).</p> <p>N – Accepts only numeric characters in the field.</p> <p>U – Underlines the field (extended attribute).</p> <p>R – Reverses the foreground and background colors for the field (extended attribute).</p> <p>B – Makes the field blink (extended attribute).</p>
<i>flag</i>	<p>One of the following:</p> <p>Y – Turns the attribute on.</p> <p>N or anything except Y – Turns the attribute off.</p>

- Usage Notes**
- Not all display devices can support the extended attributes. The extended attributes are supported only on some 3270 terminals on a z/OS system.
 - Valid values must be supplied for *all* the arguments or an error occurs.
 - \$SETATTRIBUTE must be called before the screen appears or it is ignored.
 - \$SETATTRIBUTE operates only for the duration of the transaction.

- \$SETATTRIBUTE operates only on a screen table that has real occurrences, and has the current position set by a table access. If the current position is not set in the screen table, a call to \$SETATTRIBUTE is ignored.
- For a particular screen field, specify only one of the extended attributes U, R, or B.

Example The following rule sets the reverse video attribute on the **FCNKEYS** field of the example screen and displays the screen:

```
SETATTRIBUTE_1;
-
- -----
- -----+-----
- FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');           | 1
- INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                           | 2
- CALL $SETATTRIBUTE('NEW_EMPLOYEE', 'FCNKEY_SPECS',            | 3
-   'FCNKEYS', 'R', 'Y');                                         |
- UNTIL EXIT_DISPLAY DISPLAY NEW_EMPLOYEE:                       | 4
-   CALL PROCESS_FCNKEY('NEW_EMPLOYEE');                         |
-   END;                                                         |
- -----
```

\$SETCHANNEL

Nominates a channel for passing to a a program or transaction. (C)

Invocation `CALL $SETCHANNEL(channel)`, where *channel* is the name (1-16 characters) of the channel to pass to the program that is being executed by LINK as an external CICS routine; or by XCTL or START TRANSID as a result of the SESSIONEND action.

Usage Notes A blank channel name denotes that no channel is to be passed.

\$SETCOLOUR

Sets the color of a screen field. (C)

Invocation CALL \$SETCOLOUR(*screen*, *table*, *field*, *color_type*, *color*)

<i>screen</i>	A character string specifying the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	A character string specifying the screen table. Its syntax is C with length 16. To set the color for the message area of the specified screen (where the SCREENMSG tool writes), leave <i>table</i> blank.
<i>field</i>	A character string specifying the screen field. Its syntax is C with length 16. To set the color for the message area of the specified screen (where the SCREENMSG tool writes), set <i>field</i> to @MESSAGE.
<i>color_type</i>	A character string specifying whether the color is to be foreground or background. Valid values: F – Foreground. B – Background.
<i>color</i>	A character string specifying the color to be used. Valid values are any color in the @COLOURS table. Its syntax is C with length 25.

- Usage Notes**
- *Not all display devices can support background color.* If your display device does not support background color, the specification is ignored.
 - If your display device does not support the color specified, a default color is substituted.
 - Unexpected results can occur when an extended color is sent to a terminal that does not support extended colors. Generally, if a field is specified with an extended color and sent to a terminal that does not support extended colors, the terminal software returns the entire screen in two-color mode.
 - You must supply valid values for *all* the arguments or an error occurs.
 - \$SETCOLOUR takes effect on the next DISPLAY.
 - \$SETCOLOUR operates only for the duration of the transaction.

- \$SETCOLOUR operates only on a screen table that has real occurrences and has the current position set by a table access. If the current position is not set in the screen table, a call to \$SETCOLOUR is ignored.

Example The following rule sets the color on the **FCNKEYS** field of the example screen and displays the screen:

```
SETCOLOUR_1;
-
- -----
- -----+-----
- FCNKEY_SPECS.FCNKEYS = FCNKEY_MSG('NEW_EMPLOYEE');      | 1
- INSERT FCNKEY_SPECS('NEW_EMPLOYEE');                      | 2
- CALL $SETCOLOUR('NEW_EMPLOYEE', 'FCNKEY_SPECS', 'FCNKEYS', | 3
-   'F', 'RED');
- UNTIL EXIT_DISPLAY DISPLAY NEW_EMPLOYEE:                  | 4
-   CALL PROCESS_FCNKEY('NEW_EMPLOYEE');
-   END;
- -----
```

SETCURSOR

Positions the cursor in the field of the screen table, in the specified screen. (C)

Invocation `CALL SETCURSOR(screen, table, field)`

<i>screen</i>	The name of the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	The name of the screen table within the screen. Its syntax is C with length 16.
<i>field</i>	The name of the screen field within the screen table. Its syntax is C with length 16.

- Usage Notes**
- If you supply valid values for *screen*, *table*, and *field*, the cursor is positioned on the current occurrence of the field when the screen appears. If no current position is established, the call is ignored.
 - If the value for *screen*, *table*, or *field* is invalid, an error occurs.
 - If *table* or *field* are not specified, the call to SETCURSOR is ignored.
 - The screen server determines the type of terminal you are using (for example, Mod 5) and if any screen table is placed beyond the screen boundary, it treats it as an invisible table that cannot be viewed. Since the screen table is not displayable, calls to SETCURSOR are ignored.

Example The following rule positions the cursor on the EMPLOYEE# field of the example screen and displays the screen below:

```
SETCURSOR_1;
- -----
- -----+-----
- EMPLOYEE_DATA.EMPNO=80000;                      | 1
- INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');            | 2
- CALL SETCURSOR('EMPLOYEE_SCR', 'EMPLOYEE_DATA', 'EMPNO'); | 3
- DISPLAY EMPLOYEE_SCR;                             | 4
- -----
```

Resulting Output

Executing this rule displays the following screen:

Employee Name	Employee#
-----	-----
	80000

SETCURSOR_POS

Positions the cursor in the column of the field of the occurrence, in the screen table of the screen. (C)

Invocation `CALL SETCURSOR_POS(screen, table, field, occurrence_number, column_offset)`

<i>screen</i>	The name of the screen. Its syntax is C (fixed-length character string) with length 16.
<i>table</i>	The name of the screen table within the screen. Its syntax is C with length 16.
<i>field</i>	The name of the screen field within the screen table. Its syntax is C with length 16.
<i>occurrence_number</i>	An integer specifying the occurrence within the screen table. Its syntax is B (binary) with length 2.
<i>column_offset</i>	An integer specifying the relative column number. Its syntax is B (binary) with length 2. The first column is column 1.

Usage Notes

- When you supply values for *screen*, *table*, and *field*, the cursor is positioned on the *field* in the *table* at the specified occurrence and column offset. If the values do not exist, an error occurs.
- The set of occurrences used by SETCURSOR_POS is the set of real and empty occurrences. For example, if *occurrence_number* is set to 5 and there are three real occurrences and two empty occurrences displayed, the cursor is positioned on the second empty occurrence.

As occurrences are inserted into the screen table, the number of empty occurrences is decreased by the number of occurrences inserted. The number of empty occurrences is reset to the value on the screen definition each time the screen appears. For example, if a screen definition has EMPTY OCCS set to 5 and three occurrences are inserted into the screen table, the number of empty occurrences equals 2.

If the screen appears again, the number of empty occurrences is reset to 5. Therefore, if SETCURSOR_POS is called prior to the display, the maximum valid value for *occurrence_number* is 5 (three real occurrences plus two empty occurrences). After the display, the maximum valid value for *occurrence_number* is 8 (three real occurrences plus five empty occurrences).

- If you do not specify *table* or *field*, the call to SETCURSOR_POS is ignored.
- SETCURSOR_POS takes effect on the next DISPLAY.
- If *occurrence_number* or *column_offset* are invalid, the call to SETCURSOR_POS is ignored.

Example The following rule fills the example screen with data from the example table, positions the cursor in column 1 in the first occurrence of the **LNAME** field, and displays the screen. The cursor is positioned on the first S in Smythe.

```
SETCURSOR_POS_1;
```

```

_
_  -----
_                                     +-----
_  FORALL EMPLOYEE :                               | 1
_      EMPLOYEE_DATA.* = EMPLOYEE.*;              |
_      INSERT EMPLOYEE_DATA('EMPLOYEE_SCR');      |
_      END;                                         |
_  CALL SETCURSOR_POS('EMPLOYEE_SCR', 'EMPLOYEE_DATA', | 2
_      'LNAME', 1, 1);                             |
_  DISPLAY EMPLOYEE_SCR;                           | 3
_  -----
_
```

Resulting Output

Executing this rule displays the following screen:

Employee Name	Employee#
-----	-----
SMYTHE	80000
ROTTERDAM	80002
CHANG	80003
GARZA	80004
TOWNSEND	80014
PASTARINA	80019

\$SETENVCOMMAREA

Passes data from TIBCO Object Service Broker into a calling environment that is not TIBCO Object Service Broker. (F)

Invocation `length = $SETENVCOMMAREA(value, segment#)`

<i>length</i>	On return, contains the length remaining in the segment or COMMAREA after the function is performed. Its syntax is B (binary) with length 2.
<i>value</i>	The data to be passed. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character), with a length of either 8192 or 31744, depending on the data length.
<i>segment#</i>	The number of the segment where to store the data. Its syntax is B (binary) with length 2.

- Usage Notes**
- For IMS TM, use the following values to indicate the appropriate segment number:

0	Scratch Pad Area (SPA). The first 14 bytes here should never be modified.
1	The user data or, if using a non-seamless interface, the session parameter string.
2	The user data, if the first segment contains the session parameter string.
 - For CICS, the value for *segment#* is ignored.
 - For some CICS transactions the string of data that is passed in the COMMAREA consists of the session parameter string followed by user data.
 - For the SDK (C/C++), the maximum length for *value* applies even though much larger COMMAREAs are supported through the MAP table interface.
- See Also**
- TIBCO Object Service Broker for z/OS External Environments* or *TIBCO Object Service Broker for Open Systems External Environments* for information about using TIBCO Object Service Broker with external environments
 - TIBCO Object Service Broker Managing Data* for information about MAP tables

Example The following rule retrieves data from the EMPLOYEES table and returns the data to the COMMAREA:

RULE EDITOR ==>		SCROLL: P
SETENV_1;		
_ LOCAL MESSAGE, LENGTH;		
_ -----		
_ -----		
_ FORALL EMPLOYEES WHERE REGION = 'MIDWEST' :	1	
_ MESSAGE = MESSAGE EMPLOYEES.LNAME ';' ;		
_ END;		
_ LENGTH = \$SETENVCOMMAREA(MESSAGE, 1);	2	
_ -----		

SETNLSBIT

Sets the NLS bit for the specified table, in the RESERVED field of the TABLES table. (C)

Invocation `CALL SETNLSBIT(table, flag)`

<i>table</i>	A character string specifying the name of the table. Its syntax is C (fixed-length character string) with length 16.
--------------	--

<i>flag</i>	One of the following: Y – Turns the NLS bit on. N or anything except Y – Turns the NLS bit off.
-------------	---

- Usage Notes**
- Setting the NLS bit on for a specified table indicates that the table uses the system code page, rather than the TDS code page. For example, you must turn the NLS bit on when storing binary data in a table. In this case, you must use type S and syntax V, and setting the NLS bit on insulates your table from NLS translation, because the translation could render your binary data unintelligible.
 - You cannot use SETNLSBIT to modify the NLS bit of a TIBCO Object Service Broker system table.

Example This rule sets the NLS bit on for the table whose name is passed to the rule:

```
SET-NLS-BIT(TABLENAME);
_ LOCAL TABLENAME;
_ -----
_ CALL SETNLSBIT(TABLENAME, 'Y');                               +-----+
_ -----                                                         | 1
```

See Also *TIBCO Object Service Broker National Language Support* for information about using the NLS bit.

\$SETOPT

Sets the value of a session parameter or option. (C)

Invocation `CALL $SETOPT(parameter, value)`

<i>parameter</i>	The name of the parameter or option whose value you want to change.
------------------	---

<i>value</i>	The value to which you want to set the parameter or option.
--------------	---

Available Parameters and Options

Only the parameters and options identified are accepted by this tool. The checkmark indicates that the parameter or option is effective for the indicated platform. For compatibility purposes, some parameters or options are accepted on all platforms, but are effective only on the ones indicated.

Parameter or Option Name	Abbreviation	Description	Parameter or Option Type	Win / UNIX	z/OS
CHARSET	C	The default national character set.	Execution Environment	Y	Y
CICSVSAMSYNC	CICSVSAM	Specifies whether the VSAM server should issue SYNCPOINTS under CICS.	Execution Environment		Y
DECIMALSEPARATOR	DECSEP	The character to be used as the decimal separator.	Execution Environment	Y	Y
DISPLAYCODEPAGE		The code page used to display the TIBCO Object Service Broker data.	NLS		Y
DSBIFTYPE	DSBT	The file type used to process calls to the @READDSN and @WRITEDSN tools and the load/unload to external files tools.	Execution Environment	Y	

Parameter or Option Name	Abbreviation	Description	Parameter or Option Type	Win / UNIX	z/OS
DSFIELDSEP	DSFS	The field separator for LINE_SEPARATED_ASCII type import and export tables.	Execution Environment	Y	
DSIXFTYPE	DSXT	The file type used to process import and export tables.	Execution Environment	Y	
DSQUOTE	DSQ	The quote character used for LINE_SEPARATED_ASCII import tables.	Execution Environment	Y	
ERRMESSAGESCREEN	ERRMSGSCR	The response to the CICS end message that is issued when a TIBCO Object Service Broker session ends.	Execution Environment		Y
EXECLOCALSIZE	LOCALSIZE	The size of the area for rules local variables.	Execution Environment		Y
EXECSTACKSIZE	STACKSIZE	The size of the rules executor runtime stack in bytes.	Execution Environment		Y
IMSSCREENATTRIBU	IMSEDS	Specifies whether terminals with sessions established under this IMS TM client or Native Execution Environment have the extended data stream or extended attribute support.	Execution Environment		Y
LANGUAGE		The language used to display the TIBCO Object Service Broker user interface.	NLS		Y
PRINTCLASS	CLASS	The default JES SYSOUT class for output generated using TIBCO Object Service Broker print facilities.	Execution Environment		Y
PRINTCOPY	COPY	The number of copies to print of a report generated by TIBCO Object Service Broker.	Execution Environment		Y

Parameter or Option Name	Abbreviation	Description	Parameter or Option Type	Win / UNIX	z/OS
PRINTDATASET	DATASET	The name of a file or existing data set to which a TIBCO Object Service Broker generated report is written.	Execution Environment		Y
PRINTDEST	DEST	The default printer destination for output generated by TIBCO Object Service Broker.	Execution Environment		Y
PRINTERRORLOG	PERL	The default option for printing or not printing the TIBCO Object Service Broker workbench information logs in a multi-user Execution Environment, that is, CICS or Native Execution Environment.	Execution Environment		Y
PRINTFCB	FCB	The name of a Forms Control Buffer (FCB) to be used when printing output generated by TIBCO Object Service Broker.	Execution Environment		Y
PRINTFORM	FORM	The name of a form on which JES SYSOUT output generated by TIBCO Object Service Broker is to be printed.	Execution Environment		Y
PRINTUCS	UCS	The name of a universal character set (UCS) to be used to print the JES SYSOUT output generated by TIBCO Object Service Broker.	Execution Environment		Y
PRINTXWTR	XWTR	The name of an external writer (XWTR) to be used to print output generated by TIBCO Object Service Broker.	Execution Environment		Y

Parameter or Option Name	Abbreviation	Description	Parameter or Option Type	Win / UNIX	z/OS
SESSDSPXFRSCRMX	SDXMAX	The maximum amount of display and transfer (DISPLAY & TRANSFERCALL rules language statement) screen rows storage allowed for a user session.	Execution Environment		Y
TIMEOFFSET		The difference in minutes between the time shown on the workbench and the base system installation time.	User profile		Y
TRANDSPXFRSCRMX	TDXMAX	The maximum amount of display and transfer (DISPLAY & TRANSFERCALL rules language statement) screen rows storage allowed for a transaction.	Execution Environment		Y
VARLDELIMITER	VLD	The character to be used as the left delimiter for enclosing substituted variables in @SCHEDULEMODEL.	Execution Environment		Y
VARRDELIMITER	VRD	The character to be used as the right delimiter for enclosing substituted variables in @SCHEDULEMODEL.	Execution Environment		Y

- Usage Notes
- Changes made with \$SETOPT last for the duration of the session, or until \$SETOPT is called again.
 - Any parameter that can be set with \$SETOPT, except LANGUAGE, can be retrieved with \$GETOPT.

Exceptions

Invalid argument OPTION-NAME ==> (xyz) <=== for \$SETOPT	RANGERROR raised if the parameter name, for example, invalid argument, is not supported
Invalid value for OPTION-NAME ==> (xyz) <=== of \$SETOPT	RANGERROR raised if the value, for example, invalid value, is out of range for the given parameter

See Also *TIBCO Object Service Broker Parameters* for information about these parameters.
TIBCO Object Service Broker Programming in Rules for information about exceptions.

Example The following example uses \$SETOPT to reset the DECIMALSEPARATOR parameter, then retrieves the new setting with \$GETOPT and displays it on the end message:

RULE EDITOR ==>	SCROLL: P
DECIMAL(SEPARATOR);	
_ LOCAL DEC;	
_ -----	
_ -----+-----	
_ CALL \$SETOPT('DECIMALSEPARATOR', SEPARATOR);	1
_ DEC = \$GETOPT('DECIMALSEPARATOR');	2
_ CALL ENDMSG('DECIMALSEPARATOR IS: ' DEC);	3
_ -----	

When the DECIMAL rule is run using the argument ' , ' the end message reads:
DECIMALSEPARATOR IS: ,

\$SETP#POS

Defines the position and content of page number lines. (C)

Invocation `CALL $SETP#POS(line_number, left_string, center_string, right_string)`

<i>line_number</i>	An integer specifying the line on which to place the page number string. Its syntax is B (binary) with length 2.
<i>left_string</i>	The character string to be printed flush left. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>center_string</i>	The character string to be printed centered on the page. Its syntax can be C, V, or W.
<i>right_string</i>	The character string to be printed flush right. Its syntax can be C, V, or W.

- Prerequisites**
- Before a call to \$SETP#POS, the print arguments must be set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#).
 - \$SETP#POS deletes all defined titles and footers; therefore, call \$SETP#POS before calling [\\$SETTITLE](#).

- Usage notes**
- The new settings take effect after the next page break.
 - The percent sign (%) indicates the position of the page number within a string. Only the leftmost percent sign is considered.
 - The page number strings default to the format specified by:
`$SETP#POS(0, ' ', ' ', 'Page %')`

Use of line _number

- *line_number* can take on values greater than or equal to 0 and less than or equal to *length* - 1 (where *length* is the value specified in [\\$SETPRINT](#) or [\\$RESETPRINT](#)).
- If *line_number* is 0 and no other argument has a percent sign (%), the page number line is not printed.
- *line_number* is relative to the top of the page if it is 0 or positive; it is relative to the bottom of the page if it is negative.

- If *line_number* is 0, the page number line does not interfere with title line; the page number line is printed on the first physical line and the title line is printed on the second physical line. If *line_number* is a value other than 0, the page number line takes precedence over conflicting title lines and the page number line is printed while the title line is not.
- The page number line cannot be directly reset to anything other than 0. To reset the page number line to a value other than 0, first reset it to 0, and then reset it to the desired value.

Exceptions

ROUTINEFAIL	This exception is raised if \$SETP#POS is not preceded by a call to \$SETPRINT or \$RESETPRINT, or if the absolute value of <i>line_number</i> is greater than or equal to <i>length</i> - 1 (where <i>length</i> is the page length set by \$SETPRINT or \$RESETPRINT) or 29, whichever is less.
STRINGSIZE	The combined length of character strings exceeds <i>width</i> (where <i>width</i> is the page width set by \$SETPRINT or \$RESETPRINT) or 132, whichever is less.

Example

The following rule:

1. Prints a page at the default settings
2. Calls \$SETP#POS to set new settings
3. Forces a new page so that the new settings are in effect
4. Prints a page at the new settings:

```
SETP#POS_1;
```

```

-----
-----+-----
CALL $SETPRINT(10, 70, 1, 'SCR', 'N');      | 1
CALL $PRINTLINE('THIS PAGE HAS THE DEFAULT PAGE NUMBERS'); | 2
CALL $SETP#POS(0, 'LEFT TITLE', '- % -', 'RIGHT TITLE');   | 3
CALL $NEWPAGE;                                           | 4
CALL $PRINTLINE('THIS PAGE HAS THE NEW PAGE NUMBERS');    | 5
-----
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==> SCROLL ==> P

----- NEW PAGE -----

THIS PAGE HAS THE DEFAULT PAGE NUMBERS Page 1

----- NEW PAGE -----

LEFT TITLE - 2 - RIGHT TITLE
THIS PAGE HAS THE NEW PAGE NUMBERS

\$SETPRINT

Initializes the print attributes or, if they are already set, uses it to clear the titles for the output on the following pages. (C)

Invocation `CALL $SETPRINT(length, width, page_number, media, clear_title_yn)`

<i>length</i>	An integer specifying the number of lines per page or screen. It can take on values greater than or equal to 1 and less than or equal to 32767. Its syntax is B (binary) with length 2.
<i>width</i>	An integer specifying the number of columns per page or per screen. It can take on values greater than or equal to 1 and less than or equal to 256. Its syntax is B, with length 2.
<i>page_number</i>	An integer specifying the page number that appears on the subsequent page. It can take on values greater than or equal to 1 and less than or equal to 32767. Its syntax is B, with length 2.
<i>media</i>	<p>One of:</p> <p>PRT – Direct output to the printer.</p> <p>SCR – Direct output to the screen.</p> <p><i>filename</i> – If the filename in the <i>media</i> field is a z/OS data set, there are no restrictions to the data set format. An entry, for example, could be of FORMAT = VB, LRECL = 132.</p> <p>For Open Systems operating systems, the filename can include the full path, for example, E:\temp\myfile.txt. If only the filename is provided, the output file is the current working directory where the TIBCO Object Service Broker session is launched.</p> <p>Its syntax is V (variable-length character string) with length 54.</p>
<i>clear_title_yn</i>	<p>Indicates whether titles should be printed. Valid values are:</p> <p>Y – Clears all previously defined titles (through \$SETTITLE).</p> <p>N or " – Titles should be printed. Do not clear all previously defined titles.</p> <p>Its syntax is C (fixed-length character string), with length 1. To start a new set of pages without using the titles in previous pages, set <i>clear_title_yn</i> to Y.</p>

Usage Notes

If...	Then...
<i>page_number</i> is zero	No page number is printed. \$SETPRINT or \$RESETPRINT must be called before a call to another print tool.
The print arguments are already set	The first four arguments are ignored and do not override the current arguments. To override the current arguments use \$RESETPRINT.
A data set name is specified for <i>media</i>	The z/OS data set must be pre-allocated.



Control characters are emitted only if *media* is set to PRT. Control characters can be included in the output if PRT is redirected to a data set or file (for example, by specifying a file under the Print Parameters section of the User Profile option on the workbench).

Exceptions

RANGERROR	This exception is raised for one of the following reasons: <i>length</i> – Is less than or equal to zero. <i>width</i> – Is less than or equal to zero, or is greater than 256. <i>page_number</i> – Is less than zero. <i>media</i> – Is a number, an empty string, or a string without dot (.). <i>clear_title_yn</i> – Is neither Y or N. An empty string (") is valid and is treated as N.
ROUTINEFAIL	The <i>media</i> specified is not a valid type.

Example

The following rule sets the print arguments with a call to \$SETPRINT and prints four lines to the message buffer using the new print arguments:

```
SETPRINT_1;  
-----  
-----+-----  
CALL $SETPRINT(4, 12, 1, 'SCR', 'N');          | 1  
CALL $PRINTLINE('THIS IS');                     | 2  
CALL $PRINTLINE('A VERY');                      | 3  
CALL $PRINTLINE('NARROW');                     | 4  
CALL $PRINTLINE('SHORT PAGE');                 | 5
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P

- NEW PAGE -

      Page 1
THIS IS
A VERY
NARROW

- NEW PAGE -

      Page 2
SHORT PAGE
```

SETREMOTELOC

Sets the default remote location for distributed data processing. (C)

Invocation `CALL SETREMOTELOC(remoteloc)`

<i>remoteloc</i>	The value for the default remote location.
------------------	--

Usage Notes

- Use the [REMOTELLOCATION](#) shareable tool to determine the current value of the default remote location.
- SETREMOTELOC does not check to see if the value you supply for *remoteloc* is valid. Use the TIBCO Object Service Broker Administration menu's Resource Management facility to access a list of names available to your local node.

See Also TIBCO Object Service Broker *Application Administration* for information on distributed data processing.

TIBCO Object Service Broker for z/OS Installing and Operating for information on the Administration menu.

Example The following rule changes the default remote location, if it is not already the value required.

RULE EDITOR ==>	SCROLL: P
CHANGE_LOCATION(VALUE);	
—	
— -----	
— VALUE = REMOTELLOCATION;	Y N
— -----	
— CALL SETREMOTELOC(VALUE);	1
— CALL ENDMSG('THE LOCATION IS ' VALUE);	1 2
— -----	

\$SETRPTATTRIBUTE

Sets the attributes of the report that is to be printed. (C)

Invocation `CALL $SETRPTATTRIBUTE(report, attribute, value)`

<i>report</i>	The name of the report to which the attributes are to be applied. Its syntax is C (fixed-length character string) with length 16.
<i>attribute</i>	<p>The type of attribute that can be set. Valid values are:</p> <p>IMMEDIATE – Does not sort the records of the report; just sends the records to the output as they are read. The corresponding entry in <i>value</i> must be Y.</p> <p>OCCLIMIT – Limits the number of occurrences used by the Report Server to generate the report. The corresponding entry in <i>value</i> must be the maximum number of occurrences that can be inserted into the report.</p> <p>PAGELength – The physical page length. The entry in <i>value</i> specifies the size in number of lines.</p> <p>PAGEWIDTH – The physical page width. The entry in <i>value</i> specifies the size in number of characters.</p> <p>EJECT – Indicates whether a new page should be inserted between reports when more than one report is printed in a transaction. The entry for <i>value</i> should be either Y (new page should be inserted) or N (a new page should not be inserted).</p> <p>PAGENUMBER – Specifies the start page number to be used if more than one report is printed in a transaction. If <i>value</i> is an empty string, it indicates that the page numbers are to run consecutively through all the reports. The entry in <i>value</i> overrides the default page numbering.</p>
<i>value</i>	The setting to be applied for the <i>attribute</i> specified. Each <i>value</i> is discussed together with its corresponding <i>attribute</i> .

- Usage Notes**
- Page formatting options (PAGELength, PAGEWIDTH) *must* precede the call to \$SETRPTMEDIUM, since they copy the current settings for the named report and do not refresh them until the report is printed.

- If \$SETRPTATTRIBUTE is being used with overlapping report output, the page length information can be ignored if *value* is lower than the previous report's page length.
- The attribute IMMEDIATE must precede the INSERT statement in the report generation.
- Only one report can be sent directly to the output medium at a time.
- The attribute IMMEDIATE is ignored if sorting or derived fields are detected in the report definition.
- Use of \$SETRPTMEDIUM is encouraged because it enables centralization of all current and future attributes in a single call syntax instead of their being distributed among several shareable tools.
- Each of the attributes is set one at a time. To set multiple attributes you must re-enter the line with a new setting.

Example The following rule runs three occurrences of the test report RPTEST_1 starting at page 999 and sends it to the screen.

```
RPTEST_1;
-
- -----
-
- FORALL $EMPLOYEES :                               | 1
-   FORALL EMPLOYEES (EMPLOYEES.REGION) :
-     RPTEST_1$$$$$$$4.* = EMPLOYEES.*;
-     RPTEST_1$$$$$$$4.* = $EMPLOYEES.*;
-     INSERT RPTEST_1$$$$$$$4('SETRPTATT');
-     END;
-   END;
- CALL $SETRPTATTRIBUTE('RPTEST_1', 'PAGENUMBER', '999'); | 2
- CALL $SETRPTATTRIBUTE('RPTEST_1', 'OCCLIMIT', '3');      | 3
- CALL $SETRPTMEDIUM('RPTEST_1', 'VISUAL' 'SCR');          | 4
- PRINT RPTEST_1;                                           | 5
- CALL ENDMSG(MESSAGE('@RPTGEN', 602, 'RPTEST_1', 'SCR' '' ); | 6
- -----
- ON LOGLIMIT :
-   CALL END MSG(MESSAGE('@RPTGEN', 698, ''));
```

\$SETRPTMEDIUM

Sets the medium to which a report is to be printed. (C)

Invocation `CALL $SETRPTMEDIUM(report, mediumtype, medium)`

<i>report</i>	The name of the report whose output medium you are setting. Its syntax is C (fixed-length character string) with length 16.
<i>mediumtype</i>	<p>The type of medium to be used when printing a report. Valid values are:</p> <p>VISUAL – The report is to appear on the screen. The corresponding <i>medium</i> setting must be SCR.</p> <p>PRINTER – The report is to be directed to a printer. The corresponding <i>medium</i> setting must be PRT.</p> <p>DIRECTFILE – The report is to be output to the file whose name is contained in the variable <i>medium</i>.</p> <p>INDIRECTFILE – The report is output to a file identified in <i>medium</i>. <i>Medium</i> contains either the DDNAME associated with the file (on z/OS systems) or the name of an environment variable containing the name of the file (on Open Systems).</p>
<i>medium</i>	<p>The qualification of the <i>mediumtype</i>. These can be:</p> <p>SCR – The report medium is the screen.</p> <p>PRT – The report medium is a printer.</p> <p><i>filename</i> – The output is sent directly to the named file.</p> <p><i>ddname</i> – Specifies the DDNAME associated with the file (on z/OS).</p> <p><i>environmental variable</i> – Specifies the name of the environment variable containing the name of the file to which you are directing output.</p>

- Usage Notes**
- The default settings for outputting a report are VISUAL for *mediumtype* and SCR for *medium*. These are used if NULL or an empty string is supplied, or if a report is printed without explicitly setting the medium for its output.
 - Use of \$SETRPTMEDIUM is encouraged because it adds the extra capability of indirect output (\$RPTPRINT does not support this feature).

Example The following rule sends the output of the test report RPTEST_2 to the screen:

```
RPTEST_2;
-
- -----
-                                     +-----
- FORALL $EMPLOYEES :                                     1
-   FORALL EMPLOYEES (EMPLOYEES.REGION) :
-     RPTEST_2$$$$$$$4.* = EMPLOYEES.*;
-     RPTEST_2$$$$$$$4.* = $EMPLOYEES.*;
-     INSERT RPTEST_2$$$$$$$4('SETRPTATT');
-   END;
- CALL $SETRPTMEDIUM('RPTEST_2', 'VISUAL' 'SCR');         2
- PRINT RPTEST_2;                                         3
- CALL ENDMSG(MESSAGE('@RPTGEN', 602, 'RPTEST_2', 'SCR' '' ); 4
- -----
- ON LOGLIMIT :
-   CALL END MSG(MESSAGE('@RPTGEN', 698, ''));
-
-
```


\$SETSESSIONEND

Sets what execution is to take place when a TIBCO Object Service Broker session ends by returning data from the session to an external environment. (C)

Invocation `CALL $SETSESSIONEND(action, value)`

<i>action</i>	<p>The type of execution that is to take place. Valid values are:</p> <p>IMS TM – FORMAT or SWITCH</p> <p>CICS – XCTL or START</p> <p>TSO or BATCH – ABEND, RC, or RETCODE</p>
<i>value</i>	<p>The transaction code or program name for the next execution. Valid values depend on the type of action and are:</p> <p>FORMAT or SWITCH – Syntax C, Length 8</p> <p>XCTL – Syntax C, Length 8</p> <p>START – Syntax C, Length 4</p> <p>ABEND, RC, or RETCODE – Syntax B, Range 0-3999 for z/OS, 0-127 for Open Systems.</p>

See Also *TIBCO Object Service Broker for z/OS External Environments* for information about using TIBCO Object Service Broker with external environments.

Example In the following examples:

- SETSESSIONEND_1 sets the execution for a IMS TM application. According to the rule, when the current session ends, IMS TM initiates TRAN01.
- SETSESSIONEND_2 sets the return code to 4. The return code can be used by the calling environment (for example, by JCL in z/OS batch) to determine the appropriate action to take.

SETSESSIONEND_1 Rule

```
RULE EDITOR ==>
SETSESSIONEND_1;
```

SCROLL: P

```

-
- -----
-                                     +-----
- CALL $SETSESSIONEND('FORMAT', 'TRAN01');          | 1
- -----
```

SETSESSIONEND_2 Rule

RULE EDITOR ==>		SCROLL: P
SETSESSIONEND_2;		
-		
-		
-		+
- CALL \$SETSESSIONEND('RETCODE', 4);		1
-		

\$SETTITLE

Sets a title or footer to be printed on subsequent pages of output. (C)

Invocation `CALL $SETTITLE(line_number, left_string, center_string, right_string)`

<i>line_number</i>	An integer specifying the line on which to print the title. Its syntax is B (binary) with length 2.
<i>left_string</i>	A character string specifying the string to be printed flush left. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>center_string</i>	A character string specifying the string to be centered (based on the center of the page). Its syntax can be C, V, or W.
<i>right_string</i>	A character string specifying the string to be printed flush right. Its syntax can be C, V, or W.

Prerequisites The print arguments must be previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to \$SETTITLE.

- Usage Notes**
- [\\$PRINTLINE](#) must be called after \$SETTITLE is called, to print the title/header of the page.
 - If *line_number* is positive, a title is printed. If it is negative, a footer is printed. If it is 0, an exception is raised.
 - The magnitude of *line_number* can be greater than or equal to 1 and less than or equal to *length* -1 (where *length* is the page length specified using the [\\$SETPRINT](#) or [\\$RESETPRINT](#)).
 - All three arguments must be given. If a particular title is not to be printed, replace either *left_string*, *center_string*, or *right_string* with a null string ('').

Exceptions

RANGERROR	Raised when the value provided for <i>line_number</i> is zero (0).
------------------	--

ROUTINEFAIL	Raised if the call is not preceded by a call to \$RESETPRINT or \$SETPRINT, or if the number of title lines and the number of footing lines add up to be greater than or equal to either <i>length</i> - 1 (where <i>length</i> is the page length specified by \$SETPRINT or \$RESETPRINT) or 29, whichever is less.
STRINGSIZE	Raised if the combined length of the character strings exceeds <i>width</i> (where <i>width</i> is the page width specified by \$SETPRINT or \$RESETPRINT) or 132, whichever is less; or if the length of the left or right character strings (<i>left_string</i> or <i>right_string</i>) is such that the center string (<i>center_string</i>) cannot be centered.

Example The following rule prints a page to the message log using the default title, sets new titles with a call to \$SETTITLE, forces a page break so that the new titles are in effect, and prints a page using the new titles:

```
SETTITLE_1;
-
- -----
-                                     +-----+
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N');           | 1
- CALL $PRINTLINE('THIS PAGE HAS THE DEFAULT TITLES'); | 2
- CALL $SETTITLE(1, 'LEFT TITLE', 'CENTER TITLE',   | 3
-   'RIGHT TITLE');                               |
- CALL $NEWPAGE;                                   | 4
- CALL $PRINTLINE('THIS PAGE HAS THE NEW TITLES'); | 5
- -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

----- INFORMATIONAL MESSAGE LOG -----		
COMMAND ===>		SCROLL ===> P
----- NEW PAGE -----		
		Page 1
THIS PAGE HAS THE DEFAULT TITLES		
----- NEW PAGE -----		
		Page 2
LEFT TITLE	CENTER TITLE	RIGHT TITLE
THIS PAGE HAS THE NEW TITLES		

\$SETTRANSACTION

Returns the current name of a TIBCO Object Service Broker transaction and sets a new name. (F)

Invocation `currentname = $SETTRANSACTION(attribute, value)`

<i>currentname</i>	The present name of the transaction is returned in this variable.
--------------------	---

<i>attribute</i>	The transaction attribute to set or change. NAME is the only attribute acceptable to this tool.
------------------	---

<i>value</i>	The new transaction value; use syntax C (fixed-length character string) with maximum length of 8.
--------------	---

- Usage Notes**
- The transaction name persists for the life of the transaction (until it is changed). It is inherited by other initiated transactions, whether by a TRANSFERCALL or EXECUTE statement.
 - On the z/OS platform, the transaction name appears in applicable SMF records for the transaction.

Example The following rule sets the transaction name to TRAN01:

RULE EDITOR ==>	SCROLL: P
SETTRAN_1;	
_ LOCAL PRESENTNAME;	
_ -----	
_ -----	
_ PRESENTNAME = \$SETTRANSACTION('NAME', 'TRAN01');	+-----+ 1
_ -----	

SETXPARM

Overrides a server parameter or the Table Definer default value for a field at table access time. (C)

Invocation `CALL SETXPARM(component, entity, parm name, value, location)`

<i>component</i>	The scope of the override, either TABLETYPE or TABLENAME: TABLETYPE – Indicates that the override <i>value</i> specified is used for all tables of this type. TABLENAME – Indicates that the override <i>value</i> applies only to this table name.
<i>entity</i>	The table type or table name, depending on component.
<i>parm name</i>	A valid parameter name as defined in @@SERVERPARMS(<i>tabletype</i>) and listed in Server Parameters and Fields below, or a valid field name defined in the Table Definer and listed in Server Parameters and Fields below.
<i>value</i>	The desired override value.
<i>location</i>	The physical location of the table; the Data Object Broker where the external DBMS resides.

- Usage Notes**
- The override value remains for the life of the user's session or until a [RESETXPARM](#) is issued.
 - This function is valid only for external DBMS table types.

Server Parameters and Fields

These are the server parameters or field values that you can override at runtime:

Table Type	Name	Parameter or Field	Default Value	Maximum Value Length
IDM	SERVERID	Parameter	DEFAULT	8
	DBNAME	Field		8
	READY_MODE	Field	SR	2
	OPTIMIZEUPDATE	Field	N	1
	USERSUBSCHEMA	Field		8
IMS	SERVERID	Parameter	DEFAULT	8
	SERVERTYPE	Parameter	IMS	8
	PSBNAME	Parameter		8
	DBNAME	Field		8
	OPTIMIZEUPDATE	Field	N	1
ADA	SERVERID	Parameter	DEFAULT	8
DAT	SERVERID	Parameter	DEFAULT	8
DB2	SERVERID	Parameter	DEFAULT	8
	SERVERTYPE	Parameter	DB2	8
204	SERVERID	Parameter	DEFAULT	8
SLK	SERVERID	Parameter	DEFAULT	8

- Example

 - This example sets SERVERID for all IDM (IDMS/DB) tables to TORONTO for the user’s session, until the next SETXPARM or the resetting of the overrides:
 CALL SETXPARM('TABLETYPE', 'IDM', 'SERVERID', 'TORONTO', '');
 - The following example sets the Optimize flag for table EMPLOYEE to Y:
 CALL SETXPARM('TABLENAME', 'EMPLOYEE', 'OPTIMIZEUPDATE', 'Y', '');

\$SHOWCHANNEL

Returns the 16-character name of the current channel, if one exists; otherwise, returns blanks. (F)

Invocation *channel* = \$SHOWCHANNEL, where *channel*, on return, contains the 16-character name of the program's current channel, if one exists. Otherwise, *channel* contains blanks.

\$SIGNAL

Raises the specified exception. (C)

Invocation `CALL $SIGNAL(exception,tablename)`

<i>exception</i>	The name of the exception to be signalled. Its syntax is C (fixed-length character string) with length 16.
<i>tablename</i>	The name of a table associated with the exception. If no table is to be associated with the exception, specify an empty string(""). Its syntax is C with length 16.

- Usage Notes**
- The exception to be signalled can be a user-defined exception or a system exception.
 - The exception can be specified as an indirect reference. This cannot be done using the SIGNAL statement.
 - A table name can be used to limit the scope of data access exception handlers.

See Also *TIBCO Object Service Broker Programming in Rules* for more information about exception handling and about the Signal statement.

Example The following rule demonstrates the use of \$SIGNAL to signal a user-defined exception read from a table. It also shows how to simulate a GETFAIL condition on a table without actually issuing the GET. It is assumed that exception handlers for the user-defined exceptions are specified in higher-level rules.

PROCESS(VALUE);		

VALUE > 1000;		Y N

CALL \$SIGNAL('GETFAIL','KEYVALUES');		1
GET KEYVALUES WHERE KEY = VALUE;		1
CALL \$SIGNAL(KEYVALUES.EXCEPTION, '');		2

ON GETFAIL KEYVALUES:		
CALL ENDMSG('Key value ' VALUE ' not found.');		

SIXBUILD

Creates a secondary index online for a TDS table. (C)

Invocation

Do one of the following:

- From the administrator's workbench, move the cursor to the DT Define Table option and press Enter.
- Within a rule, type:

```
CALL SIXBUILD(table , secondary_key)
```

<i>table</i>	The name of the table on which to build the index.
<i>secondary_key</i>	The name of the field for which a secondary index must be created. A secondary index cannot be created on a field with syntax RD (raw data) or UN (Unicode).

Usage Notes

- You need change definition rights to build an index on a table.
- SIXBUILD ignores the location parameter of the input table. Minimal table definitions are not allowed.
- SIXBUILD runs and updates are made even if it is called in a transaction that is running in browse mode.
- If you want to build multiple secondary indexes on a large existing table, use S6BBSIX, the batch secondary build utility.

Using SIXBUILD from the Administrator's Workbench


If you use SIXBUILD from the administrator's workbench, when you press Enter a list of tables appears on the screen. From this screen:

1. Type the line command **x** beside the table you want to build a secondary index for and press Enter.
2. From the displayed screen, select the field or fields on which you want to build the secondary index.


Unloading Tables with Secondary Indexes

Because secondary indexes are not carried over when tables are unloaded, promotion problems can result. To use SIXBUILD as a work around, complete the following steps:

- 1. Unload the table definition to data set A.
- 2. Unload table data only to data set B.
- 3. Load the table definition to the target system.
- 4. Call SIXBUILD or, on the Administrator workbench, (@ADMIN) use the Table Definer to designate SIX fields.
- 5. Load table data. SIX should be built as the table is loaded.



Be cautious when using SIXBUILD because it does not take part in the TIBCO Object Service Broker two-phase commit/intent list protocol. It is strongly recommended that this tool not be used in a transaction that accesses or updates data within the same table and that it should normally be the only logical unit of work within a transaction. If a user uses this tool on the data of a bound table, the data is updated (the updates take place on the bound copy of the data).



If SIXBUILD fails, you must run it again before running another TIBCO Object Service Broker function, to prevent damage to the database. If SIXBUILD fails again, contact your database administrator or TIBCO Support immediately.

See Also

TIBCO Object Service Broker *Application Administration* for information about building secondary indexes.
TIBCO Object Service Broker for z/OS Utilities for more information on S6BBSIX.

Exceptions

DATAREFERENCE	The field does not belong to the table.
DEFINITIONFAIL	The secondary index field length is greater than 127, the table definition does not exist or is inconsistent, or the table is not a TDS table.
LOCKFAIL	The table that you want to build the index on is being used by someone else.
ROUTINEFAIL	The table definition is minimal.
SECURITYFAIL	You did not pass the security check.

Example

The following statement builds a secondary index for the table EMPLOYEE using the field MGR#:

```
SIXBUILD_1;
-----
CALL SIXBUILD('EMPLOYEE', 'MGR#');          | 1
-----
```

Change to Table Definition

Executing the rule causes the following change to the table definition:

COMMAND==>										TABLE DEFINITION									
Table: EMPLOYEE					Type: TDS					Unit:USR40					IDgen: N				
Parameter Name		Typ	Syn	Len	Dec	Reference				'	Event Rule		Typ	Acc					
-----		-	-	---	---	-----				,	-----		-	-					
										,									
Field Name		Typ	Syn	Len	Dec	Key	Ord	Rqd		Default	Reference								
-----		-	-	---	---	-	-	-		-----	-----								
EMPNO		I	P	3	0	P													
LNAME		S	C	22	0														
POSITION		S	C	14	0														
MGR#		I	P	3	0	S													
DEPTNO		I	B	2	0														
SALARY		Q	P	3	2														
HIREDATE		D	B	4	0														
ADDRESS		S	V	38	0														
CITY		S	C	20	0														
PROV		S	C	3	0														
P_CODE		S	C	7	0														
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC																			

SIMPLESELECT

Processes a selection string into a format that can be used by the FORALLA tool.
(C)

Invocation `CALL SIMPLESELECT(selection)`

<i>selection</i>	A string specifying selection criteria.
------------------	---

- Usage Notes**
- The syntax for selection is <field name><relational operator><expression>. For valid expressions, refer to the *TIBCO Object Service Broker Programming in Rules* manual.
 - The local variable SEL_STR must be declared by the calling rule. It contains the processed selection string in a format suitable for use in FORALLA. SEL_STR must be reinitialized to the empty string between calls of SIMPLESELECT.
 - The local variable IN_TBL must be declared by the calling rule. It contains the name of the table the selection is on.
 - The local variable MSG must be declared by the calling rule. It contains the error message if SIMPLESELECT fails.

Exceptions

SYNTAX_ERROR	Raised if there is a syntax error in the selection. Returns an error message in MSG.
---------------------	--

Example The following example:

1. Takes an Author supplied by the user and calls SIMPLESELECT to build a selection string for TABLES WHERE AUTHOR = value.
2. Accesses all rows of TABLES which match the WHERE clause and counts the occurrences.
3. Writes the names of the tables to the message log.

```

QUERY_AUTHOR(AUTH);
_LOCAL SEL_STR, IN_TBL, MSG, COUNT;
- -----
-                                     +-----+
- IN_TBL = 'TABLES';                                     | 1
- CALL SIMPLESELECT('AUTHOR = ' || QUOTE(AUTH));        | 2
- CALL FORALLA(IN_TBL, '', SEL_STR, '');                 | 3

```

```
- UNTIL TABLEEND: | 4
-   COUNT = COUNT + 1; |
-   CALL MSGLOG(TABLES.NAME); |
-   CALL FORALLB(IN_TBL); |
-   END; |
- CALL FORALLE(IN_TBL); | 5
- CALL ENDMSG(COUNT || ' TABLES FOUND WITH AUTHOR ' || AUTH) | 6
- ; |
- -----
- ON SYNTAX_ERROR:
-   CALL ENDMSG(MSG);
- ON TABLEEND:
-   CALL FORALLE(IN_TBL);
-   CALL ENDMSG('NO TABLES FOUND WITH AUTHOR ' || AUTH);
```

SIXBUILD_CARDS

Defines the control cards required by the Batch Secondary Index Build utilities.
(E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type SIXBUILD_CARDS <Enter>
	COMMAND prompt	Type EX SIXBUILD_CARDS <Enter>

Usage Notes

- SIXBUILD_CARDS helps you define the control cards required by the S6BBSIX and hrnbrsix (Batch Secondary Index Build for TDS tables) utilities.
- SIXBUILD_CARDS displays a series of screens where you identify the table and secondary index fields, and then writes the necessary controls in a file.
- On non-z/OS systems, set the DSBIFFTYPE Execution Environment parameter to TEXT.



This setting could affect the behavior of other tools.

See Also

TIBCO Object Service Broker for z/OS Utilities or *TIBCO Object Service Broker for Open Systems Utilities* for complete information about the Batch Secondary Index Build utilities.

TIBCO Object Service Broker Parameters for information on the DSBIFFTYPE Execution Environment parameter.

Procedure Overview

The following table summarizes the tasks to prepare cards:

- [Prepare the file for the control cards, page 640](#)
- [Invoke SIXBUILD_CARDS, page 640](#)
- [Specify secondary indexes, page 641](#)
- [Review the output file, page 641](#)

Task A Prepare the file for the control cards

On z/OS systems, before you can use SIXBUILD_CARDS you must allocate a file to hold the control cards. This file must have a record length of 80 bytes and be fixed block.

Task B Invoke SIXBUILD_CARDS

Invoke the control card preparation facility, by following these steps:

- 1. Execute SIXBUILD_CARDS from the workbench.

The initial screen is illustrated with sample input here:

Control Card Definition for SIXBUILD

v

Control Card File

:

USR01.BATCH.CNTLCARD(FEB2000)

Table

:

EMP_MSTR_TDS

Character Set

:

ENGL

PAGE FILL LEVELS

:

(Default = 75 % if blank)

Group Index

:

Primary Index

:

Secondary Index

:

Data Pages

:

of Input Records

:

(Default = 100000 if blank)

Total # of Fields

:

(Default = 50 if blank)

Dynamic Unit Name

:

(Default = SYSDA if blank)

Dynamic Block Size

:

(Default = 4096 if blank)

PFKEYS: 1=HELP 3=EXIT 4=CONTINUE 12=CANCEL

- 2. Supply values for the following fields:

Control Card File	The filename. You can change the name by typing over it.
Table	The lower half of the screen specifies override values for the batch secondary index build process. Default values are available and appear on the screen. To override any of these defaults, type in a new value in the space provided.

- 3. To continue defining the control cards, press PF4.

The screen to specify the secondary indexes is shown here:

These are the fields and their key types for the selected table.

Primary	Secondary	Field	Name
Y	\bar{Y}	PK	
		F1	
		F2	
		F3	
		F4	
		F5	
		F6	

This screen displays a list of TIBCO Object Service Broker fields with the currently assigned primary keys. Type a Y in the Secondary column beside the field or fields for which you want to define a secondary index. You can define a secondary index on any field except the first primary key field or fields with syntax RD (raw data) or UN (Unicode).

Task D Review the output file

An example of an output file containing the control cards is shown here:

TIBCO Object Service Broker Shareable Tools

Notice the P and S designations next to the **PK** and **F1** fields. These letters mean that **PK** is a primary key, and **F1** is a secondary index. If a field is both a primary key and a secondary index, the designation is Q.

Record Layout The following tables relate control fields on the screens with their equivalent fields in the control records:

Input File Definition	Screen	Record ID/Type	Columns
Table Name	1	H/R	64 – 79
Character Set:	1	H/R	9 – 12
Page Fill Levels:	1		
Group Index		S	3 – 4
Secondary Index		S	9 – 10
# of Input Records	1	S	15 – 29
Total # of Fields	1	S	31 – 33
Dynamic Unit Name	1	S	35 – 42
Dynamic Block Size	1	S	44 – 48
# Volumes for work file	n/a	S	50 – 51

Field Definitions	Screen	Record ID/Type	Columns
Secondary Key Fields	2		
TIBCO Object Service Broker Field Definitions:	n/a		
Name		H/F or P	9 – 44
Semantic Type or Syntax ¹		H/F or P	45 – 48
Syntax for fields that are neither RD nor UN		H/F or P	48

Field Definitions	Screen	Record ID/Type	Columns
Length		H/F or P	49 – 52
Decimal		H/F or P	54 – 55
Key Type		H/F	57

1. Columns 45 to 48 contain the syntax if the field is raw data or Unicode; otherwise, the semantic type goes in column 46.



- H type records contain a sequence number in columns 3 – 5 that must be consecutive and must commence with 001. The types within these records must be in the order: R, if any, followed by P, if any, followed by F.
- The Dynamic Block Size, on z/OS only, is used for any temporary work files required to process secondary indexes and for table instances. The small default block size can be detrimental to good performance. If secondary indexes are to be built for large tables, that is, tables with large numbers of occurrences, consider optimizing the Dynamic Block Size.

SIXDELETE

Deletes an existing secondary index. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	DT Define Table option	Press Enter
From a rule		Type CALL SIXDELETE(<i>table</i> , <i>secondary_key</i>)

Where:

<i>table</i>	The name of the table whose index is deleted.
<i>secondary_key</i>	The name of the field whose secondary index must be deleted.

Usage Notes


- You need change definition rights to delete an index from a table.
- SIXDELETE runs and updates are made even if it is called in a transaction that is running in browse mode.
- SIXBUILD ignores the location parameter of the input table. Minimal table definitions are not allowed.
- The table occurrences are not modified in any way by SIXDELETE.
- The field whose secondary index is deleted has the S, or lowercase s for an incomplete secondary index, deleted from the KEY field of the table definition. If the field is part of a composite primary key, the Q is changed to P.

Using SIXDELETE from the Administrator's Workbench


If you use SIXDELETE from the administrator's workbench, when you press Enter a list of tables appears on the screen. From this screen:

1. Type the line command **x** beside the table you want to delete the secondary index from and press Enter.

- 2. From the displayed screen, select the field or fields from which to delete the secondary index.



Caution is advised when using SIXDELETE because it does not take part in the TIBCO Object Service Broker two-phase commit/intent list protocol. It is strongly recommended that this tool not be used in a transaction that accesses or updates data within the same table and that it should normally be the only logical unit of work within a transaction. If a user uses this tool on the data of a bound table, the data is updated (the updates take place on the bound copy of the data).



If SIXDELETE fails, you must run it again before running another TIBCO Object Service Broker function, to prevent damage to the database. If SIXDELETE fails again, contact your database administrator or TIBCO Support immediately.

See Also

TIBCO Object Service Broker Application Administion for information about secondary indexes.

Exceptions

DATAREFERENCE	The field does not belong to the table.
DEFINITIONFAIL	The table definition does not exist or is inconsistent, or is not a TDS table.
LOCKFAIL	The table that you want to delete the index from is being used by someone else.
ROUTINEFAIL	The table definition is minimal.
SECURITYFAIL	You did not pass the security check.

Example

The following statement deletes the secondary index on the table EMPLOYEE:

```
RULE EDITOR ==>                                SCROLL: P
SIXDELETE_1;
-
- -----
- -----+-----
- CALL SIXDELETE('EMPLOYEE', 'MGR#');           | 1
- -----
```

Table Definition Changes

Executing the rule causes the following change to be made to the table definition:

COMMAND==>										TABLE DEFINITION									
Table: EMPLOYEE					Type: TDS					Unit:USR40					IDgen: N				
Parameter Name		Typ	Syn	Len	Dec	Reference				'	Event Rule		Typ	Acc					
-----		-	-	---	---	-----				,	-----		-	-					
										,									
Field Name		Typ	Syn	Len	Dec	Key	Ord	Rqd	Default		Reference								
-----		-	-	---	---	-	-	-	-----		-----								
EMPNO		I	P	3	0	P													
LNAME		S	C	22	0														
POSITION		S	C	14	0														
MGR#		I	P	3	0														
DEPTNO		I	B	2	0														
SALARY		Q	P	3	2														
HIREDATE		D	B	4	0														
ADDRESS		S	V	38	0														
CITY		S	C	20	0														
PROV		S	C	3	0														
P_CODE		S	C	7	0														
PFKEYS: 3=END 12=CANCEL 22=DELETE 13=PRINT 14=FIELDS 21=DATA 2=DOC																			

It returns the following to the ENDMSG:
THE RETURN CODE IS 0.

\$SKIPLINE

Outputs zero or more blank lines. (C)

Invocation `CALL $SKIPLINE(count)`

<i>count</i>	An integer specifying the number of lines to skip. It can take on values greater than or equal to 0 and less than or equal to 32767. Its syntax is B (binary) with length 2.
--------------	--

Prerequisites The print arguments must be previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to \$SKIPLINE.

Exceptions

LOGLIMIT	Raised if too much output is sent to the message log.
RANGEROR	Raised if <i>count</i> is less than zero.

Example The following rule prints a line to the message log, skips five lines, and prints a second line to the message log:

```
SKIPLINE_1;
```

```

_
_  -----
_
_  -----+-----
_ CALL $SETPRINT(10, 70, 1, 'SCR', 'N');          | 1
_ CALL $PRINTLINE('THIS IS LINE ONE');           | 2
_ CALL $SKIPLINE(5);                             | 3
_ CALL $PRINTLINE('THIS IS LINE SEVEN');         | 4
_  -----
_
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATION LOG -----
COMMAND ===>                                SCROLL ===> P

----- NEW PAGE -----

THIS IS LINE ONE
```

Page 1

THIS IS LINE SEVEN

\$SLEEP

Causes the Execution Environment to go dormant. (C)

Invocation `CALL $SLEEP(milliseconds)`

<i>milliseconds</i>	The number of milliseconds you want the Execution Environment to stay dormant.
---------------------	--

Exceptions None.

Example The following example uses \$SLEEP to control the length of time between successive sweeps of a table.

```
MONTABLE(TABLENAME, SECONDS);
- LOCAL INTERVAL;
-----
- INTERVAL = SECONDS * 1000;
- UNTIL DONE :
- FORALL TABLENAME :
-     CALL ROWCHECK(TABLENAME);
-     END;
- CALL $SLEEP(INTERVAL);
- END;
-
-
-
-----
```

SOE

Edits a single occurrence in a table. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type SOE (<i>tablespec</i>) <Enter>
	COMMAND prompt	Type EX SOE (<i>tablespec</i>) <Enter>
From a rule		Type CALL SOE(<i>tablespec</i>)

Where:

<i>tablespec</i>	The name of the table (and parameters, if any) containing the single occurrence that is to be edited.
------------------	---

Usage Notes If you do not supply a value for *tablespec*, pressing Enter displays a screen prompting for a value.

- See Also**
- [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Single Occurrence Editor
 - *TIBCO Object Service Broker Managing Data* for information about the Single Occurrence Editor.

Example To edit the record for employee number 61219 in the EMPLOYEES(MIDWEST) table, type the following at the COMMAND line:

```
EX SOE(EMPLOYEES)
```

The following prompt screen appears:

```

          --- SINGLE OCCURRENCE EDITOR ---
EDITING TABLE   : EMPLOYEES
TABLE TYPE      : TDS

-----
ENTER PARM VALUE REGION      :
ENTER KEY VALUE  EMPNO      :
```

PFKEYS: 1=HELP 3=EXIT 12=EXIT

Editing Screen

After supplying the value MIDWEST for REGION and 61219 for EMPNO, the following editing screen appears:

```
          --- SINGLE OCCURRENCE EDITOR ---  
EDITING TABLE : EMPLOYEES(MIDWEST)  
TABLE TYPE    : TDS  
COMMAND ==>
```

```
EMPNO          : 61219  
LNAME          : WONG  
POSITION       : SENIOR ANALYST  
MGR#           : 79912  
DEPTNO         : 50  
SALARY         : 820.00  
ADDRESS        : 567 Pine St.  
CITY           : MISSISSAUGA  
STATE_PROV     :  
ZP_CODE        :  
HIREDATE       : 1997-12-03
```

PFKEYS: 1=HELP 2=DOCUMENTATION 3=SAVE 12=CANCEL 13=PRINT 22=DELETE

@STATICSQL

Defines and generates static SQL to be used to access DB2 data. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type @STATICSQL <Enter>
	COMMAND prompt	Type EX @STATICSQL <Enter>

Usage Notes Pressing Enter displays the screen illustrated here:

```

Defining Static SQL for use by DB2 Server
Assemble modules generated      Link Static SQL Handlers      Re-Bind DB2 Server

      MetaStor Table          DB2 Table          Status  Member  DB2
                                ----->  Pre/Suffix  Vers
_ @DB2FSTRXDB                HURON.HRNTRXDB
_ @EXTNDB2TRXDB              EZSA0.TRXDB
_ @SYSCOLUMNS               SYSIBM.SYSCOLUMNS
_ @SYSTABLES                 SYSIBM.SYSTABLES
_ DAT2                       ABC20.PREDICATE_TABLE
_ DB2                        FGH30.CLAIMS82
_ DB2_BRANCH                 OST00.BRANCH          G    PFC          V2.3
_ DB212                     USR20.PREDICATE_TABLE
_ DB251                     USR20.PREDICATE_TABLE
_ DB5                       USR20.STRUCTURE_TABLE
_ HAIG_DB2                  OST68.HRNPremium
_ NJSDB2                    MNO30.T00            I    T00          V2.3
_ NJSDB2COL                 SYSIBM.SYSCOLUMNS
_ NJSDB2COL6000             DB26000.SYSIBM.SYSCOLUMNS
_ NJSDB2TBL                 SYSIBM.SYSTABLES
G=GENERATE  R=REMOVE
PFKEYS: 1=HELP 3=GENERATE(ONLINE) 15=GENERATE(BATCH) 12=CANCEL

```

See Also *TIBCO Service Gateway for DB2 Installing and Operating* for information on @STATICSQL.

STE

Invokes the Table Editor. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	ED edit table option	Type <i>tablename</i> <Enter>
	EX Execute Rule option	Type STE(<i>tablename</i>) <Enter>
	COMMAND prompt	Type ED <i>tablename</i>) <Enter>
From a rule		Type CALL STE(<i>tablename</i>)

Where:

<i>tablename</i>	The name of the source table and any parameters.
------------------	--

Usage Notes

- If the table is parameterized and you do not supply any parameter values, executing STE displays a prompt for the parameter values.
- If you supply a value for *tablename*, executing STE displays a screen that you use to edit the table.
- If you do not supply a value for *tablename* when you use the ED edit table option or the command line, executing STE displays the list of tables defined in your TIBCO Object Service Broker database. Select the table that you require from this list.
- If you do not supply a value for *tablename* when you use the EX execute rule option, pressing Enter displays a screen prompting for a table name.
- The default search path for event rules varies depending on how you invoke STE. When executed from within a rule or from the EX execute rule option, STE searches your local library for any event rules that can be specified. When invoked through the ED edit table option or at the command prompt through the ED command, the default TIBCO Object Service Broker installation specifies that STE searches your system library for the event rule. You can therefore use STE to verify if a new or newly modified event rule is functioning.

See Also

- [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Table Editor

- *TIBCO Object Service Broker Managing Data* for information about the Table Editor.

Example From the following screen, the user can browse or edit a table containing data on the employees of a selected region:

Date: 2000-04-11 Employees by Region

- _ East
- _ Mideast
- _ Central
- _ Midwest
- _ West

FCNKEYS: 2=GET EMPLOYEES 4=EDIT INFORMATION 12=EXIT

EMPLOYEES Table

The fields of the screen table containing the regions are named to match the parameters of the EMPLOYEES table, which is also parameterized by region:

SCREEN PAINTER COMMAND ==> Scroll: P
...+...1...+...2...+...3...+...4...+...5...+...6...+...7...+...
-A East
-A Mideast
-A Central
-A Midwest
-A West

Table: REGIONS			Unit: USR40											
ROW	COL	FIELD NAME	Type	Syn	Len	Dec	Just	Fill	Prot	Show	Rqd	Hi	Skip	Null
---	---	-----	-	-	----	---	-	-	-	-	-	-	-	-
1	2	EAST		C	1	0	L	—	N	Y	N	N	Y	Y
2	2	MIDEAST		C	1	0	L	—	N	Y	N	N	Y	Y
3	2	CENTRAL		C	1	0	L	—	N	Y	N	N	Y	Y
4	2	MIDWEST		C	1	0	L	—	N	Y	N	N	Y	Y
5	2	WEST		C	1	0	L	—	N	Y	N	N	Y	Y

PFKEYS: 6=+FLD 18=-FLD 4=+LINE 5=CUT 19=FLD_HELP 17=PASTE 16=-LINE 13=PRINT

EDITREGION Rule

When you press PF4, the following rule returns the name of the field where the cursor is placed and calls STE to display the contents of the table for editing:

```

RULE EDITOR ==>
EDITREGION;
LOCAL REG;
-----
REG = CURSORFIELD('EMPLOYEE_SCR_C');
CALL STE('EMPLOYEES(' || REG || ')');
-----

```

Data for Editing

When you place the cursor beside Midwest and press PF4, the following data appears for editing:

```

BROWSING TABLE : EMPLOYEES(MIDWEST)
COMMAND ==>

```

EMPNO	LNAME	POSITION	MGR#	DEPTNO	SALARY
22001	DRABEK	CUST SUPPORT	56112	30	900.00
22007	ROEDER	CUST SUPPORT	56112	30	900.00
30058	HOEGSON	PRE-SALES	37219	20	675.00
34111	TERAMURA	PRE-SALES	37219	20	710.00
34121	LEES	CUST SUPPORT	56112	30	700.00
36162	MORANG	JR OPERATOR	44798	80	575.00
41001	CROFTON	TECH WRITER	80002	70	675.00
41007	STEVENSON	EDUCATOR	80002	60	700.00
41009	SMITH	TESTER	79912	50	600.00
44385	SOUZA	SALES	37219	10	719.00
44622	SAUNDERS	ACCOUNTANT	98895	40	800.00
51111	HRODEK	ANALYST	79912	50	710.00
51121	CANNON	ANALYST	79912	50	700.00
51162	KIMURA	JR PROGRAMMER	79912	50	575.00
61219	WONG	SENIOR ANALYST	79912	50	820.00
61385	DHILLON	EDUCATOR	80002	60	685.00
61622	SCHULTZ	SENIOR ANALYST	79912	50	800.00

PFKEYS: 1=HELP 5=FIND NEXT 9=RECALL 18=EXCLUDE 13=PRINT 3=END 14=EXPAND

[At TOP](#)

STEBROWSE

Views the contents of a TIBCO Object Service Broker table. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	BR browse table option	Type <i>input</i> <Enter>
	EX Execute Rule option	Type STEBROWSE(<i>input</i>) <Enter>
	COMMAND prompt	Type EX STEBROWSE(<i>input</i>) <Enter>

Where:

<i>input</i>	A string containing the table name (and parameters, if any).
--------------	--

- Usage Notes**
- If you do not supply a value for *input*, executing STEBROWSE displays a list of tables that can be browsed.
 - STEBROWSE must be executed with the workbench option BROWSE set to Y.
 - [BROWSER](#) is the version of this tool called from within a rule.

See Also [Chapter 2, Using User Exits in Workbench Tools, on page 25](#) about using user exits with the Table Browser.

TIBCO Object Service Broker Managing Data for information on STEBROWSE.

SUBSTRING

Returns a selected portion of a string. (F)

Invocation `subsequence = SUBSTRING(string, start, length)`

<i>subsequence</i>	On return, contains the selected portion of the string. Its syntax is the same as <i>string</i> except that, if <i>string</i> is C (fixed-length character string), <i>subsequence</i> becomes V (variable-length character string).
<i>string</i>	The string to select characters from. Its syntax can be C, UN (Unicode), V, or W (double-byte character).
<i>start</i>	An integer specifying the position of the first character of the subsequence. Its syntax is B (binary) with length 4. Its value must be greater than zero. A value of 1 indicates the first character of the string.
<i>length</i>	An integer specifying the number of characters to select. Its syntax is B (binary) with length 4. Its value must be equal to or greater than zero.

- Usage Notes**
- If *start* plus *length* is greater than the length of *string*, SUBSTRING raises an error condition.
 - Errors detected by SUBSTRING are not recoverable.

Example This rule extracts a subsequence from a string and prints to the message log:

```

RULE EDITOR ==>                                SCROLL: P
SUBSTRING_1;
_ LOCAL SOURCE_STRING, SUBSEQ_STRING;
_ -----
_ -----+-----
_ SOURCE_STRING = 'THIS IS A STRING';              | 1
_ SUBSEQ_STRING = SUBSTRING(SOURCE_STRING, 2, 3);   | 2
_ CALL MSGLOG('THE SUBSTRING IS: ' || SUBSEQ_STRING); | 3
_ CALL MSGLOG('THE SOURCE STRING IS STILL: ' || SOURCE_STRING); | 4
_ -----

```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ===>                                SCROLL ===> P  
THE SUBSTRING IS: HIS  
THE SOURCE STRING IS STILL: THIS IS A STRING
```

\$SYSTEMDATE

Returns the date when \$SYSTEMDATE is called based on the local machine's time zone in which the Execution Environment is running. (F)

Invocation date = \$SYSTEMDATE

date On return, contains the date when \$SYSTEMDATE is invoked.

- Usage Notes**
- The date is returned as a semantic type date.
 - The result appears in the default date format for the installation.

Example The following rule, which uses PAYMENT as an argument, records the amount received and the current date in the table RECEIPTS:

```

RULE EDITOR ==>
RECEIVED(PAYMENT);

```

SCROLL: P

```

- -----
- |
- |-----+-----
- | RECEIPTS.AMOUNT = PAYMENT;          | 1
- | RECEIPTS.DATE = $SYSTEMDATE;        | 2
- | INSERT RECEIPTS;                    | 3
- |-----
- -----

```

Resulting Output

When the rule is executed with the argument 67.89, the bottom occurrence is inserted in the table:

```

EDITING TABLE : RECEIPTS
COMMAND ==>

```

SCROLL: P

```

      NUM      DATE      AMOUNT
- -----
-      1 2000-02-23      34.56
-      2 2000-03-10      78.56
-      3 2000-03-18      23.00
-      4 2000-03-18      67.89
-

```

SYSTEMLIB

Returns the name of the currently designated system library. (F)

Invocation `result = SYSTEMLIB`

<code>result</code>	On return, contains the name of the currently designated system library. This value is a typeless string of syntax C with a maximum length of 8.
---------------------	--

Usage Note The default system library is COMMON.

Example

<pre> RULE EDITOR ==> WHAT_LIBRARY; _ LOCAL LIB; _ _ _ LIB = SYSTEMLIB; _ CALL MSGLOG('THE CURRENT SYSTEM LIBRARY IS ' LIB); _ _ _ _ _ _ _ _ _ </pre>	<pre> SCROLL: P 1 2 </pre>
---	----------------------------

Pressing PF2 after executing the rule displays the following screen:

<pre> ----- INFORMATION LOG ----- COMMAND ==> THE CURRENT SYSTEM LIBRARY IS COMMON . </pre>	<pre> SCROLL: P </pre>
--	------------------------

TABLEPRINT

Prints the contents of a table or of a set of joined tables. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	PT print table option	Type <i>tablespect</i> <Enter>
	EX Execute Rule option	Type TABLEPRINT(<i>tablespect</i>) <Enter>
	COMMAND prompt	Type PT <i>tablespect</i> <Enter>

Where:

<i>tablespect</i>	The name of the table and any parameters to print.
-------------------	--

- Usage Notes**
- If you supply a value for *tablespect*, the table is sent to the printer.
 - If you do not supply a value for *tablespect*, the Table Print screen appears when TABLEPRINT is executed. An example of this screen is illustrated here:

```

-----  TABLE  PRINT  UTILITY  -----
PAGE LENGTH: 60      PAGE WIDTH: 132    PRT OR SCR: PRT
TABLE :
```

```
PFKYS:3=PRINT EXIT 12=CANCEL 13=PRINT REPEAT 6=ADD TABLE 22=DELETE TABLE
```

- The following function keys are recognized while the screen appears:

Enter	Checks that tables exist.
PF3	Prints and exits.

PF6	Adds a table to the join.
PF12	Exits without printing.
PF13	Prints and repeats.
PF22	Removes a table from a join.

- The print arguments are set at the top of the screen. The defaults are Page Length 60, Page Width 132 (which is the only supported width for a printed page), and PRT. The normal values for **SCR** are Page Length 25 and Page Width 80.



Fields that do not fit within a page are not printed.

- To print a table, enter its name in the **TABLE** field and press PF3.
- To print two or more tables successively, enter the name of one table and press PF13. The table is printed and the screen appears again. Another table name can be entered, the print arguments can be changed, and the process can be repeated.
- To cancel without printing, press PF12.

Joining Tables for Printing

Tables can be joined for printing. The following conditions apply:

- The join is based on equality of values in one field of each table.
- Fields that are to be compared must be specified.
- At least one of the two fields that are being compared must be a primary key.
- A parameter value must be supplied for a parameterized table. For example:
TABLE: EMPLOYEE_DEPT(10).
- Such a table can be included in a join where the selection is based on its parameter values. In this case, the parameter is specified in the form *t.f* on the **JOIN TABLE** field.
- Each value of the field *f* is used, so that several instances of the table can be printed immediately.
- The table *t* must already be named in the **TABLE** or **JOIN TABLE** fields.
- When several tables are joined for printing, both matching of fields and matching of parameters can be used in the same join.

Examples Statements

- The statement: `EX TABLEPRINT('EMPLOYEE')` prints the contents of the table `EMPLOYEE`.
- The statement: `EX TABLEPRINT()` displays the `TABLEPRINT` screen.

Use of Joins

The following example demonstrates the use of joins when printing tables. The table `DEPARTMENT` contains the department number in a field called **DEPTNO**. The table `EMPLOYEE` contains the department number for each employee in a field also called **DEPTNO**.

To join these tables for printing, type the name of one table, and then press PF6 to be able to enter the name of the second table and the fields to compare, in a screen like this:

```
-----  TABLE  PRINT  UTILITY  -----
PAGE LENGTH: 60      PAGE WIDTH: 132    PRT OR SCR: PRT
TABLE: DEPARTMENT
JOIN TABLE
MATCH FIELD:                WITH (T.F): DEPARTMENT.
```

`TABLEPRINT` automatically fills in the first table in the **WITH (T.F)** field. Fill in the information about the table you want to join (for example, the `EMPLOYEE` table), as shown in this screen:

```
-----  TABLE  PRINT  UTILITY  -----
PAGE LENGTH: 60      PAGE WIDTH: 132    PRT OR SCR: PRT
TABLE: DEPARTMENT
JOIN TABLE  EMPLOYEE
MATCH FIELD: DEPTNO                WITH(T.F): DEPARTMENTS.DEPTNO
```

Conditions for a Join

The following conditions apply to a join:

- At least one of the two fields that are being compared must be a primary key. In this example, `DEPTNO` is the primary key of departments.
- More than two tables can be included in a join.

- The table named in the field **WITH(T.F)** must already be named in the **TABLE** field or it must be named in a **JOIN TABLE** field above the one that names the table currently being added to the join.
- All the fields of all the tables are printed.
- The primary key of the first table named appears on the left of each page.

Printing Parameterized Tables

- The following example demonstrates the printing of a parameterized table.
- The table EMPLOYEE_DEPT, parameterized by DEPTNO, contains a listing of all the employees.
 - A list of all the department numbers is contained in the DEPARTMENT table in the field **DEPTNO**.

The joined tables are printed as follows:

----- TABLE PRINT UTILITY -----		
PAGE LENGTH: 60	PAGE WIDTH: 132	PRT OR SCR: PRT
TABLE: DEPARTMENT		
JOIN TABLE	EMPLOYEE_DEPT(DEPARTMENT.DEPTNO)	
MATCH FIELD:	WITH (T.F): DEPARTMENT.	

In this case, the **MATCH FIELD** and **WITH (T.F)** fields are ignored.

Resulting Output

PRINTING TABLE(S): DEPARTMENT EMPLOYEE_DEPT(DEPARTMENT.DEPTNO)			
DEPTNO	DNAME	EMPNO	LNAME
-----	-----	-----	-----
10	ACCOUNTING	1011	CROFTON
10	ACCOUNTING	1007	STEVENSON
10	ACCOUNTING	1009	SMITH
.....
20	SALES	1121	KIMURA
20	SALES	1622	SCHULTZ
.	.	.	.
.	.	.	.
.	.	.	.

TAILSTRING

Returns the tail portion of the string. (F)

Invocation `tail = TAILSTRING(string, length)`

<i>tail</i>	On return, contains the source string with the specified number of leading characters removed. Its syntax is the same as <i>string</i> except that, if <i>string</i> is C (fixed-length character string), <i>tail</i> becomes V (variable-length character string).
<i>string</i>	The string to remove characters from. Its syntax can be C, UN (Unicode), V, or W (double-byte character).
<i>length</i>	An integer specifying the number of characters to remove from the front of the string. Its syntax is B (binary) with length 4.

- Usage Notes**
- If *length* is larger than the length of *string*, an empty string is returned.
 - If *length* is negative, an empty string is returned.

Example This rule extracts a tail string from a string and prints both strings to the message log:

```

TAILSTRING_1;
_ LOCAL SOURCE_STRING, TAIL_STRING;
_ -----
_ -----+-----
_ SOURCE_STRING = 'THIS IS A STRING';                | 1
_ TAIL_STRING = TAILSTRING(SOURCE_STRING, 12);        | 2
_ CALL MSGLOG('THE TAIL STRING IS: ' || TAIL_STRING); | 3
_ CALL MSGLOG('THE SOURCE STRING IS STILL: ' ||      | 4
_   SOURCE_STRING);
_ -----

```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE TAIL STRING IS: RING
THE SOURCE STRING IS STILL: THIS IS A STRING

```

TED

Displays a table for text editing. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type TED(<i>text_input</i>) <Enter>
	COMMAND prompt	Type EX TED(<i>text_input</i>) <Enter>

Where:

text_input A string or the name of a table.

Usage Notes

- If you do not supply a value for *text_input*, executing TED displays a screen prompting for a value.
- After TED is invoked, a table appears for text editing. If a string is supplied, TED displays the table @TEXT(*userid,name*). If a table name is typed, TED displays the table.
- The table named in *text_input* must be a text table. A text table can be parameterized and it must contain the following two fields:

NUMBER	A primary key field with a length of 4, a syntax of B (binary), and a semantic type of I (identifier).
---------------	--

LINE	A field with a syntax of V (variable-length character string) and a semantic type of S (string).
-------------	--

Line Commands

The following line commands are available:

D	Deletes this line.
----------	--------------------

I	Inserts after this line. This inserts a new blank line on the screen. Another line is inserted each time you press Enter, as long as the cursor is on the previous line. Using TED you can insert blank lines.
----------	--

R	Replicates this line after itself.
----------	------------------------------------

S	Splits this line. The line splits where the cursor is positioned. This command is not sensitive to tokens. Words can be split.
C	Copies this line. The line copies to a destination as explained below.
M	Moves this line. The line moves to a destination as explained below.
A	Destination for move or copy occurs after this line.
B	Destination for move or copy occurs before this line.



If **A** or **B** marks the destination for a move or copy command, the marked destination is used. Otherwise, the destination is assumed to be after the line where the cursor is placed.

**Primary
Commands**

A command line exists at the top of both the formatted and unformatted screens. The primary command line accepts the following primary commands (except where noted otherwise):

CANCEL	Exits TED without saving changes to the text.
CH or CHANGE <i>old new</i> [ALL]	<p>The first occurrence of string <i>old</i> after the cursor position is replaced with the string <i>new</i>. If the option ALL is included, every string after the cursor changes. Tokens, indicated by spaces (' '), must separate strings in the command. This separator token must appear three times on the command line. This command is sensitive to case.</p> <p>This command is valid only on the unformatted text screen.</p>

COPY (<i>source</i>)	<p>Copies text from an existing text table into the document already on the screen. The source can be one of:</p> <ul style="list-style-type: none"> 1 – A table name, if un-parameterized. 2 – A table name, including parameter values, if parameterized. 3 – The second parameter of the table @TEXT. <p>The text is positioned:</p> <ul style="list-style-type: none"> 1 – At the top of the document, if the cursor is on the primary command line. 2 – Before or after a line containing B (before) or A (after) in the line command field. <p>Only valid on the unformatted text screen.</p>
DELETE	<p>The entire text is deleted from the database. You are prompted to confirm the deletion. This command is valid only on the unformatted text screen.</p>
F or FIND <i>string</i>	<p>The first occurrence of <i>string</i> after the cursor is located and highlighted. This command is sensitive to case.</p>
SAVE (<i>tablename</i>)	<p>Saves the text without leaving TED. If no <i>tablename</i> is given, the source table is overwritten. <i>tablename</i> follows the conventions of TED's argument and can be a name or a table. Using this command, you can make a copy of your text by saving it under a new name.</p>

PF Keys

The following PF keys are recognized by TED:

PF3	Saves the text and leave the editor. A script check is performed.
PF5	Toggles between formatted and unformatted text.
PF9	Repeats the last primary command.
PF12	Leaves the editor without saving changes to the text.
PF13	Prints the version of the text that you are viewing.
PF22	Deletes the text. Refer to the remarks for the DELETE primary command above.

PF22 is available only from the unformatted version of the text.

Example The following application uses TED to write notes about employees. The initial application screen is EMPLOYEE_SCR_B, which contains the scrollable screen table EMPLOYEE_NAME:

Date: 2000-03-17		Employees by Region	
Employee Name		Employee#	
— DRABEK		22001	
— ROEDER		22007	
— HOEGSON		30058	
— TERAMURA		34111	
— LEES		34121	
— MORANG		36162	
— CROFTON		41001	
— STEVENSON		41007	
— SMITH		41009	
— SOUZA		44385	
— SAUNDERS		44622	
— HRODEK		51111	
— CANNON		51121	
— KIMURA		51162	
— WONG		61219	

FCNKEYS: 2=GET INFO 12=EXIT 4=NOTES

MAKE_NOTE Rule

When you type an S in the selection field beside the employee name and press PF4, the following rule is executed:

RULE EDITOR ==>		SCROLL: P	
MAKE_NOTE;			
_ LOCAL EMP;			

		+	
_ GET EMPLOYEE_NAME('EMPLOYEE_SCR_B') WHERE SELECT = 'S';			1
_ EMP = EMPLOYEE_NAME.EMPNAME;			2
_ CALL TED('EMP_NOTES(' EMP ')');			3

Table for Editing

This rule displays the following table (in this case the table is for employee ROEDER and contains previously entered text, including **SCRIPT** commands):

Text Editor ==>	Scroll: P
Editing: EMP_NOTES(ROEDER)	

Responsible for:	
<ul style="list-style-type: none">	
Performing site assessment	
Writing proposal	
Presenting proposal to board of directors	
Due Date for activities: Sept. 13	

PFKEYS: 12=CANCEL 22=DELETE 3=SAVE 5=SCRIPT 9=REPEAT CMND

Scripted Text

Pressing PF5 from this screen displays the scripted text:

Command ==>	Formatted Output	Scroll: P
		page 1
	Responsible for:	
	-> Performing site assessment	
	-> Writing proposal	
	-> Presenting proposal to board of directors	
	Due Date for activities: Sept. 13	

PFKEYS: 12=CANCEL 13=PRINT 3=SAVE 5=EDIT 9=FIND NEXT

TEXTSETUP

Defines a setup for formatting a text document. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type TEXTSETUP(<i>setupname</i>) <Enter>
	COMMAND prompt	Type EX TEXTSETUP(<i>setupname</i>) <Enter>

Where:

setupname The name of the setup to use.

Usage Notes

- If you do not supply a value for *setupname*, invoking TEXTSETUP displays a screen prompting for a value.
- After TEXTSETUP is invoked, a screen appears for defining a new setup or modifying an existing setup.
- Use the **.setup** *setupname* **SCRIPT** command at the beginning of a text file to have SCRIPT use a particular setup to format a file.

TEXTSETUP Screens

TEXTSETUP contains three screens:

Page Setup screen	Defines the page layout.
Top Title screen	Defines the title lines for the setup.
Bottom Title screen	Defines the bottom lines for the setup.

Setups Supplied with TIBCO Object Service Broker

TIBCO Object Service Broker supplies four predefined setups:

Default	In effect whenever formatting begins and if no setup is specified. It turns formatting on and sets the page setup to suit the 24 line screen.
Screen	Sets a larger page size intended to fill a screen.
Help	Sets a page without a page number and is used by the Screen Definer to display both field- and screen-level help.
Print	Sets a page that can print in portrait form through the use of a print function that you provide later.

These predefined setups cannot be overwritten (saved) or deleted.

Page Setup Screen

After entering a new or existing text setup name, a screen similar to the following appears:

TEXT SETUP: DEFAULT		COMMAND ==>
This is the page definition; change any field as desired.		
+-----+		
:	< ADjust = 5 >	< LineLength = 60 >
:	1L 1C 1R	Top Margin has 2 line(s).
P	2L 2C 2R	The first three lines can have titles
a	3L 3C 3R	in the positions shown. PF5 to set.
+-----+		
g		
e		
L	This area is	Page Length is the number of lines
e = 23	used for the	used for the text and top and
n	formatted text.	bottom margins.
g		
+-----+		
t		
h		Bottom Margin has 2 line(s).
:	3L 3C 3R	Last three lines can have titles
:	2L 2C 2R	in the positions shown. PF6 to set.
:	1L 1C 1R	
+-----+		
Format Parameters --		Format on: Y Justify on: Y Line Spacing = 1
		PageNumbering: Y Bullet Characters = ->
PFKEYS: 3=SAVE 5=TOP TITLES 6=BOTTOM TITLES 12=CANCEL 22=DELETE		

Use the Page Setup screen to define the page layout; it contains formatting parameters for a page. You can change these parameters as required. The name of the setup is shown at the top of the screen in the **TEXT SETUP** field. You can change the name and save the setup under a new name.

Formatting Parameters

Each parameter name contains two capitalized letters. These letters indicate the SCRIPT command that sets the formatting. Refer to SCRIPT page layout commands for more information.

The formatting parameters are as follows:

Primary Commands and PF Keys for the Page Setup screen

These primary commands and PF keys are available from the Page Setup screen:

Command	PF Key	Function
SAVE	PF3	Saves the definition and exit.
TOP	PF5	Displays the TOP Titles screen.
BOTTOM	PF6	Displays the BOTTOM Titles screen.
CANCEL	PF12	Cancels the definition and exit.
DELETE	PF22	Deletes the definition. You are prompted to confirm the deletion.

Top Titles Screen

You use the TOP Titles screen to define the title lines for the setup. You can change the default values for a user-defined setup. Press PF5 to display this screen. The following shows the screen that appears for DEFAULT setup:

TEXT SETUP: DEFAULT
COMMAND ==>

TOP Titles

Enter titles. If you want the same title on all
pages, enter it for both odd and even numbered pages.

Even Numbered Pages

Odd Numbered Pages

1L =
1R = page %
2L =

1L =
1R = page %
2L =

2R =	2R =
3L =	3L =
3R =	3R =

Centre Titles

1C =
2C =
3C =

PFKEYS: 3=SAVE 12=CANCEL

Title lines contain the following characteristics:

- Lines are numbered from the top down.
- Lines can be entered for even, odd, or all pages. To enter values for all pages, title lines must be entered for both even and odd pages.
- Odd and even pages are treated as separate entities.
- Center titles apply to both even and odd numbered pages.
- Each title line can be left justified, right justified, or centered within each page type.
- You can enter up to three title lines, if there is enough room in the margin.
- A title line can start with **.date str**. The variable *str* is any date format acceptable to **\$TRXDATE**. This is replaced with the current date in the format you specify.
- The percent sign (%) is entered as a default value (for the first line, right side (1R) even and odd pages). This symbol is converted to a page number if page numbering is ON. The symbol does not print if page numbering is OFF. Refer to SCRIPT page layout commands for more information on page numbering.

Bottom Titles Screen

Use the BOTTOM Titles screen to define bottom lines for the setup. You can change the default values for a user-defined setup. Press PF6 to display this screen. The following figure shows the screen that appears for DEFAULT setup:

TEXT SETUP: DEFAULT
COMMAND ==>

BOTTOM Titles

Enter titles. If you want the same title on all pages, enter it for both odd and even numbered pages.

Even Numbered Pages

Odd Numbered Pages

	:		2L	2C	2R		in the positions shown. PF6 to set.
	:		1L	1C	1R		

```

Format Parameters  --  Format on: Y  JUsTify on: N  Line Spacing = 2
                        PageNumbering: Y  Bullet Characters = *
PFKEYS: 3=SAVE 5=TOP TITLES 6=BOTTOM TITLES 12=CANCEL 22=DELETE

```

Unscripted Text

The following is an example of unscripted text created using the [TED](#) tool. The first line instructs SCRIPT to use the setup EXAMPLE:

Text Editor =====>

Scroll: P

Editing: @TEXT(BZDD1, EXAMPLE)

```

-----
- .setup example
- .cc.textsetup example
- .p.This example illustrates how the setup determines the format of
- scripted text. Some of the qualities that the setup controls are:
- .ul.
- .li.Line length
- .li.Justification
- .li.Page length
- .li.Bullet characters
- .p.These are easily set using the TEXTSETUP tool.

```

PFKEYS: 12=CANCEL 22=DELETE 3=SAVE 5=SCRIPT 9=REPEAT CMND

Scripted Text

The scripted text appears as follows:

Command =====> Formatted Output

Scroll: P

page 1

TEXTSETUP EXAMPLE

This example illustrates how the setup determines the

format of scripted text. Some of the qualities that the setup controls are:

- * Line length
- * Justification
- * Page length
- * Bullet characters

page 2

These are easily set using the TEXTSETUP tool.

TIME

Returns a string containing the time of the day when the transaction started. (F)

Invocation string = TIME

string On return, contains the time. Its syntax is C (fixed-length character string) with length 8.

Usage Notes The returned string is in the form HH:MM:SS, for example, 23:59:59.

Example The following rule determines the time of day when the transaction started and prints it to the message log:

```
TIME_1;
_ LOCAL TIME_OF_DAY;
-----
_
_ -----+-----
_ TIME_OF_DAY = TIME;                               | 1
_ CALL MSGLOG('THIS TRANSACTION WAS STARTED AT: ' || | 2
_   TIME_OF_DAY);                                     |
_ -----
```

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THIS TRANSACTION WAS STARTED AT: 08:28:21
```

\$TOCPRINT

Prints the table of contents. (C)

Invocation `CALL $TOCPRINT(fill_char)`

<i>fill_char</i>	The character to be printed to fill the space between the end of the section name and the beginning of the page number. Its syntax is V (variable-length character string).
------------------	---

- Usage Notes**
- The print arguments must be previously set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to `$TOCPRINT`.
 - Construct the table of contents using [\\$TOCPUT](#).
 - *fill_char* is repeated as many times as fits.

Exceptions

LOGLIMIT	Raised if too much output is sent to the message log.
ROUTINEFAIL	Raised if <code>\$TOCPRINT</code> is not preceded by a call to <code>\$RESETPRINT</code> or <code>\$SETPRINT</code> .

Example The following rule builds a table of contents and prints it to the message log:

```
TOCPRINT_1;
```

```

-
- -----
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N');          | 1
- CALL $TOCPUT('SECTION ONE', 1, 'Y');            | 2
- CALL $BLANKPAGE('N');                          | 3
- CALL $TOCPUT('SECTION TWO', 1, 'Y');            | 4
- CALL $BLANKPAGE('N');                          | 5
- CALL $TOCPRINT(' ');                          | 6
- -----
-

```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P

```


----- NEW PAGE -----

----- NEW PAGE -----

----- NEW PAGE -----

CONTENTS
=====

SECTION ONE	1
SECTION TWO	2

\$TOCPUT

Puts a line in the table of contents. (C)

Invocation `CALL $TOCPUT(section_name, spacing, numbering_yn)`

<i>section_name</i>	The string that appears in the table of contents as the section name. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
<i>spacing</i>	An integer specifying the number of lines between the current table of contents entry and the next. Its syntax is B (binary) with length 2.
<i>numbering_yn</i>	Specifies whether the page number should be printed: Y – Prints page numbers. N or " (NULL) – Do not print page numbers. Its syntax is C with length 1.

Prerequisites The print arguments must be set with a call to [\\$SETPRINT](#) or [\\$RESETPRINT](#) before a call to \$TOCPUT.

Usage Notes

- Use [\\$TOCPRINT](#) to print the table of contents.
- *section_name* cannot exceed the page width.
- Four character spaces are reserved in *section_name* for the page number, even if it is not printed.
- *spacing* can take on values greater than or equal to 0 and less than or equal to the page length set by [\\$SETPRINT](#) or [\\$RESETPRINT](#).
- If *n* is the number specified for *spacing*, *n*-1 spaces are printed. For example, if *n*=2, one blank line is printed between each content line.

Exceptions

RANGERROR	<p>Raised for one of the following reasons:</p> <p><i>spacing</i> – Is zero or a negative number.</p> <p><i>spacing</i> – Is greater than or equal to length (where length is the page length set by \$RESETPRINT or \$SETPRINT).</p> <p><i>numbering_yn</i> – Is neither Y nor N. An empty string (") is valid and treated as N.</p>
ROUTINEFAIL	Raised if \$TOCPUT is not preceded by a call to \$SETPRINT or \$RESETPRINT.
STRINGSIZE	Raised if the length of <i>section_name</i> exceeds <i>width</i> -4 (where <i>width</i> is the page width set by \$SETPRINT or \$RESETPRINT).

Example The following rule builds a table of contents and prints it to the message log:

```
TOCPUT_1;
-
- -----
-                                     +-----+
- CALL $SETPRINT(10, 70, 1, 'SCR', 'N');           | 1
- CALL $TOCPUT('SECTION ONE', 1, 'Y');             | 2
- CALL $TOCPUT('SECTION TWO', 2, 'N');             | 3
- CALL $TOCPUT('SECTION THREE', 1, 'Y');           | 4
- CALL $TOCPRINT(' ');                             | 5
- -----
-
```

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P

----- NEW PAGE -----

                                CONTENTS
                                =====

SECTION ONE . . . . . 1
SECTION TWO

SECTION THREE . . . . . 1
```


TOKEN

Parses an input string and returns the first token and the string with the token removed. (F)

Invocation `token_string = TOKEN(string)`

<code>token_string</code>	The token returned. Its syntax is V (variable length character).
<code><i>string</i></code>	The character string to parse. On return, it contains <i>string</i> with the token removed. Its syntax must be C (fixed-length character), UN (Unicode), or V (variable-length character) and its semantic type must be S (string).

Usage Notes

- The input *string* for TOKEN must be a local variable or a field of a table.
- TOKEN parses *string* until it encounters a space (' '), a special character, a literal notation (r', R', u', U', x', or X'), or one of the following operators: ¬, &, |, *, /, **, +, -, =, ||, <, >, <=, >=, or ¬=. It then returns the first token and the *string* with the token removed. If *string* begins with an operator, the operator is the token returned.
- Valid tokens are: identifiers, positive numbers, quoted strings, Unicode literals, raw data literals, hexadecimal literals, operators, and special characters. For additional information about these valid tokens, refer to *TIBCO Object Service Broker Parameters*.
- In quoted strings, two single quotation marks are converted to a single quotation mark.
- If *string* is of syntax UN and the token to be returned contains any character that cannot be coerced to syntax V, *token_string* is a Unicode literal representing the value of the token.
- For more complex manipulation of tokens, refer to the [PARSE](#) tool.

Exceptions

MISMATCHEDQUOTES	Raised if there are unmatched quotation marks in the argument <i>string</i> .
-------------------------	---

Examples The following rule prints the returned token and truncated string to the message log.

RULE EDITOR ==>		SCROLL: P
TOKEN_EXAMPLE(INPUT);		
_ LOCAL TOK, STRING;		
_ -----		
_ -----		
_ STRING = INPUT;		1
_ TOK = TOKEN(STRING);		2
_ CALL MSGLOG('FIRST TOKEN IS ' TOK);		3
_ CALL MSGLOG('RETURNED STRING IS ' STRING);		4
_ -----		

Resulting Output

If it is executed with the argument '10-12-2000', the following message log is produced:

----- INFORMATIONAL MESSAGE LOG -----	
COMMAND ==>	SCROLL ==> P
FIRST TOKEN IS 10	
RETURNED STRING IS -12-2000	

Example that Completely Parses a String

The following rules completely parse a string into tokens:

RULE EDITOR ==>		SCROLL: P
TOKEN_EXAMPLE2(INPUT);		
_ LOCAL TOK, STRING;		
_ -----		
_ -----		
_ STRING = INPUT;		1
_ TOK = TOKEN(STRING);		2
_ CALL MSGLOG('TOKEN IS ' TOK);		3
_ CALL TOKEN_EXAMPLE3(STRING);		4
_ -----		

RULE EDITOR ==>		SCROLL: P
TOKEN_EXAMPLE3(STRING);		
_ -----		
_ -----		
_ STRING = '';		Y N
_ -----		
_ CALL TOKEN_EXAMPLE2(STRING);		1

Resulting Output

If TOKEN_EXAMPLE2 is run with the argument 10-12-2000, the following message log is produced:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>
TOKEN IS 10
TOKEN IS -
TOKEN IS 12
TOKEN IS -
TOKEN IS 2000
SCROLL ==> P
```

Example With Processing of Tokens

The following rules retrieve the keywords for a table, separate the keywords, and insert the name of the table associated with the keyword in a table instance for the keyword.

TABLE_KEYWORD Table

The first table, which contains the keywords for table descriptions, is TABLE_KEYWORD:

BROWSING TABLE: TABLE_KEYWORD	
COMMAND ==>	
TABLE	KEYWORDS
-----	-----
EMPLOYEE	PERSONNEL , NAME , NUMBER , DEPARTMENT , MANAGER , SALARY
DEPARTMENT	DEPARTMENT , NAME , NUMBER
MANAGER	MANAGER , NAME , NUMBER

TKEYWORDINDEX Table

The second table, TKEYWORDINDEX, is parameterized by KEYWORD. Each table instance lists the table names that have the parameter value as a keyword. An example is shown here.

```
BROWSING TABLE:      TKEYWORDINDEX (NAME)
COMMAND ==>
```

```
TABLE
-----
EMPLOYEE
DEPARTMENT
MANAGER
```

KEYWORD_PARSE Rule

The parent rule is KEYWORD_PARSE. It retrieves each occurrence of the TABLE_KEYWORD table, and calls the KEYWORD_INDEX rule. The arguments for KEYWORD_INDEX are:

- The name of the table that is stored in the **TABLE** field of the TABLE_KEYWORD table.
- The first token (that is, the first keyword) for the table, which is extracted with TOKEN:

```
KEYWORD_PARSE;
LOCAL STR;
-----
FORALL TABLE_KEYWORD:                                | 1
  STR = TABLE_KEYWORD.KEYWORDS;                      |
  CALL KEYWORD_INDEX(TABLE_KEYWORD.TABLE, TOKEN(STR)); |
END;                                                    |
-----
```

KEYWORD_INDEX Rule

KEYWORD_INDEX checks for valid keywords, and then inserts the table name into the TKEYWORDINDEX table. It also uses TOKEN to extract the next keyword.

```
KEYWORD_INDEX(NAME, TOK);
-----
TOK = ',';      | Y N N N
TOK = ''';     |   Y N N
STR = ''';      |     Y N
-----
TKEYWORDINDEX.TABLE = NAME; | 1 1
INSERT TKEYWORDINDEX(TOK);  | 2 2
CALL KEYWORD_INDEX(NAME, TOKEN(STR)); | 1 3
-----
```

Resulting Output

The result of running KEYWORD_PARSE against the table TABLE_KEYWORD is a TKEYWORDINDEX table with table instances such as DEPARTMENT, MANAGER, and NAME. If you browse the NAME table instance of the TKEYWORDINDEX table, you see the following:

BROWSING TABLE:	TKEYWORDINDEX(NAME)
COMMAND ==>	
	TABLE
—	-----
—	EMPLOYEE
—	DEPARTMENT
—	MANAGER

TO_UNICODE

Converts a raw data string encoded in an external code page to Unicode.(F)

Invocation `unistring = TO_UNICODE(rdstring, externalcodepage)`

<i>unistring</i>	On return, contains the string in UN (unicode) syntax.
<i>rdstring</i>	Contains a string of RD (raw data) syntax. The data encoding corresponds to one of the 16 possible user-defined external syntaxes XC01 to XC16.
<i>externalcodepage</i>	One of the values 'XC01' to 'XC16' representing the user-defined external syntax.

Example This rule converts a string encoded in external syntax XC02 to Unicode.

```

TESTTOUNI;
_ LOCAL RESULT;
_ -----
_ -----+-----
_ TESTUNI.RD = 'ABC DEF';                      | 1
_ RESULT = TO_UNICODE(TESTUNI.RD, 'XC02');      | 2
_ CALL ENDMSG(RESULT);                          | 3

```

Line 1 sets Raw Data field RD of table TESTUNI to a string "ABC DEF". Line 2 converts this string (encoded in external code page XC02) to Unicode. Line 3 displays the result: ABC DEF.

@TRACEMESSAGES

Records message traffic between the Execution Environment and the Data Object Broker. (TBL)

Table Definition

Parameters This table has two parameters: ADDRESS and LOCATION.

Fields The fields of the @TRACEMESSAGES table are as follows:

ID	This is a reserved field.
TRACE	Y means trace all messages in the @MESSAGETRACE table. This entry is ignored if MESSAGELOG =Y.
TABLE	Traces accesses only to the table named.
REQUEST	Traces accesses only of the named request type (for example, N=FORALLS), independent of TABLE field.
SHOWSYNC	Y means show SYNC messages.
COUNT	The maximum number of rows that the system creates in the @MESSAGETRACE and @MESSAGEDUMP SES tables. When the number of rows in the table reaches the number given in COUNT , the table is cleared and restarted.
DUMP	Use this field with caution: Y means create HEX dump of messages in table @MESSAGEDUMP. Ignored if MESSAGELOG =Y.
LOGOUT	This is a reserved field.
OUTDUMPLIMIT	When dumping, indicates the maximum number of bytes to show from the outbound message.
RETURNDUMPLIMIT	When dumping, indicates the maximum number of bytes to show from the inbound message.
MESSAGELOG	Y means create a HEX dump of all messages in the message log.

KEEPMESSAGELOG	Y means show all messages on the message log, including those normally erased by TIBCO Object Service Broker.
LOGTABLECALLS	This is a reserved field.

Constraints This facility is available only on a z/OS system.

Usage Notes Use a rule or the Table Editor to set the value of @TRACEMESSAGES.TRACE to Y and then run the applications to be traced. Message tracing can be further refined by modifying other fields of the @TRACEMESSAGES table.

Unless you specifically route messages to the message log or to the @MESSAGE DUMP session table, all messages are logged in the @MESSAGE TRACE session table. By analyzing the occurrences in the @MESSAGE TRACE table, you can determine the access paths requested by the Execution Environment for each table access. This information can be used to optimize your application by using different variations of application table access statements.



- Use **DUMP=Y** with caution, and only on the advice of your TIBCO Support representative.
- Setting **MESSAGELOG** to Y can be used to suppress inserts to the @MESSAGE TRACE and @MESSAGE DUMP tables. This would show that these accesses are not interfering with other processing. Use this option only on the advice of an TIBCO Support representative.
- Using **KEEPMESSAGELOG=Y** causes the message log to increase in size dramatically. Use this option only on the advice of your TIBCO Support representative.

Example The TRACEON sample rule illustrates how a typical TAM table call and message flow trace is activated. To capture TAM table calls and Execution Environment or Data Object Broker messages for a given session, execute the TRACEON rule. If you are interested in the calls for a particular table, modify the rule and specify a table name for the statement @TRACEMESSAGES.TABLE='tablename' (where *tablename* is the name of the table you are interested in). For results to this example, refer to @MESSAGE TRACE.

```
TRACEON;
-----
+-----
GET @TRACEMESSAGES(0);          | 1
@TRACEMESSAGES.TRACE='Y';      | 2
@TRACEMESSAGES.TABLE=NULL;     | 3
```

```

_ @TRACEMESSAGES.REQUEST=NULL; | 4
_ @TRACEMESSAGES.SHOWSYNC='Y'; | 5
_ @TRACEMESSAGES.COUNT=0; | 6
_ @TRACEMESSAGES.DUMP='Y'; | 7
_ @TRACEMESSAGES.OUTDUMPLIMIT=0; | 8
_ @TRACEMESSAGES.RETURNDUMPLIMIT=0; | 9
_ @TRACEMESSAGES.MESSAGELOG=NULL; | A
_ @TRACEMESSAGES.KEEPMESSAGELOG=NULL; | B
_ REPLACE @TRACEMESSAGES(0); | C
_ -----+-----
```

Examining the Results

To deactivate the trace and examine the results, run the TRACEOFF rule as shown below. Deactivate the TAM trace and then examine the contents of the @MESSAGETRACE and @MESSAGEDUMP tables using the Table Browser. If you want to print the contents of the @MESSAGEDUMP, edit out unprintable characters in the character portion of the trace prior to printing (use the Table Editor).

```

TRACEOFF;
-----+-----
GET @TRACEMESSAGES(0); | 1
@TRACEMESSAGES.TRACE='N'; | 2
@TRACEMESSAGES.TABLE=NULL; | 3
@TRACEMESSAGES.REQUEST=NULL; | 4
@TRACEMESSAGES.SHOWSYNC='N'; | 5
@TRACEMESSAGES.COUNT=0; | 6
@TRACEMESSAGES.DUMP='N'; | 7
@TRACEMESSAGES.OUTDUMPLIMIT=0; | 8
@TRACEMESSAGES.RETURNDUMPLIMIT=0; | 9
@TRACEMESSAGES.MESSAGELOG=NULL; | A
@TRACEMESSAGES.KEEPMESSAGELOG=NULL; | B
REPLACE @TRACEMESSAGES(0); | C
-----+-----
```

The TRACEON sample rule causes the collection of a generalized trace. Refer also to the information for @MESSAGEDUMP and @MESSAGETRACE.

\$TRXDATE

Returns the start date of the transaction that called this tool based on the local machine’s time zone in which the Execution Environment is running. (F)

Invocation date = \$TRXDATE

date	On return, contains the start date of the transaction.
------	--

- Usage Notes**
- The date is returned as a semantic type date.
 - The result is displayed using the default date format of the installation.

Example As part of a larger transaction that could run overnight, the following rule writes the starting date of the transaction to the message log:

```

      RULE EDITOR ==>
START_DATE;
_ LOCAL DATE;
_ -----
_ -----+-----
_ DATE = $TRXDATE;                               | 1
_ CALL MSGLOG('TRANSACTION STARTED: ' || DATE);   | 2
_ -----
```

\$TRXMODE

Retrieves the transaction mode of the current rule. (F)

Invocation result = \$TRXMODE

result The transaction mode of the current rule, either BROWSE or UPDATE.

Example	Rule
<pre>RULE EDITOR ==> GET_MODE; _ LOCAL A; _ ----- _ -----+----- _ A = \$TRXMODE; 1 _ CALL ENDMSG('The current rule is running in ' 2 _ A ' mode.'; _ -----</pre>	<pre>SCROLL: P</pre>

Output

Running GET_MODE could produce:
The current rule is running in BROWSE mode.

\$TYPECAST

Converts a variable according to the arguments supplied. (F)

Invocation `result = $TYPECAST(type, syntax, size, decimals, value)`

<i>result</i>	The variable cast according to the input arguments.
<i>type</i>	The semantic type to be used. This is a typeless 1-byte identifier of character syntax.
<i>syntax</i>	The syntax to be used. This is a typeless 3-byte identifier of character syntax.
<i>size</i>	The size to be used. This is a typeless 2-byte identifier of binary syntax with 0 decimal places. The size value must be greater than zero.
<i>decimals</i>	The decimals value to be used. This is a typeless 2-byte identifier of binary syntax with 0 decimal places. The decimals value must be greater than or equal to zero and must be consistent with the syntax requested.
<i>value</i>	The expression to be cast. This argument can be any semantic type, syntax, and size.

Exceptions

CONVERSION	Signaled if any of the input arguments <i>type</i> , <i>syntax</i> , <i>size</i> , or <i>decimals</i> is invalid.
-------------------	---

Example Rule

```

RULE EDITOR ==>
COERCE_ASSIGN(TO_TABLE, TO_FIELD, FROM_TABLE, FROM_FIELD);
--
-- -----
-- GET FIELDS(TO_TABLE) WHERE NAME = TO_FIELD;
-- (TO_TABLE).(TO_FIELD) = $TYPECAST(FIELDS.TYPE,
--   FIELDS.SYNTAX, FIELDS.LENGTH, FIELDS.DECIMAL,
--   (FROM_TABLE).(FROM_FIELD));
--
--
--
-- -----

```

SCROLL: P

Explanation

This rule assigns a field of a table to another field of another table, if there exists an allowed conversion between the source value and the destination value, irrespective of their semantic types. For example, binary quantity values can be assigned to binary identifier fields and binary date values to quantity binary fields. Assignments such as assigning a string containing letters to a numerical fields are not allowed.

```
CALL COERCE_ASSIGN('T1', 'CB4', 'T2', 'DB4');
```

performs this assignment:

```
T1.CB4 = T2.DB4;
```

ignoring the semantic type prohibition against it and leaving T1.CB4 with the internal representation of the date in field T2.DB4 (assuming that CB4 is a binary count field of length 4 and DB4 is a binary date field of length 4).

Refer to *TIBCO Object Service Broker Programming in Rules* for allowed arithmetic conversions.

@UNINSTALL

Requests that the specified component be uninstalled. (CE)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Developer's workbench	EX Execute Rule option	Type @UNINSTALL(<i>component</i>) <Enter>
	COMMAND prompt	Type EX @UNINSTALL(<i>component</i>) <Enter>
From a rule		Type @UNINSTALL(<i>component</i>)

Where:

<i>component</i>	The component you want to uninstall.
------------------	--------------------------------------

Usage Notes

- The argument *component* is the name of an instance of the @TOINSTALL table. This instance contains the names of objects, table instances, and occurrences included in the component.
- The user ID you use to run @UNINSTALL must have level-7 security clearance.

Exceptions

ROUTINEFAIL	Raised if @UNINSTALL cannot find the component.
--------------------	---



When you delete a component, its tables are also deleted, and any data they contain is lost.

Example The following example uses @UNINSTALL to uninstall a component.

```
@UNINSTALL(TIMELAPSE)
```

UNLOAD

Unloads definitions of valid TIBCO Object Service Broker object types from a source system to a z/OS data set or a Windows or Solaris file. Data from table object types could also be unloaded. (E)

Invocation Do one of the following:

From the...	Move the cursor to the...	And...
Administrator's workbench	UL Unload file option	Press Enter
Developer's workbench	EX Execute Rule option	Type UNLOAD <Enter>
	COMMAND prompt	Type EX UNLOAD <Enter>

UNLOAD of Definitions and Data Screen

Pressing Enter displays the screen shown here:

```

                                UNLOAD of Definitions and Data
File Name:                                Log msgs to: SCR
Location (for all objects):                Object Count: 0
Default Library      : USR40
Default Environment: 3270

Scroll Amount: P

Object      Object      Library or      |----- For Tables -----|
Name        Type        Environment   Type #Prms Defn? Data?  Inst      All
-----
-----
```

PFKEYS: 5=SELECT OBJECTS 6=INSTANCE SELECT 3=UNLOAD 12=CANCEL

After selecting all the definitions required, or in the case of tables, all the data or definitions, press PF3 to unload. This returns you to the workbench and a message appears indicating where the log of your activities is.

After unloading the items, use **LOAD** from within your target system to load the items. If you want to cancel the unload, press PF12 from within the Unload screen.

Top Section

The following describes the fields of the UNLOAD of Definitions and Data Screen:

File Name	Specifies the name of the file where the unloaded items are to be placed. It can be re-used at a later time with the contents being overwritten.
Location (for all objects)	If the objects to be unloaded are located on a remote node, type the name of the node where they are located.
Default Library	Enter the name of the default library from which rules are to be unloaded.
Default Environment	This field is currently not in use.
Log msgs to	Specifies the destination where the messages are logged. Press PF1 for a list of valid values.
Object Count	A protected field that displays the number of objects that are selected to be unloaded.

You can do one or both of the following to specify objects to be unloaded:

- Type values directly into the fields in the bottom portion of the screen.
- Press PF5 to display a selection screen. Refer to [Object Selection Screen on page 705](#) for more information.

Bottom Section

Use the bottom portion of the Unload screen to specify the objects to unload. This section contains the following fields:

Scroll Amount	Specifies the amount to scroll when you use the scroll keys PF7 and PF8. Valid entries are: P (page), H (half a page), M (maximum), <i>nn</i> (number of lines).
Object Name	The name of the object or objects to unload. Using the EOF key, you can also delete the name of objects you no longer require. You can also delete the name by pressing Delete, Backspace, space bar.
Object Type	The name of the object type for the object specified in the Object Name field. Press PF1 for a list of valid values.
Library or Environment	<p>Enter the name of the rules library from which to unload individual rules.</p> <p>Because you can unload both the definition and the data for objects of type Table, use an additional section entitled "For Tables" for Table object types. Values are entered by default into the fields Type, #Prms, Defn?, and Data?. You can modify the fields Defn?, Data?, and All Inst as follows:</p> <p>Defn? – The default is N (no), do not unload the definition of the table. You can change this to Y (yes), unload the definition.</p> <p>Data? – The default is N (no), do not unload the data of the table. You can change this to Y (yes), unload the data. If you change it to Y, you can select the instances that you require or leave it as the default for all instances.</p>

Instance Selection Screen

To select the instances, position your cursor on the table name that you are selecting and press PF6. A screen similar to the one here appears.

```
Place an "S" beside the parameter values for instances to be unloaded;
if NO values are selected, occurrences for ALL instances are unloaded.
```

```
TABLE:                                ( #parameters:    )    ...more ->

-> FIND/SELECT _____
```

(parm name) (op) (unquoted value or pattern)

----- Scroll:

PFKEYS: 10=LEFT 11=RIGHT 5=FIND 23=SELECT 6=DETAIL 16=UNSELECT 3=SAVE 12=CANCE

Fields

TABLE	The name of the table is entered by default.
#parameters	The number of parameters for the table is entered by default.
...more	A more indicator (-> or <-) shows if there is additional information to view to the left (PF10) or right (PF11) of the displayed screen.
FIND/SELECT	<p>Use this field to narrow your search for parameter values to select. Specify one parameter, an operator, and a parameter value in the blank spaces to the right of this field and press PF23. Your cursor is positioned on the first instance that meets this minimal criteria. Press PF5 to find the next instance that meets this criteria.</p> <p>You can specify only up to 30 characters for the parameter value. You can use the LIKE operator with the wild card character asterisk (*) to assist you for values that are longer.</p>
Scroll	You can specify the amount that you want to scroll when you use the scroll keys. Valid entries are: P (page), H (half a page), M (maximum), <i>nn</i> (number of lines).

The lower portion of the screen displays the parameter names and parameter values for the table. You use this portion to select the instances that you require. To select table instances, type an **S** in the line command field of the instances that you require and press Enter.

PF Keys

You can use the following PF keys in this screen:

PF3	Saves the selection and return to the Unload screen.
PF5	Finds the next instance that meets the selection criteria.
PF6	Displays a list of the full parameter values for the table instance that your cursor is positioned on. Use this if the parameter values are greater than 30 display characters each, which is the limit for this screen.
PF7	Scrolls up.
PF8	Scrolls down.
PF10	Scrolls left.
PF11	Scrolls right.
PF12	Cancels the selection and return to the Unload screen.
PF16	Clears all the instances that you previously selected.
PF23	Selects the instances that meet the selection criteria specified in the FIND/SELECT field.

Object Selection Screen

Pressing PF5 from the Unload screen displays the screen shown below. You can use this screen to select the objects to be unloaded.

O b j e c t S e l e c t i o n			
COMMAND ==>			
Location:		Select All: N	
Library (for RULES):		List Children: N	
Presentation Environment:			
+----- Selection Specification -----+			
	Attr	Op	Value

	NAME	_____	AND unspecified

	TYPE	=	_____	AND	attributes will	
	UNIT	_____	_____	AND	be ignored	
	AUTHOR	_____	_____			
+-----+-----+-----+-----+-----+-----+						
Scroll:						
Name	Type	Library	Environment	Unit		

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

Specify the following information in the fields:

Top Section

Location	The name of the node where the selection criteria are applied. If you do not specify a value, your home location is used.
Library (for RULES)	If the selection list is to contain rules, type the name of the rules library to be searched. Press PF1 for a list of valid values.
Presentation Environment	This field is currently not in use.
Select All	Specifies whether all the items displayed based on the selection criteria should be copied into the Unload screen.
Scroll	You can specify the amount that you want to scroll when you use the scroll keys. Valid entries are: P (page), H (half a page), M (maximum), <i>nn</i> (number of lines). Y – Unloads all the items displayed. N – Do not unload all the items displayed.
List Children	Specify if you want to list all the child objects that an object is composed of. Valid values are: Y – Lists all the child objects. N – Do not list the child objects.

Middle Section

The middle section of the screen can be used to select the items to be unloaded, or to narrow down the selection list. You can use more than one type of selection criteria for each object type and you can specify multiple object types within one session. For a list of valid values for each of these fields, press PF1. For more information, refer to “Selection Criteria” below.

Bottom Section

When you press Enter after specifying the selection criteria, the selected items appear in the bottom portion of the screen. You can select the objects displayed in this section by typing an **S** in the line command field beside the objects. For more information, refer to the following “Selection Criteria”.

Selection Criteria

NAME	If you know the name of the item, enter the logical operator to be used in the Op field. Type the name of the object in the Value field.
TYPE	The name of the object type. Press PF1 for a list of valid values. If you do not supply an object type, you must specify a value in at least one of the other selection fields. If you specify only an object type and no further selection values, a listing of the items for the object type defined in your TIBCO Object Service Broker database appears for further selection.
UNIT	In the Op field, type the logical operator to be used. In the Value field, type the name of the unit associated with the object.
AUTHOR	In the Op field, type the logical operator to be used. In the Value field, type the name of the author of the object.

PF Keys

You can use the following PF keys in this screen:

Enter	Updates the screen.
PF3	Saves the selection and returns to the Unload screen.

PF7	Scrolls up.
PF8	Scrolls down.
PF10	Scrolls right.
PF11	Scrolls left.
PF12	Exits without selecting objects and returns to the Unload screen.

You can specify the amount that you want to scroll when you use the scroll keys, using the Scroll field. Valid entries are: P (page), H (half a page), M (maximum), *nn* (number of lines).

Usage Notes

- UNLOAD captures all the children of an object—these do not have to be explicitly specified.
- Definitions of all table types can be unloaded using the UNLOAD tool. However, data from only TDS and SES tables can be unloaded using the UNLOAD tool.
- On non-z/OS systems, set the DSBIFTYPE Execution Environment parameter according to the following:

When unloading...	Use this DSBIFTYPE Setting
From Windows to Windows, or from Solaris to Solaris.	LENGTH_PREFIXED_EBCDIC or LENGTH_PREFIXED_EBCDIC_NATIVE_ENDIAN. Use the same at load time.
Windows to Solaris, or from Solaris to Windows.	LENGTH_PREFIXED_EBCDIC.
To z/OS	LENGTH_PREFIXED_EBCDIC.

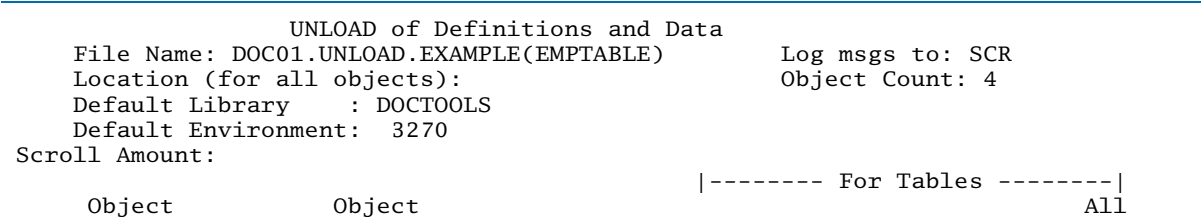
This setting can affect the behavior of other tools. For more information on the DSBIFTYPE Execution Environment parameter, refer to *TIBCO Object Service Broker Parameters*.

- To unload large volumes of data quickly, consider using one of the batch Unload utilities (S6BBRULH /hrnbrulh or S6BBRULB /hrnbrulb). For more information on these utilities refer to *TIBCO Object Service Broker for z/OS Utilities* or *TIBCO Object Service Broker for Open Systems Utilities*.
- This tool should be run in Browse mode (that is, set the **BROWSE** field at the top of the workbench to Y).

- For unloading to a z/OS data set, it is recommended that you pre-allocate the data set as follows: RECORD FORMAT - VB, LRECL >=2250 (2250 is the minimum for unloading rules and that 250 is the minimum for table definitions). It can be a partitioned data set. To optimize LRECL when unloading data, use [EXPOCC_SIZE](#). Depending on your environment, the data set could have to be authorized for you to use it.
- For unloading to a Solaris file only, specify a name that uses all uppercase letters. For Windows or Solaris, specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to *TIBCO Object Service Broker Parameters* for more information about this parameter.
- To unload the table instances of a parameterized table, you must have a parameter value (PRM) table defined.
- If UNLOAD is executed using the EX option from the workbench, the search path used for event rules is local, installation, and then system. If UNLOAD is executed using the LO or UL options from the Administrator’s menu, the search path is the system library since the search path is indicated in @MENU_ITEMS(@ADMIN) as S.
- If you are using FTP to transfer unloaded data from z/OS to Open Systems, you must either issue FTP’s QUOTE SITE RDW command before sending the file from z/OS, or run the S6BBRFRU (Reformat TIBCO Object Service Broker Files Transferred with FTP) utility against the file on z/OS before using FTP. Refer to *TIBCO Object Service Broker Parameters* for more information about S6BBRFRU.
- To load on z/OS a file unloaded on Windows or Solaris, simply FTP the file, in binary format, from the platform of origin to z/OS.
- To UNLOAD a DB2 table definition from the current release in Release 5.0.0 format, set Library to S6B50DB2 and execute the UNLOAD tool. To verify that your definition is compatible with Release 5.0.0, set Library to S6B50DB2 and execute the rule CHK_COMPATDB2TBL(tablename).

Example Sample UNLOAD of Definitions and Data Screen

The figure below shows a sample screen for unloading the definitions of several tables:



Name	Type	Library	Type	#Prms	Defn?	Data?	Inst
EMPLOYEES_ODPARM	TABLE				---	-	-
EMPLOYEES_1DPARM	TABLE				TDS	0	Y
EMPLOYEES_2DPARM	TABLE				TDS	1	Y
EMPLOYEES_3DPARM	TABLE				TDS	2	Y
					TDS	3	Y

PFKEYS: 5=SELECT OBJECTS 6=INSTANCE SELECT 3=UNLOAD 12=CANCEL

Sample Object Selection Screen

The figure below shows a sample screen used to select objects:

Object Selection

COMMAND ==>

Location:

Library (for RULES): DOCTOOLS

Presentation Environment: 3270

Select All: N

List Children: N

Selection Specification

Attr	Op	Value		
NAME	LIKE	EMPLO*_____	AND	unspecified
TYPE	=	_____	AND	attributes will
UNIT	_____	_____	AND	be ignored
AUTHOR	_____	_____		

Scroll: P

Name	Type	Library	Environment	Unit
EMPLOYEEIDM	TABLE			USR40
EMPLOYEES	TABLE			ACC
s EMPLOYEES_ODPARM	TABLE			ACC
s EMPLOYEES_1DPARM	TABLE			ACC
s EMPLOYEES_2DPARM	TABLE			ACC
s EMPLOYEES_3DPARM	TABLE			ACC

PFKEYS: ENTER=UPDATE 3=SAVE SELECTION 12=CANCEL

Sample Message Log

The figure below shows the message log summarizing the results of the unload.

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
                                           H U R O N  MESSAGES
Date: March 14, 2000
                                           Report on UNLOAD Status of Objc

Tally of objects UNLOADED

OBJECT          COUNT
TABLE DEFN      4

-----

Unloading definitions of TABLEs

NAME              NAME              NAME              NAME
EMPLOYEES_0DPARM  EMPLOYEES_1DPARM  EMPLOYEES_2DPARM  EMPLOYEES_

Above message occurred 4 times

PFKEYS: 2=NEXT LOG 3=EXIT 5=REPEAT 12=EXIT 13=PRINT
```

UNLOAD_DATA

Unloads the data of a table to a z/OS data set or a Windows or Solaris file. (C)

Invocation `CALL UNLOAD_DATA(tablespec, selection, location)`

<i>tablespec</i>	The name of an existing table or table instance.
<i>selection</i>	The selection criteria to be used, if required.
<i>location</i>	The name of the node where the data is located.

Prerequisites You must call [@OPENDSN](#) before you call UNLOAD_DATA. After calling @OPENDSN, you can make multiple calls to UNLOAD_DATA.

- Usage Notes**
- [UNLOAD](#) is the interactive version of this tool.
 - On non-Z/OS systems, the Execution Environment parameter DSBIFFTYPE must be set to LENGTH_PREFIXED_EBCDIC.
 - Only TDS and session (SES) table data can be loaded using the UNLOAD_DATA tool.
 - You must declare the local variable UNLOAD_MSG.
 - Parameter values must be specified in the *tablespec* argument rather than the *selection* argument. The *tablespec* must specify a value for each parameter of a parameterized table.
 - You can call [UNLOAD_DEFN](#) within the same rule.
 - If you specify an empty string for *select*, all occurrences are unloaded from the table, provided the table is non-parameterized or only a single instance is specified.
 - The syntax for *select* is `<field name> <relational operator> <value>`.
 - Specify a value for *location* only if the data is located on a different node.
 - If UNLOAD_DATA is executed using the EX option from the workbench, the search path used for event rules is local, the installation library, and then COMMON. If UNLOAD_DATA is executed using the LO or UL options from the @ADMIN menu, the search path is COMMON since the search path is indicated in @MENU_ITEMS(@ADMIN) as S.
 - If you are using FTP to transfer unloaded data between z/OS and Windows, Solaris, or UNIX, it is no longer mandatory to use the S6BBRFRU z/OS utility

before the data can be used by TIBCO Object Service Broker. Refer to *TIBCO Object Service Broker for z/OS Utilities* for more information about S6BBRFRU.

Exceptions

UNLOAD_FAILED	Raised if invalid values are specified for <i>tablespec</i> , <i>selection</i> , or <i>location</i> . It is also raised if you do not have security access to the data or definition of the table or table instance.
---------------	--

Examples

The following rule unloads data from two tables. Data is unloaded from an instance of the EMPLOYEES table and from a different node than your default node location for the MANAGERS table.

RULE EDITOR ==>	SCROLL: P
UNLOAD_DATA_1;	
_ LOCAL UNLOAD_MSG;	
-----+-----	
-----+-----	
_ CALL @OPENDSN('USR40.UNLOAD.HURON(DATA)');	1
_ CALL UNLOAD_DATA('EMPLOYEES(CANADA)', '', '');	2
_ CALL UNLOAD_DATA('MANAGERS', '', 'NODE3');	3
-----+-----	

The following rule unloads the data from all instances of the table EMPLOYEES by using a FORALL loop on the \$EMPLOYEES parameter value table:

RULE EDITOR ==>	SCROLL: P
UNLOAD_EMPS;	
-----+-----	
-----+-----	
_ CALL @OPENDSN('EMPS.DATA.OUT');	1
_ FORALL \$EMPLOYEES :	2
_ CALL UNLOAD_DATA('EMPLOYEES(' \$EMPLOYEES.REGION	
_ ')', '', '');	
_ END;	
_ CALL @CLOSEDSN;	3
-----+-----	

UNLOAD_DEFN

Unloads the definition of a TIBCO Object Service Broker object to a z/OS data set or a Windows or Solaris file. (C)

Invocation `CALL UNLOAD_DEFN(objecttype, objectname, library, presentationenv, location, parentonly)`

<i>objecttype</i>	<p>The TIBCO Object Service Broker object type of the object definition to be unloaded. Valid object types are:</p> <ul style="list-style-type: none"> • GLOBALFIELD • LIBRARY • MENU • OBJECTSET • REPORT • RULE • SCREEN • TABLE • WEBSERVICEPROD
<i>objectname</i>	The name of the object definition to be unloaded.
<i>library</i>	If the object is a rule, the name of the rules library where the rule is stored.
<i>presentationenv</i>	This argument, although not currently used, must be supplied. You can enter a null (") value.
<i>location</i>	The name of the node where the object is located.
<i>parentonly</i>	<p>Specifies if all the objects or only the parent object should be unloaded. Valid values are:</p> <p>Y – Unloads only the parent.</p> <p>N – Unloads the parent and child objects.</p>

Prerequisites You must call [@OPENDSN](#) before you call UNLOAD_DEFN. After calling @OPENDSN, you can make multiple calls to UNLOAD_DEFN provided you are writing to the same data set or file to the same member of a partitioned data set. Unloads to different data sets or to different members of a partitioned data set must be done in separate transactions.

- Usage Notes**
- [UNLOAD](#) is the interactive version of this tool.
 - On non-Z/OS systems, the Execution Environment parameter DSBIFTYPE must be set to LENGTH_PREFIXED_EBCDIC.
 - You can call [UNLOAD_DATA](#) within the same rule.
 - Specify a value for *location* only if the definition is located on a different node.
 - If you are using FTP to transfer unloaded data between z/OS and Windows or Solaris, it is no longer mandatory to use the S6BBRFRU z/OS utility before the data can be used by TIBCO Object Service Broker. Refer to *TIBCO Object Service Broker for z/OS Utilities* for more information about S6BBRFRU.

Example The following rule unloads a report definition from NODE3 and a table definition from your default node location:

```
RULE EDITOR ==>                                SCROLL: P
UNLOAD_DEFN_1;

- -----+-----
- -----+-----
- CALL @OPENDSN('USR40.UNLOAD.HURON(TEST2)');      | 1
- CALL UNLOAD_DEFN('REPORT', 'EMP_EXPENSE', '',    | 2
- '', 'NODE3', 'N');                               |
- CALL UNLOAD_DEFN('TABLE', 'EMPLOYEES', '', '',   | 3
- '', 'N');                                         |
- -----+-----
```

UNLOADLIBRARY

Unloads all rules in the specified library at the specified location to a z/OS data set or a Windows or Solaris file. (C)

Invocation `CALL UNLOADLIBRARY(library, location)`

<i>library</i>	The name of the library from which to unload the rules.
<i>location</i>	The name of the node where the library resides.

Usage Notes • You must perform the following steps to use UNLOADLIBRARY:

If the output is going to...	Do this
A z/OS data set	Allocate a data set with these characteristics: RECORD FORMAT - VB, LRECL - 2250 or larger. It can be a partitioned data set. You must call @OPENDSN to set up the destination data set.
Windows	Use @OPENDSN to set up the destination file. Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to <i>TIBCO Object Service Broker Parameters</i> for more information about this parameter.
A Solaris file	Specify a filename in uppercase. Use @OPENDSN to set up the destination file. Specify either the full path or only the filename. If you specify only the filename, the DSDIR Execution Environment parameter must be set to point to the directory to use. Refer to <i>TIBCO Object Service Broker Parameters</i> for more information about this parameter.

- Use [\\$SETPRINT/\\$RESETPRINT](#) to set up a destination for status messages.
- On non-Z/OS systems, the Execution Environment parameter DSBIFFTYPE must be set to LENGTH_PREFIXED_EBCDIC.
- This tool is recommended as the preferred tool for unloading libraries.
- If you are using FTP to transfer unloaded data between z/OS and Windows or Solaris, it is no longer mandatory to use the S6BBRFRU z/OS utility before the

data can be used by TIBCO Object Service Broker. Refer to *TIBCO Object Service Broker for z/OS Utilities* for more information about S6BBRFRU.

Example The following rule unloads all the files from the specified library and writes an end message at completion:

```
UNLOADLIBRARY_1(FILE);
```

```

-
- -----
-                                     +-----
- CALL @OPENDSN(FILE);                               | 1
- CALL $SETPRINT(10, 45, 1, 'SCR', 'N');              | 2
- CALL UNLOADLIBRARY('USR40', '');                   | 3
- CALL ENDMSG('THE OUTPUT HAS BEEN DIRECTED TO ' xx 'SCR'); | 4
- -----

```

Resulting Output

After running this rule, the following message appears in the message log:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P

----- NEW PAGE -----

Unloaded 14 rules from library USR40

```

Page 1

\$UNPIC

Determines the original value submitted given a masked value produced by `$PIC` and the display mask that produced it. (F)

Invocation `value = $UNPIC(picval, mask)`

<i>value</i>	The original value submitted to <code>\$PIC</code> . Its syntax is V (variable-length character string).
--------------	--

<i>picval</i>	The value produced by <code>\$PIC</code> . Its syntax is V.
---------------	---

<i>mask</i>	The display mask used by <code>\$PIC</code> . Its syntax is V.
-------------	--

Usage Notes `$UNPIC` reverses the function of `$PIC`:

`picval=$PIC(value, mask)`

`value=$UNPIC(picval, mask)`

For information on mask elements and terminology, refer to `$PIC`.

picval

You can omit the following in *picval*:

- Mask elements
- Leading zeros or fill characters (asterisks or blanks)
- Trailing decimal zeros
- Message characters in the basic string, provided they are all omitted

picval can have leading zeros with message characters between them; it cannot have other fill characters (asterisks or blanks) with message characters between them.

Sign of a Value

The sign of a value is determined by all or one of the following:

- The sign holder in the unconditional left string of *picval*
- A user input positive (+) or negative (-) sign preceding the basic string in *picval* (allowed only if there is a sign holder in the mask)
- The presence of a conditional right string in *picval*

Any number of fill characters can be entered in the conditional right string of *picval*, provided they are not ambiguous (for example, when a user-supplied negative (-) sign conflicts with the positive value implied by the presence of fill characters). \$UNPIC fails if it cannot with certainty determine the sign associated with a value. A similar failure occurs if the presence of a conditional right string conflicts with a user-supplied positive (+) sign.

Effect of Display Masks

If there is no basic string in the mask or if the basic string is in the mask and not in *picval*, \$UNPIC returns a null value.

If the *picval* is inconsistent with the mask because of an overflow either before or after the decimal point, \$UNPIC fails. \$UNPIC truncates only if the mask has a decimal separator and the digits being truncated are all decimal zeros.

- `$UNPIC('12345', '999V.99')` fails
- `$UNPIC('12.345', '999V.99')` fails
- `$UNPIC('123.45000', '999V.99')` returns '123.45'

\$PIC/\$UNPIC Inconsistencies

In some cases, the display mask causes values to be truncated or information to be lost when \$PIC is used. In cases such as the following, inconsistencies occur:

- `$PIC(-123, '999')` returns '123' (information is lost)
- `$UNPIC('123', '999')` returns '123' instead of '-123'
- `$PIC(0.345, '9V.99')` returns '0.34' (truncation occurs)
- `$UNPIC('.34', '9V.99')` returns '0.34' instead of '0.345'
- `$PIC(0, 'CDN$ZZ')` or `$PIC('', 'CDN$ZZ')` returns CDN\$ (0 and null produce the same mask)
- `$UNPIC('CDN$', 'CDN$ZZ')` returns a null

Examples

Below are some examples of \$PIC values and the \$UNPIC values that result:

- `$PIC (0.01, 'NV999') = '010'`
- `$UNPIC('010', 'NV999') = '0.01'`
- `$PIC(23, ',***,999') = ',****023'`
- `$UNPIC(',****023', ',***,999') = '23'`

UNQUOTE

Returns a string with the single quotation marks removed. (F)

Invocation `unquoted = UNQUOTE(string)`

<i>unquoted</i>	<i>string</i> without single quotation marks at the beginning and end. Pairs of single quotation marks within <i>string</i> are changed to single quotation marks.
<i>string</i>	The character string from which to remove the single quotation marks.

- Usage Notes**
- Only the first and last single quotation mark are removed. If you have a pair of single quotation marks at the beginning and a pair at the end, the returned string has one quotation at the beginning and one at the end.
 - If you have a single quotation mark in the string and it is not at the beginning or end, it is not removed.
 - If you have only a single quotation mark at the end of the string, it is not removed.
 - If you have only a single quotation mark at the beginning of the string, it and the last character are removed.

Example

The following rule removes quotation marks from an entry that a user made on a screen and passes the unquoted string to be processed. The screen is called EMPINFO_SCR and the screen table into which the information is entered is called EMPINFO_SCRTAB.

```
CLEAN_INPUT;
-
- -----
- GET EMPINFO_SCRTAB(EMPINFO_SCR);                               | 1
- CALL INPUT_EMPINFO(UNQUOTE(EMPINFO_SCRTAB.EMPNAME));           | 2
- -----
```

If you enter:
'AB' 'CD EF'
the string that is passed to the INPUT_EMPINFO rule is:
AB'CD EF

UPPER_EBCDIC

Converts a string to uppercase EBCDIC characters. (F)

Invocation `upper_string = UPPER_EBCDIC(string)`

<code>upper_string</code>	<p>On return, contains the string in uppercase letters.</p> <p>If <i>string</i> is not Unicode, or if it is Unicode and entirely convertible to EBCDIC, <i>upper_string</i>'s syntax is V (variable-length character string).</p> <p>If <i>string</i> is Unicode and not entirely convertible to EBCDIC, <i>upper_string</i>'s syntax is UN (Unicode). In <i>string</i>, characters that can be converted are cased and then reverted to Unicode before being added to <i>upper_string</i>. Characters that cannot be converted are added to <i>upper_string</i> unchanged.</p>
<code>string</code>	<p>The string to convert to uppercase letters. Its syntax is C (fixed-length character), RD (raw data), UN, V, or W (double-byte character).</p>

Usage Note UPPER_EBCDIC uppercases strings using the TIBCO Object Service Broker EBCDIC casing rules.

The casing for EBCDIC depends on the locale and is explained in *TIBCO Object Service Broker National Language Support*.

For example, “ç” remains unchanged when the EBCDIC rules for Swedish are in use.

Example The following rule uppercases a string and prints the result to the message log:

```
UPPERCASE_SAMPLE;
_ LOCAL A;
_ -----
_ -----+-----
_ A = U'AÇaç';                                | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A); | 2
_ CALL MSGLOG(' ');                            | 3
_ CALL MSGLOG('UPPER_EBCDIC GIVES ' || UPPER_EBCDIC(A)); | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPER_EBCDIC(A))); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A);           | 6
_ CALL MSGLOG(' ');                            | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A); | 8
_ CALL MSGLOG(' ');                            | 9
_ CALL MSGLOG('UPPER_EBCDIC GIVES ' || UPPER_EBCDIC(A)); | A
```

```
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPER_EBCDIC(A)); | B
```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P

CASING OF UNICODE STRING AÇaç
UPPER_EBCDIC GIVES AÇAÇ
SYNTAX IS V

CASING OF EBCDIC STRING AÇaç
UPPER_EBCDIC GIVES AÇAÇ
SYNTAX IS V
```

See Also Related tools: [LOWER_EBCDIC](#), [LOWER_UNICODE](#), [LOWERCASE](#), [UPPER_UNICODE](#), and [UPPERCASE](#).

UPPER_UNICODE

Converts a string to uppercase Unicode characters. (F)

Invocation `upper_string = UPPER_UNICODE(string)`

<code>upper_string</code>	On return, contains the string in uppercase letters. Its syntax is UN (Unicode).
<code>string</code>	The string to convert to uppercase letters. Its syntax is C (fixed-length character string), RD (raw data), UN, or V (variable-length character string).

Usage Note UPPER_UNICODE uppercases strings using the TIBCO Object Service Broker Unicode casing rules.

The default Unicode casing rules supplied with the product are the recommended default from the Unicode consortium. You can tailor it for your environment.

For example, “ç” cases to “Ç” in the default Unicode rules.

Example The following rule uppercases a string and prints the result to the message log:

```
UPPERCASE_SAMPLE;
_ LOCAL A;
_ -----
_ -----+-----
_ A = U'AÇaç';                                | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A); | 2
_ CALL MSGLOG(' ');                             | 3
_ CALL MSGLOG('UPPER_UNICODE GIVES ' || UPPER_UNICODE(A)); | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPER_UNICODE(A)); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A);             | 6
_ CALL MSGLOG(' ');                             | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A);   | 8
_ CALL MSGLOG(' ');                             | 9
_ CALL MSGLOG('UPPER_UNICODE GIVES ' || UPPER_UNICODE(A)); | A
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPER_UNICODE(A)); | B
```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ===>                                SCROLL ===> P

CASING OF UNICODE STRING AÇaç
UPPER_UNICODE GIVES AÇAÇ
SYNTAX IS UN

CASING OF EBCDIC STRING AÇaç
UPPER_UNICODE GIVES AÇAÇ
SYNTAX IS UN
```

See Also Related tools: [LOWER_EBCDIC](#), [LOWER_UNICODE](#), [LOWERCASE](#), [UPPER_EBCDIC](#), and [UPPERCASE](#).

UPPERCASE

Converts all lowercase characters in a string to uppercase characters. (F)

Invocation `upper_string = UPPERCASE(string)`

<code>upper_string</code>	On return, contains the string in uppercase letters. Its syntax is the same as <i>string</i> except that, if <i>string</i> is C (fixed-length character string), <i>upper_string</i> becomes V (variable-length character string).
<code>string</code>	The string to convert to uppercase letters. Its syntax can be C, UN (Unicode), V, or W (double-byte character).

Usage Note UPPERCASE uppercases EBCDIC strings using the TIBCO Object Service Broker EBCDIC casing rules and Unicode strings using the TIBCO Object Service Broker Unicode casing rules.

The casing for EBCDIC depends on the locale and is explained in *TIBCO Object Service Broker National Language Support*. The default Unicode casing supplied with the product is the recommended default from the Unicode consortium. You can tailor it for your environment.

For example, “ç” cases to “Ç” in the default Unicode rules and remains unchanged using the EBCDIC rules for Swedish.

Example The following rule uppercases a string and prints the result to the message log:

```
UPPERCASE_SAMPLE;
_ LOCAL A;
_ -----
_ -----+-----
_ A = U'AÇaç';                                     | 1
_ CALL MSGLOG('CASING OF UNICODE STRING ' || A);   | 2
_ CALL MSGLOG(' ');                                 | 3
_ CALL MSGLOG('UPPERCASE GIVES ' || UPPERCASE(A));  | 4
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPERCASE(A))); | 5
_ A = $TYPECAST('S', 'V', 4, 0, A);                 | 6
_ CALL MSGLOG(' ');                                 | 7
_ CALL MSGLOG('CASING OF EBCDIC STRING ' || A);     | 8
_ CALL MSGLOG(' ');                                 | 9
_ CALL MSGLOG('UPPERCASE GIVES ' || UPPERCASE(A));  | A
_ CALL MSGLOG('SYNTAX IS ' || $GET_SYNTAX(UPPERCASE(A))); | B
```

Line 1 sets local variable A as a Unicode string. Line 6 changes it to an EBCDIC V string.

Resulting Output

Pressing PF2 after executing this rule displays the following screen:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ==>                                SCROLL ==> P  
  
CASING OF UNICODE STRING AÇaç  
  
UPPERCASE GIVES AÇAÇ  
SYNTAX IS UN  
  
CASING OF EBCDIC STRING AÇaç  
  
UPPERCASE GIVES AÇAÇ  
SYNTAX IS V
```

See Also Related tools: [LOWER_EBCDIC](#), [LOWER_UNICODE](#), [LOWERCASE](#),
 [UPPER_EBCDIC](#), and [UPPER_UNICODE](#).

USERID

Returns a string containing the user ID. (F)

Invocation string = USERID

string On return, contains the user ID. Its syntax is V (variable-length character string) with length 16.

Example The following rule determines the user ID and prints it to the message log:

```
USERID_1;
_ LOCAL ID_STRING;
_ -----
_ -----+-----
_ ID_STRING = USERID;                               | 1
_ CALL MSGLOG('THE USER ID IS: ' || ID_STRING);      | 2
_ -----
```

Resulting Output

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THE USER ID IS: USR40
```

UTCDATE

Returns the Coordinated Universal Time (UTC) date when UTCDATE is called.
(F)

Invocation `date = UTCDATE`

`date` On return, contains the UTC date.

- Usage Notes**
- The date is returned as a semantic type date.
 - The result is displayed in the default date format for the installation.

Example The following rule, which uses PAYMENT as an argument, records the amount received and the current UTC date in the table RECEIPTS:

```

      RULE EDITOR ==>>                                     SCROLL: P
RECEIVED(PAYMENT);
-
- -----
- RECEIPTS.AMOUNT = PAYMENT;                                | 1
- RECEIPTS.DATE = UTCDATE;                                   | 2
- INSERT RECEIPTS;                                           | 3
- -----

```

Resulting Output

When the rule is executed with the argument 67.89, the bottom occurrence is inserted in the table:

```

EDITING TABLE      : RECEIPTS
COMMAND ==>>
-
- NUM      DATE      AMOUNT
- -----
-      1 2005-02-23      34.56
-      2 2005-03-10      78.56
-      3 2005-03-18      23.00
-      4 2005-03-18      67.89
-

```

UTCTIME

Returns a string containing the current Coordinated Universal Time (UTC) time.
(F)

Invocation string = UTCTIME

string	The string containing the current UTC time. Its syntax is C (fixed-length character string) with length 8.
--------	--

- Usage Notes**
- The time is returned in the format HH:MM:SS; for example, 23:59:59.
 - The current UTC time is returned, not the UTC time the transaction started.

Example The following rule determines the current time and prints it to the message log:

```
UTCTIME_1;  
_ LOCAL TIME;  
_ -----  
_ -----+-----  
_ TIME = UTCTIME;                               | 1  
_ CALL MSGLOG('THE CURRENT UTC TIME IS: ' || TIME); | 2  
_ -----
```

Pressing PF2 after executing this rule displays the following:

```
----- INFORMATIONAL MESSAGE LOG -----  
COMMAND ==>                               SCROLL ==> P  
THE CURRENT UTC TIME IS: 07:42:20
```

VAL_TO_LIT

Converts a value to a string containing a token describing its value. (F)

Invocation `string = VAL_TO_LIT(value)`

<i>string</i>	On return, this string contains a token describing the input value. Its syntax is V (variable-length character string).
---------------	---

<i>value</i>	The value to convert. It can have any syntax. The token in <i>string</i> depends on the syntax of <i>value</i> as follows:
--------------	--

Value Syntax	String Token
RD (raw data)	R'xx...'
UN (Unicode)	U'xx...'
Other	X'xx...'

Example This rule creates three new values from the information contained in input strings:

```
CREATE_RD (VALUE1, VALUE2, VALUE3);
```

```

-
- -----+-----
- CALL MSGLOG('VALUE1 CONTAINS ' || VAL_TO_LIT(VALUE1));
- CALL MSGLOG('VALUE2 CONTAINS ' || VAL_TO_LIT(VALUE2));
- CALL MSGLOG('VALUE3 CONTAINS ' || VAL_TO_LIT(VALUE3));
- -----

```

VALUE1 is a raw data field that contains 1234.

VALUE2 is a Unicode field that contains E12/34.

VALUE3 is a string field that contains \$1234.

Pressing PF2 after executing this rule displays the following screen:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>
VALUE1 CONTAINS R'1234'
VALUE2 CONTAINS U'/20AC12//34'
SCROLL ==> P

```

VALUE3 CONTAINS X'5BF1F2F3F4'

VALID_NAME

Determines if a given string satisfies the TIBCO Object Service Broker definition of an identifier. (F)

Invocation `verify = VALID_NAME(name)`

<code>verify</code>	On return, contains the value Y if <i>name</i> is a valid TIBCO Object Service Broker identifier or N if <i>name</i> is not valid.
---------------------	--

<code>name</code>	A string specifying the identifier.
-------------------	-------------------------------------

Usage Notes

- A valid TIBCO Object Service Broker name is a character string of up to 16 characters beginning with a letter (A - Z) or a special character (\$ or #), and continuing with more letters, special characters, digits (0 - 9), or underscore characters (_).

A table name starting with an @ symbol denotes a table supplied with TIBCO Object Service Broker.

- If the string passed to VALID_NAME contains unmatched quotation marks, the value of N is returned.

Example

The following rule verifies that the name you provide for a table instance is a valid TIBCO Object Service Broker name:

RULE EDITOR ==>		SCROLL: P
VERIFY_PARM(PARMNAME);		

VALID_NAME(PARMNAME) = 'Y';		Y N

CALL INPUT_DATA(PARMNAME);		1
CALL ENDMSG(PARMNAME ' IS NOT A VALID PARAMETER NAME');		1

If the parameter name is valid, INPUT_DATA is called to insert data into the table. If the parameter name is not valid, such as 4-5ITEMS, the user sees the message:

4-5ITEMS IS NOT A VALID PARAMETER NAME.

@WRITEDSN

Writes a record to the current file. (C)

Invocation `CALL @WRITEDSN(string)`

<i>string</i>	The character string that is written as the next record. Its syntax can be C (fixed-length character string), V (variable-length character string), or W (double-byte character).
---------------	---

- Usage Notes**
- The file must be previously identified using [@OPENDSN](#).
 - An attempt to open the file is made with the first write operation.
 - You must have write access to the file.
 - @WRITEDSN accesses a z/OS file using the data set name. There is no provision for using a DDNAME with this tool instead of a data set name.
 - If the file specified in the @OPENDSN statement is a z/OS data set and it does not exist, @WRITEDSN fails. If the file is a Windows or Solaris file, it is created for you.
 - If a partitioned data set is specified in the @OPENDSN statement a member name must be supplied.
 - Within the same transaction, to write to a file that you have read, close it and then re-open it.
 - On the z/OS platform, @WRITEDSN always writes in EBCDIC format.
 - On non-z/OS platforms, when the data is written to the external file, it is subject to the type specification for the file as given in filespec.dsn or by the DSBIFFTYPE Execution Environment parameter. If the file type is LENGTH_PREFIXED_EBCDIC, the data is left alone and written as EBCDIC. If the file type is LINE_SEPARATED_ASCII, the data is converted from EBCDIC to ASCII when written and back from ASCII to EBCDIC when read (using [@READDSN](#)).

Exception

ROUTINEFAIL	Raised if you are attempting to output to a file that has not yet been specified by @OPENDSN, if the file cannot be opened, or if the length of the source record is bigger than the record length of the output file.
--------------------	--

Example The following rule opens an existing file, writes data from the example table to it, closes the file, reopens it, reads back the first record from it, and prints that record to the message log:

RULE EDITOR ==>		SCROLL: P
WRITEDSN_1;		
_ LOCAL RECORD;		
_ -----		
_ -----		
_ CALL @OPENDSN(TSOID ' .EXAMPLES.DATA');	1	
_ FORALL EMPLOYEE;	2	
_ CALL @WRITEDSN(EMPLOYEE.LNAME);		
_ END;		
_ CALL @CLOSEDSN;	3	
_ CALL @OPENDSN(TSOID ' .EXAMPLES.DATA');	4	
_ RECORD = @READDSN;	5	
_ CALL MSGLOG(RECORD);	6	
_ CALL @CLOSEDSN;	7	
_ -----		

Resulting Output

Pressing PF2 after executing this rule displays the following output:

----- INFORMATIONAL MESSAGE LOG -----	
COMMAND ==>	SCROLL ==> P
SMYTHE	

XMLPARSE

Initiates the parsing of an XML document. (C)

Invocation CALL XMLPARSE(*docname*, *validate*, *docsource*, *docdata*).

<i>docname</i>	The name of the event map document that processes the XML document.
<i>validate</i>	Specifies whether to validate the document. The valid values are as follows: <ul style="list-style-type: none"> Y – Validate the document against a DTD. N – Do not validate; check for wellformedness only.
<i>docsource</i>	Specifies the source of the document. The valid values are as follows: <ul style="list-style-type: none"> STORAGE – The document resides in memory. URL – The document is retrieved by means of a URL. FILE – The document is retrieved from a local file. STRING – The document resides in a string.
<i>docdata</i>	Specifies the location of the document. The valid values are as follows: <ul style="list-style-type: none"> STORAGE – The pointer to the memory address holding the document. URL – The URL from which to retrieve the document. FILE – The name of the file containing the document. STRING – The string holding the document.

- Usage Notes**
- Depending on the first characters of the file, the XML parser assumes that an XML document is stored in ASCII or UNICODE.
 - The encoding of an XML document is defined by the XML document header. The encoding parameter in the header is always honored. If one is not provided, then the parser probes the header of the document to detect whether the document is ASCII or UNICODE. The parser supports standard ISO encoding names. On z/OS we also support IBM extensions to the

encoding name pool, such as `ibm037-s390`, which represents the US EBCDIC code page. For example:

```
<?xml version='1.0' encoding='ibm037-s390'?>
```

When passing XML documents as string from a rule, the string is automatically converted to UTF-16. As a result, specifying an encoding is optional. However, should you chose to specify an encoding in the XML document header then you must specify UTF-16.

- The parser on z/OS can access XML documents from normal z/OS data sets and from UNIX System Services (USS) files. You must use a USS-like file name when referring to the file. For example, if the data-set name is `MYUSERID.TEST.XML`, name the file `MYUSERID/TEST/XML`.
- XML in mapped data areas must be terminated with a null character.
- If the document is stored in a single byte encoding or UTF-8, the document must end with the characters `0x00`.
- If the document is stored as UTF-16, the document must end with the characters `0x0000`.
- XML documents held in strings must be smaller than 65,536 bytes in size when converted to syntax UN.

Exceptions

ECTSERROR	An error occurred during the parsing of the XML document. For more information, inspect the return value of GETENDMSG.
-----------	--

Example

The following example shows XML in a mapped data area being parsed with the `STORAGE` operand.

```
CALL XMLPARSE('MYDOC', 'N', 'STORAGE', MAPAREAPointer);
```


XMLSTART

Generates an XML document based on the passed data access arguments. (C)

Invocation CALL XMLSTART(*xmldocname*, *predicate*, *parm*)

<i>xmldocname</i>	The name of the XML document to be generated.
<i>predicate</i>	The selection criteria for the table.
<i>parm</i>	The names of the table parameters, if any, for the data to be returned.

- Usage Notes**
- In addition to its parameters, XMLSTART requires that you declare and initialize several variables within the scope of the rule to be invoked.
 - XMLSTART can output the generated XML document to one of the following:
 - Multiple rows of a particular table, inserting portions of the XML document into a particular field of the table
 - A data set or file
 - A region in memory
 - The values to assign the required variables depend on the destination of the generated XML document. See the examples for details.

Exceptions

ECTSErrorR	An error occurred during the generation of the XML document.
-------------------	--

Examples Example 1: Populate rows of a table with portions of an XML document.
Here is the table definition in this example:

COMMAND==>										TABLE DEFINITION									
Table: XMLDATA					Type: TEM			Unit: USER40			IDgen: Y								
Parameter Name	Typ	Syn	Len	Dec	Class									Event	Rule	Typ	Acc		
LOCATION	I	C	16	0	L														
Field Name	Typ	Syn	Len	Dec	Key	Ord	Rqd		Default								Reference		

_ KEY	I	B	4	0	P
_ TEXT	S	V	60	0	
_					

The following generic rule populates the table:

```
RULE EDITOR ==>                                SCROLL: P
XML2TABLE(TABLESPEC, FIELDSPEC, XMLDOCNAME, PREDICATE, PARM);
_ LOCAL DOCCOFFSET, ECTSMODE, WRITETOMEM, WRITETOEXP;
_ -----
_ -----+-----
_ CALL XMLSTARTSETDEST(TABLESPEC, FIELDSPEC);      | 1
_ WRITETOMEM = 'N';                                | 2
_ WRITETOEXP = 'N';                                | 3
_ CALL ECTSNLSINIT;                                | 4
_ CALL XMLSTART(XMLDOCNAME, PREDICATE, PARM);      | 5
_ -----
```

The following invokes the generic rule:

```
RULE EDITOR ==>                                SCROLL: P
T_XML2TABLE;
_ -----
_ -----+-----
_ CALL XML2TABLE('XMLDATA', 'TEXT', 'BOOKS',      | 1
_ 'GENRE='Science Fiction'', '');                  |
_ FORALL XMLDATA :                                | 2
_   CALL MSGLOG(XMLDATA.TEXT);                      |
_   END;                                             |
_ -----
```

The result from the session log is as follows:

```
<?xml version="1.0" ?>
<!-- XMLDocName=BOOKS -->
<!-- Generated By ObjectStar Integration Gateway V2.5 - Code
Level 1.00 -->
<bookstore>
<book KEY="3">
<genre>Science Fiction</genre>
<bktitle>Stranger In A Strange Land</bktitle>
<author_ln>Heinlein</author_ln>
<author_fn>Robert A</author_fn>
<price>7.99</price>
</book>
<book KEY="4">
<genre>Science Fiction</genre>
<bktitle>I, Robot</bktitle>
<author_ln>Asimov</author_ln>
<author_fn>Isaac</author_fn>
```

```
<price>5.99</price>
</book>
</bookstore>
```

Example 2: Output an XML document to a file.

Here is a generic rule for outputting an XML document to a data set or file:

```
RULE EDITOR ==>                                SCROLL: P
XML2DATASET(DSN, XMLDOCNAME, PREDICATE, PARM);
_ LOCAL DOCOFFSET, ECTSMODE, WRITETOMEM, WRITETOEXP;
_ -----
_ -----+-----
_ WRITETOMEM = 'N';                               | 1
_ WRITETOEXP = 'Y';                               | 2
_ CALL ECTSNLSINIT;                               | 3
_ CALL @OPENDSN(DSN);                             | 4
_ CALL XMLSTART(XMLDOCNAME, PREDICATE, PARM);      | 5
_ CALL @CLOSEDSN;                                 | 6
_ -----
```

The following invokes the generic rule for generating the XML document in ASCII on an Open Systems platform:

```
RULE EDITOR ==>                                SCROLL: P
T_XML2DATASET;
_ LOCAL OLD_DSBIFTYPE;
_ -----
_ -----+-----
_ OLD_DSBIFTYPE = $GETOPT('DSBIFTYPE');             | 1
_ CALL $SETOPT('DSBIFTYPE', 'LINE_SEPARATED_ASCII'); | 2
_ CALL XML2DATASET('XML.BOOKS.OUTPUT', 'BOOKS',    | 3
_   'GENRE='Science Fiction'', '');               |
_ CALL $SETOPT('DSBIFTYPE', OLD_DSBIFTYPE);        | 4
_ -----
```

The contents of the resulting file XML.BOOKS.OUTPUT is the same as the contents of the session log in Example 1.

Example 3: Output an XML document to a region of memory.

Here is a generic rule for generating an XML document into memory:

```
RULE EDITOR ==>                                SCROLL: P
XML2MEMORY(ADDRESS, LENGTH, XMLDOCNAME, PREDICATE, PARM);
_ LOCAL DOCOFFSET, ECTSMODE, WRITETOMEM, WRITETOEXP, CURRENTPOINTER,
_   STORAGEEND, ENCODING, ITEMCOUNT;
_ -----
_ -----+-----
_ WRITETOMEM = 'Y';                               | 1
_ WRITETOEXP = 'N';                               | 2
```

```

_ CALL ECTSNLSINIT; | 3
_ CURRENTPOINTER = ADDRESS; | 4
_ STORAGEEND = ADDRESS + LENGTH; | 5
_ ENCODING = 'ASCII'; | 6
_ ITEMCOUNT = 0; | 7
_ ECTS_TOC.ITEMCOUNT = 0; | 8
_ CALL XMLSTART(XMLDOCNAME, PREDICATE, PARM); | 9
_ RETURN(ITEMCOUNT); | A
_ -----
```

The following shows the generic rule and retrieves and displays the generated XML document through a MAP table:

```

RULE EDITOR ==>                                SCROLL: P
T_XML2MEMORY;
_ LOCAL ITEMS, ADDRESS;
_ -----
_ @MAP.SIZE = 4096; | 1
_ INSERT @MAP('TRANSACTION'); | 2
_ ADDRESS = @MAP.ADDRESS; | 3
_ ITEMS = XML2MEMORY(ADDRESS, @MAP.SIZE, 'BOOKS', | 4
_ 'GENRE='Science Fiction'', '');
_ UNTIL EQ : | 5
_ CALL @EQ(ITEMS, 0);
_ GET XMLMEM(ADDRESS) WHERE KEY = 1;
_ CALL MSGLOG(HEADSTRING(XMLMEM.TEXT, XMLMEM.LENGTH));
_ ADDRESS = ADDRESS + XMLMEM.LENGTH + 2;
_ ITEMS = ITEMS - 1;
_ END;
_ -----
```

Here is the definition of the MAP table:

TABLE DEFINITION													
Table: XMLMEM				Type: MAP				Unit: USER40				IDgen: Y	
Parameter	Name	Typ	Syn	Len	Dc	Cls	Reference			Event	Rule	Typ	Acc
ADDRESS		B		4	0	A							
LOCATION		I	C	16	0	L							
----- EXTERNAL ----- ----- Metadata Definition -----													
Field Name		Xsyn	Xlen	Xdec	Offset	Key	Typ	Syn	Len	Dec	Rqd	Default	
KEY		B		4	0	0	P	I	B	4	0		
LENGTH		B		2	0	0			B	2	0		
TEXT		J		80	0	2			V	80	0		

The contents of the session log are the same as in Example 1.

XMLSTARTDSN

Generates an XML document based on the passed data access arguments and places it in the specified file. (C)

Invocation CALL XMLSTARTDSN(*xmldocname*, *predicate*, *parm*)

<i>outdsn</i>	The name of the data set or file for the output.
<i>xmldocname</i>	The name of the XML document to be generated.
<i>predicate</i>	The selection criteria for the table.
<i>parm</i>	The names of the table parameters, if any, for the data to be returned.

Usage Notes If the destination for the XML document is a z/OS data set, it must already exist.

Example The following rule sends the output to the data set XML.BOOK.OUTPUT:

```
RULE EDITOR ==>                                SCROLL: P
T_XMLSTARTDSN;
_ LOCAL FILENAME, XMLDOCNAME, PREDICATE, PARM;
_ -----
_ FILENAME = 'XML.BOOK.OUTPUT';                    | 1
_ XMLDOCNAME = 'BOOKS';                            | 2
_ PREDICATE = 'GENRE='Science Fiction''';          | 3
_ PARM = '';                                         | 4
_ CALL XMLSTARTDSN(FILENAME, XMLDOCNAME, PREDICATE, PARM); | 5
_ -----
```


XMLSTARTSETDEST

Sets up the output table and field for XMLSTART. (C)

Invocation `CALL XMLSTARTSETDEST(tablespec, fieldspec)`

<code><i>tablespec</i></code>	The name of the table that holds the output.
<code><i>fieldspec</i></code>	The name of the field that holds the output.

- Usage Notes**
- Call XMLSTARTSETDEST to prepare for outputting an XML document to a particular field with XMLSTART.
 - XMLSTARTSETDEST identifies the table and its field for XMLSTART.

Example See [XMLSTART](#).

XMLSTARTTAB

Returns the data of a table instance to the OIG client. (C)

Invocation `CALL XMLSTARTTAB(tablename, format, predicate, parm)`

<i>tablename</i>	The table to be returned to the client.
<i>format</i>	The XML schema for the document for the table. The valid values are ROWSET, ADO.NET, and MSSHEMA.
<i>predicate</i>	The selection criteria for the table.
<i>parm</i>	The names of the table parameters, if any, for the data to be returned.

- Usage Notes**
- Call XMLSTARTTAB from a rule invoked while executing an OIG transaction.
 - XMLSTARTTAB returns to the OIG client the data of the specified table instance and that of any output tables defined for the transaction.
 - XMLSTARTTAB sends tables to the OIG client as XML documents but accesses them by client code through tabular interfaces.

Example

```

RULE EDITOR ==>                                SCROLL: P
XMLEXAMPLE;

-
- -----
- CALL XMLSTARTTAB('ECTSGETARG('TABLESPEC'), ECTSRETURNARG(      | 1
-   'SCHEMATYPE'), ECTSRETURNARG('SELECTSTRING'),                |
-   ECTSRETURNARG('PARMSTRING')));                                |
- -----

```

YEAR

Returns the two-digit year when the transaction started based on the local machine 's time zone in which the Execution Environment is running. (F)

Invocation time = YEAR

time	On return, contains the year. Its syntax is C (fixed-length character string) with length 2.
------	--

Usage Notes The returned value is a character string containing the year modulo 100 (98, 99, 00...).

Example The following rule determines the year when the current transaction started and prints it to the message log:

```

      RULE EDITOR  ==>
YEAR_1;
_  LOCAL TIME;
_  -----
_  -----+-----
_  TIME = YEAR;                               | 1
_  CALL MSGLOG('THIS TRANSACTION WAS STARTED IN 20' || TIME); | 2
_  -----
```

Output for the YEAR_1 Rule

Pressing PF2 after executing this rule displays the following in the message log:

```

----- INFORMATIONAL MESSAGE LOG -----
COMMAND ==>                                SCROLL ==> P
THIS TRANSACTION WAS STARTED IN 2000.
```

Index

Symbols

- @ARCH_ACCESSLOGI tool 39
- @CLOSEDSN tool 94
- @CONFIGURESERVER tool 98
- @ENTRY_VALIDATE user exit 27
- @FORALLA tool 231
- @INSTALL tool 297
- @MAKEMEMBERS tool 341
- @MAP tool 346
- @MESSAGEDUMP tool 361
- @MESSAGETRACE tool 363
- @MNG_USERS tool 370
- @MOM tools 379
 - @MOMCLOSE 373
 - @MOMCOMMIT 374
 - @MOMCONNECT 375
 - @MOMDISCONN 377
 - @MOMGET 378
 - @MOMMAPLENGTH 381
 - @MOMOPEN 382
 - @MOMOPTION 383
 - @MOMPUT 384
 - @MOMROLLBACK 386
 - @MOMSETOPT 387
 - @MOMSPECIALCMD 388
 - @MOMVALIDRC 389
- @MOMCLOSE tool 373
- @MOMCOMMIT tool 374
- @MOMCONNECT tool 375
- @MOMDISCONN tool 377
- @MOMGET tool 378
- @MOMINIT 379
- @MOMINIT tool 379
- @MOMMAPLENGTH tool 381
- @MOMOPEN tool 382
- @MOMOPT tool 387
- @MOMOPTION tool 383
- @MOMPUT tool 384
- @MOMROLLBACK tool 386
- @MOMSPECIALCMD tool 388
- @MOMVALIDRC tool 389
- @MQSMAP tool 394
- @MQSMAP_PORT tool 394
- @OPENDSN tool 411
- @PEERSERVERID tool 453
- @PRE_SAVE_OBJECT user exit 27
- @PRESENTATIONENV tool 462
- @READDSN tool 495
- @SAVED_OBJECT user exit 27
- @SERVERERROR tool 577
- @SESSION tool 583
- @SESSIONCOUNTS tool 587
- @STATICSQL tool 652
- @TRACEMESSAGES tool 692
- @UNINSTALL tool 699
- @WRITEDSN tool 733
- \$ADD_DATE tool 35
- \$BATCHOPT tool 69
- \$BEEP tool 76
- \$BLANKPAGE tool 77
- \$BRCONTAINER tool 79
- \$CALLRULE tool 84
- \$CLRTAB tool 95, 637
- \$CREATE_DATE tool 120
- \$DATE_DEFAULT tool 141
- \$DATE_LENGTH tool 143
- \$DATE_PIC tool 146
- \$DATE_REF tool 149
- \$DELCONTAINER tool 175
- \$EXCEPTION tool 212
- \$EXCEPTIONOBJECT tool 213
- \$FLUSHPRINT tool 223
- \$FUNCTION tool 244
- \$GET_DECIMALS tool 258
- \$GETATTRIBUTE tool 263
- \$GETCOLOUR tool 269
- \$GETCONTAINER tool 256

\$GETENVCOMMAREA tool 273
 \$GETFLOAT tool 274
 \$GETOPT tool 276, 276
 \$GETTRANSACTION tool 280
 \$GTFSET tool 281
 \$HTTPREQUEST tool 291, 291
 \$LISTDSN tool 309
 \$LISTPDS tool 314
 \$MOVECONTAINER tool 390
 \$NEWPAGE tool 398
 \$PIC tool 455
 \$PRINTFIELD tool 467
 \$PRINTLINE tool 469
 \$PUTCONTAINER tool 487
 \$PUTLINE tool 489
 \$REALTIMER tool 498
 \$RESETPRINT tool 504
 \$RPTIMMEDIATE tool 513
 \$RPTOCLIMIT tool 515
 \$RPTOVERLAP tool 517
 \$RPTPARMS tool 519
 \$RPTPRINT tool 521
 \$RPTSKIPLINES tool 523
 \$SETATTRIBUTE tool 597
 \$SETCHANNEL tool 599
 \$SETCOLOUR tool 600
 \$SETENVCOMMAREA tool 606
 \$SETOPT tool 609
 \$SETP#POS tool 614
 \$SETPRINT tool 617
 \$SETRPTATTRIBUTE tool 621
 \$SETRPTMEDIUM tool 623
 \$SETSESSIONEND tool 625
 \$SETTITLE tool 627
 \$SETTRANSACTION tool 629
 \$SHOWCHANNEL tool 632
 \$SKIPLINE tool 647
 \$SLEEP tool 649
 \$SYSTEMDATE tool 660, 728
 \$TOCPRINT tool 680
 \$TOCPUT tool 682
 \$TRXDATE tool 695
 \$TRXMODE tool 696
 \$TYPECAST tool 697
 \$UNPIC tool 718

A

ABS tool 34
 absolute values, returning 34
 adding
 fields to global field dictionary 219
 new user IDs 123
 ADMIN_RIGHTS tool 37
 allocating storage 346, 394
 ALLOCDSN tool 38
 applications, printing 531
 applying rules to tokens 432
 archiving audit log 483, 485
 attribute setting
 for reports 621
 for screen table fields 597
 audit log
 archiving 483
 purging 483, 485
 querying 40
 specifying archive file 485
 AUDITLOG tool 40

B

batch jobs
 queues, viewing status of 41
 submitting 41
 tools for 4
 batch load
 defining input for 56
 defining output for 56
 batch options, setting 69
 BATCH tool 41
 batch unload
 defining input for 71
 defining output for 71
 BATCH_ENABLE tool 55
 BATCHLOAD_CARDS tool 56
 BATCHUNLD_CARDS tool 71
 beeps, issuing 76
 binary data, returning a syntax V string 250
 blank lines, outputting 647

blank page, outputting 77
 BROWSER tool 81

C

CA-IDMS

defining databases 293
 displaying menu 293

calling

a functional rule 244
 a procedural rule 84

calling, menus 194

change requests, managing 511

CHANGE_SERVERID tool 86

CHANGERULE tool 88

character format, storing packed decimal data in 254

characters

removing leading 449
 removing leading or trailing 447
 removing trailing 451
 replacing 567

checking

if string is identifier 732
 if string is numeric literal 404

clearing data from tables 95, 637

CLEARTABLE_APPL tool 91

closing, files 94

COBOL, invoking language pre-processor for 286

color, setting for screen fields 600

column number, of cursor location 129

column, positioning cursor in 604

columns, number on physical screen 548

commands, defining for table display 405, 408

comparing

definitions 185, 188
 table data 182

components

installing 297
 uninstalling 699

CONFIRMATION tool 100

confirmation messages, issuing 100

container, deleting from channel 175

control cards, defining for secondary index builds 639

converting

any field to a string 730
 strings to date type 120
 strings to lowercase 339
 strings to lowercase EBCDIC 335
 strings to lowercase Unicode 337
 strings to typeless fields 318
 strings to uppercase 725
 strings to uppercase EBCDIC 721
 strings to uppercase Unicode 723
 variables 697

COPY_DATA tool 102

COPY_DEFN tool 104

COPYDEFN tool 108

copying

data 102
 definitions 104, 108
 rules 116
 selected table occurrences 117

COPYLIB tool 116

COPYTABLE_APPL tool 117

COUNTOCCURRENCES tool 119

CREATEUSERS tool 123

creating

member lists for object sets 341
 menus 156
 secondary indexes online 634

creating a buffer containing float values 253, 253

CROSSREFSEARCH tool 126

CTABLESIZE, returning maximum 202

current time, returning 497

current UTC time, returning 729

cursor

positioning in a field 602
 positioning in column 604
 returning
 column number of location 129
 field name 130
 occurrence number in screen table 132
 screen table names 137
 value of fields selected by 134

CURSOR_FLDCOL tool 129

CURSORFIELD tool 130

CURSOROCC_VALUE tool 134

CURSOROCC# tool 132

CURSORTABLE tool [137](#)

customer support [xxiii](#)

D

DASTATS tool [139](#)

data

comparing between tables [182](#)

copying [102](#)

deleting from tables [176](#)

loading [322](#), [325](#)

passing to non-OSB calling environments [606](#)

printing [463](#)

retrieving from non-OSB calling environments [273](#)

unloading [701](#)

unloading from tables [712](#)

Data Object Broker

recording message traffic [692](#)

tools for monitoring [5](#)

data sets, listing of [309](#)

database level, displaying [152](#)

DATACom tool [140](#)

Datacom, menu for managing [140](#)

date format, returning

maximum length [143](#)

dates

converting from type string [120](#)

returning transaction start date [695](#)

tools for manipulating [5](#)

DB2, defining static SQL for [652](#)

DBMAINTLVL tool [152](#)

DD names, allocating files to [38](#)

DEBUG tool [154](#)

debugging, invoking Rule Debugger for [154](#)

debugging, tools for [6](#)

defaults

server, resetting [506](#)

server, setting [630](#)

DEFINE_LIBRARY tool [155](#)

DEFINE_MENU tool [156](#)

DEFINE_OBJECTSET tool [157](#)

DEFINE_OBJLIST tool [168](#)

DEFINE_REPORT tool [173](#)

DEFINE_TABLE tool [174](#)

defining

CA-IDMS databases [293](#)

control cards for secondary index builds [639](#)

formats for text documents [672](#)

IMS/DB databases [294](#)

input for batch load [56](#)

input for batch unload [71](#)

libraries, new [155](#)

object list [168](#)

object sets [157](#)

output for batch load [56](#)

output for batch unload [71](#)

page number lines [614](#)

reports [173](#), [252](#)

rules [198](#)

screens [197](#)

tables [174](#)

definitions

comparing [185](#), [188](#)

copying [104](#), [108](#)

deleting [178](#)

loading [322](#), [325](#)

printing [465](#)

unloading [701](#), [714](#)

DELETE_DATA tool [176](#)

DELETE_DEFN tool [178](#)

DELETESCREENDATA tool [180](#)

deleting

container from channel [175](#)

data from tables [176](#)

definitions [178](#)

occurrences from table [91](#), [295](#)

screen table occurrences [180](#)

secondary indexes [644](#)

deleting data rows from tables [95](#), [637](#)

determining leap years [305](#)

DIFF_DATA tool [182](#)

DIFF_DEFN tool [185](#)

DIFFDEFN tool [188](#)

DISPLAY_MENU tool [194](#)

DISPLAY_USERS tool [195](#)

- displaying
 - list of libraries 155
 - list of rules 155
 - message log 332
 - messages on screens or windows 549
 - names and count of containers associated with a channel 79
 - options 414
 - PF keys available 216
 - statistics 289
 - table contents 81, 405, 408
 - tables for text editing 667
- displaying database level 152
- DRAW tool 197
- DSBIFTYPE Execution Environment parameter 56, 72, 320, 324, 327, 358, 495, 639, 708, 733

E

- editing
 - table occurrences 650
 - text 246
 - text in tables 667
- EDITRULE tool 198
- enabling object sets 55
- ENDMSG tool 199
- entering text 246
- ENTERKEY tool 200
- error messages from external DBMS, handling 577
- ESTIMATETBLDFN tool 202
- EVENTFIELD tool 205
- EVENTSCREEN tool 206
- EVENTSUBVIEW tool 209
- EVENTTABLE tool 210
- exceptions
 - returning error messages from 509
 - returning system error messages from 510
- Execution Environment
 - getting information on events 587
 - recording message traffic 692
- EXIT_DISPLAY tool 214
- EXIT_DISPLAY, signalling 214
- EXPOCC_SIZE tool 215

- external
 - databases, tools for manipulating 7
 - DBMS, handling error messages from 577
 - files, opening for reading or writing 411
 - memory, tools for 8
 - routines
 - returning return codes 508
 - tools for 8

F

- FCNKEY_MSG tool 216
- fields
 - converting to strings 730
 - in which cursor is located, returning 130
 - positioning cursor in 602
 - returning value when selected by cursor 134
 - screen table, setting attributes 597
 - screen, printing 481
 - screen, setting color 600
- files
 - allocating to DD names 38
 - closing 94
 - loading from files with mixed-case names 319
 - writing records to 733
- FLDMGR tool 219
- float representation, returning a syntax V string 253, 253
- footers, setting for output 627
- FORALLA tool 224
- FORALLB tool 238
- FORALLE tool 241
- formats of text documents, setting up 672
- formatting
 - numbers with masks 455
 - output 504
 - text 552
- FROM_UNICODE tool 243

functional categories of tools

- batch jobs [4](#)
 - CICS channels and containers [4](#)
 - Data Object Broker information and operations [5](#)
 - dates and times [5](#)
 - debugging [6](#)
 - external databases and servers [7](#)
 - external memory and routines [8](#)
 - load/unload to external files [9](#)
 - menus [10](#)
 - messages and message logs [10](#)
 - printing and output [11](#)
 - promotions [12](#)
 - read from/write to external files [13](#)
 - reports [13](#)
 - rules and rules libraries [14](#)
 - screens [15](#)
 - searches for objects [16](#)
 - secondary indexes [17](#)
 - security [17](#)
 - session options and parameters [18](#)
 - strings and text [18](#)
 - table definitions and data [21](#)
 - trigger or validation rules [23](#)
- functional rule, calling [244](#)

G

- GEN_TED tool [246](#)
- GENBIN tool [250](#)
- GENERATE_REPORT tool [252](#)
- generating an internal buffer [250](#)
- GENFLOAT tool [253](#)
- GENPACK tool [254](#)
- GETCHAR tool [267](#)
- GETENDMSG tool [271](#)
- global cross-reference index
 - rebuilding [499](#)
 - searching [126](#), [561](#)
- global field dictionary, adding fields [219](#)

H

- HEADSTRING tool [284](#)
- HLIPREPROCESSOR tool [286](#)
- HOUR tool [288](#)
- hour, returning [288](#)
- HURON_STATS tool [289](#)

I

- identifier, checking if string is [732](#)
- IDMS tool [293](#)
- IMS
 - displaying menu [294](#)
 - tool [294](#)
- IMS OTMA Callable Interface [419](#)
- IMS/DB database, defining [294](#)
- INDEXCHK tool [295](#)
- inserting
 - lines in table of contents [682](#)
 - strings in message log [396](#)
- INSTALLIB tool [299](#)
- installing components [297](#)
- integers
 - random, returning [491](#)
 - rounding [512](#)
- invoking
 - language pre-processor for COBOL [286](#)
 - Promotion facility on source system [344](#)
 - Promotion facility on target system [343](#)
 - Promotion system [478](#)
 - Query Audit Log tool [40](#)
 - rule that does nothing [403](#)
 - Security Manager menu [573](#)
 - Table Editor [653](#)
- issuing
 - beeps [76](#)
 - confirmation messages for PF keys [100](#)

J

joined tables, printing [662](#)

K

key, returning last used [200](#)

keyword index

 searching [303](#), [561](#)

 updating [300](#)

KEYWORDMGR tool [300](#)

keywords, searching with [126](#)

KEYWORDSEARCH tool [303](#)

L

leading characters, removing [449](#)

leap years [305](#)

LEAPYEAR tool [305](#)

LENGTH tool [307](#)

length, of strings, returning [307](#)

LIBID tool [308](#)

libraries

 defining new [155](#)

 displaying rules [155](#)

 unloading rules from [716](#)

limiting occurrences in reports [515](#)

lines, printing current [489](#)

listing data sets at a certain level [309](#)

listing members of a partitioned data set [314](#)

LIT_TO_VAL tool [318](#)

LLOAD tool [319](#)

LOAD tool [322](#)

LOADER tool [325](#)

loading

 data [322](#), [325](#)

 definitions [322](#), [325](#)

 from files with mixed-case names [319](#)

 tools for [9](#)

LOCALTIME tool [330](#)

location, remote

 returning [503](#)

 setting default [620](#)

LOG_BROWSE tool [332](#)

logging in

 to Session Manager [590](#)

 to workbench [590](#)

LOWER_EBCDIC tool [335](#)

LOWER_UNICODE tool [337](#)

lowercase EBCDIC, converting strings to [335](#)

LOWERCASE tool [339](#)

lowercase Unicode, converting strings to [337](#)

lowercase, converting strings to [339](#)

M

MANAGE_APPLY tool [343](#)

MANAGE_REQUESTS tool [344](#)

MANAGE_RIGHTS tool [345](#)

managing change requests [511](#)

MAP tables [346](#)

masks

 determining original value [718](#)

 formatting numbers [455](#)

MATCH tool [352](#)

matching patterns [446](#)

MAX tool [354](#)

member lists, creating [341](#)

members of a partitioned data set

 listing [314](#)

 retrieving statistics [314](#)

menus

 CA-IDMS [293](#)

 calling [194](#)

 creating [156](#)

 IMS [294](#)

 modifying [156](#)

 tools for [10](#)

merged reports, not printing selected tables or fields [517](#)

- message log
 - displaying 332
 - inserting lines into 396
 - preserving contents 357
- MESSAGE tool 355
- MESSAGE_LOG tool 357
- messages
 - displaying in screens or windows 549
 - indicating transaction completion 199
 - returning
 - end-of-transaction messages 271
 - error messages for exceptions 509
 - from MESSAGES table 355
 - tools for 10
- MESSAGES table 355
- MIN tool 368
- MINUTE tool 369
- minute, returning 369
- MOD tool 371
- modifying
 - menus 156
 - object sets 157
 - reports 173, 252
 - rules 198
 - screens 197
 - security profiles 370
 - session-related items 583
 - tables 174
- moving a container and its contents 390
- MOVTAB tool 391
- MQSMAP table 394
- MQSMAP_PORT table 394
- MSGLOG tool 396

N

- new
 - pages, positioning output on 398
 - user IDs, adding 123
- NLS bit setting for tables 608
- NLS tool 400
- nodes, copying definitions to and from 104
- nominating a channel for passing to a program or

- transaction 599
- NOOP tool 403
- NUM_CHK tool 404
- numbers
 - formatting with masks 455
 - returning absolute values 34
- numeric literals, checking 404

O

- object list, defining 168
- Object Manager 168
- object sets
 - creating member lists 341
 - defining 157
 - enabling 55
 - modifying 157
- OBJECT_MGMT tool 405
- OBJECTMGR tool 408
- objects, selecting 574
- occurrences, deleting from screen tables 180
- occurrences, returning selected 119
- opening external files for reading or writing 411
- OPSTATS tool 413
- OPTIONLISTER tool 414
- options
 - displaying 414
 - returning selected 414
- OTMA Callable Interface 419
- output
 - blank lines 647
 - controlling for reports 519
 - positioning on new pages 398
 - releasing to print spool 223
 - sending unsorted records 513
 - setting
 - arguments 617
 - formatting and medium 504
 - titles or footers 627
 - tools for 11
- outputting blank page 77
- overlapping pages, not printing report tables or fields on 517

P

- packed decimal, storing in character format [254](#)
- PAD tool [428](#)
- padding strings [428](#)
- page buffer, clearing table index from [241](#)
- page number lines, defining [614](#)
- parameter values, returning [430](#)
- parameters
 - server
 - configuration parameters, setting or modifying [98](#)
 - resetting [506](#)
 - setting [630](#)
 - session, setting [609](#)
 - tools for changing [18](#)
- PARMVALUE tool [430](#)
- PARSE tool [432](#)
- PARSE_TAM tool [443](#)
- partitioned data set, listing members [314](#)
- passing data to non-OSB calling environments [606](#)
- PATTERN_MATCH tool [446](#)
- patterns
 - matching [446](#)
 - replacing [567](#)
 - returning starting position of [352](#)
- PEEL tool [447](#)
- PEEL_HEAD tool [449](#)
- PEEL_TAIL tool [451](#)
- performance analysis [281](#), [289](#)
- PF keys
 - displaying available [216](#)
 - issuing confirmation messages [100](#)
 - processing [473](#)
- positioning cursor in column [602](#), [604](#)
- presentation environments, returning [462](#)
- PRINT_DATA tool [463](#)
- PRINT_DEFN tool [465](#)
- printing
 - applications [531](#)
 - blank lines [647](#)
 - definitions [465](#)
 - joined tables [662](#)
 - lines constructed by \$PRINTFIELD [489](#)
 - releasing output to spool [223](#)
 - reports to specified medium [521](#)
 - rules [531](#)
 - screen fields [481](#)
 - setting
 - medium for reports [623](#)
 - output arguments [617](#)
 - report attributes [621](#)
 - titles or footers [627](#)
 - strings [469](#)
 - table data [463](#)
 - table of contents [680](#)
 - tables [471](#), [662](#)
 - tools for [11](#)
- PRINTTABLE tool [471](#)
- problem determination [289](#)
- procedural rule, calling [84](#)
- PROCESS_FCNKEY tool [473](#)
- PROCESS_TABLE tool [474](#)
- processing
 - PF keys [473](#)
 - selected table occurrences [474](#)
- PROM_MAIN tool [478](#)
- Promotion facility
 - invoking on source system [344](#)
 - invoking on target system [343](#)
- promotion rights, obtaining, releasing, or transferring [37](#), [345](#)
- Promotion system, invoking [478](#)
- promotions, tools for [12](#)
- PRT_VSCR tool [481](#)
- PURGELOG_BATCH tool [483](#)
- PURGELOG_SCREEN tool [485](#)
- purging audit log [483](#)

Q

Query Audit Log tool, invoking [40](#)
 quotation marks
 adding to strings [490](#)
 removing from strings [720](#)
 QUOTE tool [490](#)

R

random integers
 returning [491](#)
 setting starting seed [493](#)
 RANDOM tool [491](#)
 RANDOMSEED tool [493](#)
 raw data
 converting [691](#)
 returning [243](#)
 RD fields, converting to strings [730](#)
 reading to external files, tools for [13](#)
 REALTIME tool [497](#)
 rebinding security data [477](#), [571](#)
 rebuilding global cross-reference index [499](#)
 record size, minimum needed for unload [215](#)
 recording message traffic [692](#)
 records
 returning next from file [495](#)
 sending to output without sorting [513](#)
 writing to files [733](#)
 REFMAKER tool [499](#)
 releasing
 print output [223](#)
 promotion rights [345](#)
 REMAINDER tool [501](#)
 remainders, returning [501](#)
 remote location
 returning [503](#)
 setting default [620](#)
 REMOTELOCATION tool [503](#)
 removing
 leading characters [449](#)
 trailing characters [451](#)
 replacing characters [567](#)

Report Generator, invoking [252](#)
 reports
 controlling physical output [519](#)
 controlling spacing [523](#)
 defining [173](#), [252](#)
 limiting number of occurrences in [515](#)
 modifying [173](#), [252](#)
 not printing selected tables or fields [517](#)
 print to specified medium [521](#)
 setting attributes [621](#)
 setting print medium [623](#)
 tools for [13](#)
 resetting
 default field values for servers [506](#)
 server parameters [506](#)
 retrieving
 data associated with specified channel
 container [256](#)
 number of decimal places for an expression [258](#)
 retrieving data from non-OSB calling
 environments [273](#)
 retrieving statistics for a member of a partitioned data
 set [314](#)
 retrieving the current rule's transaction mode [696](#)
 return codes, returning [508](#)
 return the name of the designated installation
 library [299](#)
 return the name of the designated local library [308](#)
 return the name of the designated system library [661](#)
 RETURN_CODE tool [508](#)
 RETURN_MESSAGE tool [509](#)
 RETURN_SYSMMSG tool [510](#)

returning

- absolute value of a number 34
- column number of cursor location 129
- current time 497
- current UTC time 729
- end-of-transaction messages 271
- error messages for exceptions 509
- field in which cursor is located 130
- field values when selected by cursor 134
- first character from strings 267
- first selected occurrence 224, 231
- head of string 284
- hour 288
- larger of two values 354
- last key used 200
- local time 330
- maximum
 - CTABLESIZE 202
 - length of date format 143
 - number of rows on physical screen 551
 - XTABLESIZE 202
- messages from MESSAGES table 355
- minimum record size for unload 215
- minute 369
- next record from file 495
- next selected table occurrence 238
- number of columns on physical screen 548
- occurrence number, within screen table 132
- padded strings 428
- parameter values 430
- presentation environments 462
- random integers 491
- remainders 501
- remote location 503
- return codes 508
- rounded integers 512
- screen table with cursor 137
- screens activating validation rules 205, 206, 209
- second 572
- selected occurrences 119
- selected options 414
- session option values 276
- smaller of two values 368
- starting position of patterns 352
- strings

length 307

- with quotation marks 490
- without leading or trailing characters 447
- without quotation marks 720
- subsequence of strings 658
- system error messages for exceptions 510
- tables activating derived field, trigger, or validation
 - rules 210
- tail portion of strings 666
- time 679
- tokens 685
- transaction names 280, 629
- transaction start date 695
- user IDs 727
- year 748
- returning the name of the current channel 632
- rights, promotion, obtaining, releasing, or
 - transferring 37
- RMANAGE_REQUESTS tool 511
- ROUND tool 512
- rounding integers 512
- rows, maximum number on physical screen 551
- Rule Debugger, invoking 154
- RULEPRINTER tool 531
- rules
 - \$HTTPREQUEST 291
 - applying to tokens 432
 - copying 116
 - defining 198
 - modifying 198
 - printing 531
 - tools for 14
 - trigger, tools for 23
 - unloading 716
 - validation, tools for 23

S

- S6BCALL tool 537
- S6BFUNCTION tool 539
- S6BNOTIFY tool 541
- S6BTROFF tool 543
- S6BTRON tool 545

- screen tables
 - deleting all occurrences 180
 - positioning cursor in 602
 - returning
 - name with cursor 137
 - occurrence number of cursor 132
 - setting field attributes 597
- SCREENCOL tool 548
- SCREENMSG tool 549
- SCREENROW tool 551
- screens
 - activating validation rules 205, 206, 209
 - color of fields 600
 - defining 197
 - displaying messages 549
 - fields, printing 481
 - for selecting objects 574
 - modifying 197
 - physical
 - maximum number of rows on 551
 - returning number of columns on 548
 - processing PF keys 473
 - tools for 15
- SCRIPT tool 552
- search path for user exits 29
- SEARCH tool 561
- SEARCH_REPLACE tool 567
- searching
 - global cross-reference index 126, 561
 - keyword index 303, 561
 - tools for 16
- SEARCHLIB tool 568
- SEC_REBIND tool 477, 479, 571
- SECOND tool 572
- second, returning 572
- secondary indexes
 - creating online 634
 - defining control cards 639
 - deleting 644
 - tools for 17
- security
 - data, rebinding 477, 571
 - profiles, modifying 370
 - tools for 17
- Security Manager menu, invoking 573
- SECURITY tool 573
- segment numbers, changing 391
- SELECT_OBJ tool 574
- selecting objects 574
- server IDs, updating 86
- servers
 - directing table accesses to 453
 - resetting
 - default field values 506
 - parameters 506
 - setting
 - configuration parameters 98
 - default field values 630
 - parameters 630
 - tools for 7
- Session Manager, logging in to 590
- session options
 - returning value 276
 - tools for changing 18
- session parameters, setting 609
- sessions
 - altering related items 583
 - getting information on events 587
- SESSMGR tool 590
- SETCURSOR tool 602
- SETCURSOR_POS tool 604
- SETNLSBIT tool 608
- SETREMOTELOC tool 620
- setting
 - batch options 69
 - default
 - field values for servers 630
 - remote location 620
 - execution on return to non-OSB calling
 - environments 625
 - formats of text documents 672
 - print medium for reports 623
 - server configuration parameters 98
 - server parameters 630
 - session parameters 609
 - staring seed for random numbers 493
 - transaction completion messages 199
 - transaction names 629
- SETXPARAM tool 630
- signalling EXIT_DISPLAY 214

- SIXBUILD tool 634
- SIXBUILD_CARDS tool 639
- SIXDELETE tool 644
- SOE tool 650
- sound, issuing 76
- source system, invoking Promotion facility 344
- spacing of reports, controlling 523
- specifying archive file for audit log 485
- starting seed for random numbers 493
- static SQL, defining to access DB2 data 652
- statistics 289
- STE tool 653
- STEBROWSE tool 657
- storage, allocating 346, 394
- storing trace information 363
- strings
 - adding quotation marks 490
 - breaking into table specification tokens 443
 - breaking into tokens 432, 685
 - checking if identifier 732
 - converting to lowercase 339
 - converting to lowercase EBCDIC 335
 - converting to lowercase Unicode 337
 - converting to typeless fields 318
 - converting to uppercase 725
 - converting to uppercase EBCDIC 721
 - converting to uppercase Unicode 723
 - inserting in message log 396
 - numeric literals, checking 404
 - padding 428
 - parsing 685
 - pattern matching 446
 - printing 469
 - removing
 - leading characters 449
 - leading or trailing characters 447
 - trailing characters 451
 - removing first character 267
 - replacing characters 567
 - returning
 - head 284
 - length 307
 - starting position of patterns 352
 - subsequence 658
 - tail 666

- without quotation marks 720
- tools for manipulating 18
- writing to printline 467
- submitting batch jobs 41
- subsequence of string, returning 658
- SUBSTRING tool 658
- support, contacting xxiii
- SYSTEMLIB tool 661

T

- table
 - defining object list 168
 - deleting occurrences 91, 295
 - returning selected occurrences 119
- table accesses, directing to servers 453
- Table Editor, invoking 653
- table index, clearing 241
- table of contents
 - inserting lines 682
 - printing 680
- TABLEPRINT tool 662
- tables
 - activating derived field, trigger, or validation rules 210
 - changing segment number 391
 - clearing data 95, 637
 - comparing data between 182
 - copying
 - data to and from 102
 - occurrences 117
 - creating secondary indexes online 634
 - data, tools for changing 21
 - defining 174
 - definitions, tools for changing 21
 - deleting data 176
 - displaying contents 81, 405, 408
 - displaying names and count of containers associ-

- ated with a channel 79
- editing occurrences 650
- editing text 667
- formatting text in 552
- joined, printing 662
- modifying 174
- printing 463, 471, 662
- processing selected occurrences 474
- returning
 - first selected occurrence 224, 231
 - next selected occurrence 238
 - parameter values 430
- setting NLS bit 608
- unloading data 712
- viewing contents 657

tail of string, returning 666

TAILSTRING tool 666

target system, invoking Promotion facility 343

technical support xxiii

TED tool 667

text

- defining formats 672
- displaying tables for editing 667
- editing 246
- entering 246
- formatting commands 552
- removing first character in strings 267
- returning
 - head of string 284
 - starting position of patterns 352
- tools for manipulating 18

text editor 246

TEXTSETUP tool 672

TIBCO_HOME xx

time

- current UTC, returning 729
- current, returning 497
- local, returning 330
- returning
 - hour 288
 - minute 369
 - second 572
 - transaction start time 679
 - year 748

TIME tool 679

- times, tools for manipulating 5
- titles, setting for output 627
- TO_UNICODE tool 691
- TOKEN tool 685
- tokens
 - applying grammar rules to 432
 - parsing a string of table specification 443
 - returning 685
- tools
 - functional categories 2
 - general function of 2
- trace information
 - table access messages 363
 - writing to files 361
- tracing messages 692
- tracing, turning on and off 281
- trailing characters, removing 451
- transaction mode, retrieving 696
- transaction names
 - returning 280, 629
 - setting 629
- transferring promotion rights 345
- trigger rules
 - returning parameter values 430
 - tools for 23
- typeless fields, converting strings to 318

U

- UN fields, converting to strings 730
- unicode string
 - converting from 243
 - converting to 691
- uninstalling components 699
- UNLOAD tool 701
- UNLOAD_DATA tool 712
- UNLOAD_DEFN tool 714

- unloading
 - batch, defining input and output for 71
 - data 701
 - definitions 701, 714
 - returning minimum record size 215
 - rules 716
 - table data 712
 - tools for 9
- UNLOADLIBRARY tool 716
- UNQUOTE tool 720
- updating
 - keyword index 300
 - server IDs 86
- UPPER_UNICODE tool 721, 723
- uppercase EBCDIC, converting strings to 721
- UPPERCASE tool 725
- uppercase Unicode, converting strings to 723
- uppercase, converting strings to 725
- user exits 25
- user IDs
 - adding new 123
 - returning 727
- USERID tool 727
- users who are logged in, listing of 195
- UTCTIME tool 729

V

- VAL_TO_LIT tool 730
- VALID_NAME tool 732
- validation rules
 - returning parameter values 430
 - returning screens activating 205, 206, 209
 - tools for 23
- values
 - returning larger of two 354
 - returning smaller of two 368
- variable string fields, converting other fields to 730
- variables, converting 697
- viewing
 - list of users logged in 195
 - table contents 657

W

- windows, displaying messages in 549
- workbench, logging in to 590
- writing
 - records to files 733
 - strings to prinline 467
 - to external files, tools for 13
 - trace information to files 361

X

- XML
 - parsing 735
 - return table data to OIG 747
 - start processing 737
 - start processing to an output table 745
 - start processing to specified file 743
- XMLPARSE tool 735
- XMLSTART tool 737
- XMLSTARTDSN tool 743
- XMLSTARTSETDEST tool 745
- XMLSTARTTAB tool 747
- XTABLESIZE, returning maximum 202

Y

- YEAR tool 748
- year, returning 748