# Table of Contents

## Foreword
0

## Part I Preface
16

## Part II Installation Guide
18

1 Requirements ......................................................................................................................... 18
2 Installing ................................................................................................................................. 19
3 Upgrading ............................................................................................................................... 21
4 Runtime Configuration .......................................................................................................... 22
5 Embedded web server .......................................................................................................... 26
6 Samples .................................................................................................................................. 27
7 Third party tools .................................................................................................................... 28
8 Licensing ................................................................................................................................. 30

## Part III XCICS Transaction Server
32

1 XCICS System Management Guide .................................................................................. 33
The architecture ......................................................................................................................... 34
Configuration reference ........................................................................................................... 36
The configuration file ............................................................................................................... 37
Region parameters .................................................................................................................. 38
General ...................................................................................................................................... 38
Region tuning ............................................................................................................................ 39
Networking ............................................................................................................................... 42
Programs and modules loading ............................................................................................. 43
Files and databases .................................................................................................................. 44
Compatibility ........................................................................................................................... 45
Security ..................................................................................................................................... 47
Logging and journaling ........................................................................................................... 48
Printing and spooling .............................................................................................................. 50
Customization .......................................................................................................................... 51
Debugging ................................................................................................................................. 51
IMS/DC compatibility TK ......................................................................................................... 52
WebSphere/MQ ......................................................................................................................... 52
LDAP ......................................................................................................................................... 53
Deprecated settings .................................................................................................................. 54
Web Interface ........................................................................................................................... 56
Application resources .............................................................................................................. 58
TCT - Terminal Control Table ................................................................................................. 58
FCT - File Control Table .......................................................................................................... 60
PPT - Program Processing Table ............................................................................................ 62
PCT - Program Control Table ................................................................................................. 63
DCT - Destination Control Table ........................................................................................... 64
MPS - Mapset Table ................................................................................................................. 65
CCT - Communication Control Table ..................................................................................... 65
JCT - Journal Control Table .................................................................................................... 67
SNT - Sign-on Table .................................................................................................................. 67
Logging, Accounting & Journaling

TCP/IP services

Intersystem Communication

Print services

Data integrity

Security management

Region control

Tuning

Other

© 2007 H.T.W.C. Srl
Contents

XCICS Messages .......................................................................................................................... 158
Accounting ................................................................................................................................. 163
Tracing ........................................................................................................................................ 164
Journaling .................................................................................................................................. 165
Command line utilities ............................................................................................................... 166
Customization Guide .................................................................................................................. 167
Exit programs .............................................................................................................................. 168
odcs_ts routine .......................................................................................................................... 169
sm Loader routine ...................................................................................................................... 169
user_signon routine .................................................................................................................. 170
user_signoff routine .................................................................................................................. 171
user_chpasswd routine ............................................................................................................. 172
dfhaid routine .......................................................................................................................... 173
On-line data conversion ............................................................................................................ 175
MQ Trigger Monitors ............................................................................................................... 179
Programming considerations .................................................................................................... 181
2 XCICS Programming Guide .................................................................................................. 182
Programming guide ................................................................................................................... 183
General rules .............................................................................................................................. 184
Programming in COBOL ............................................................................................................ 187
Programming in C ....................................................................................................................... 191
Programming in PL/I .................................................................................................................. 195
Programming in Java .................................................................................................................. 196
Compiling programs .................................................................................................................... 198
Application design ..................................................................................................................... 201
Debug .......................................................................................................................................... 204
COBOL programs ...................................................................................................................... 204
C language programs .................................................................................................................. 208
PL/I programs ............................................................................................................................ 209
BMS support .............................................................................................................................. 210
Commands reference ................................................................................................................ 211
ALLOCATE ................................................................................................................................. 212
ASSIGN ....................................................................................................................................... 213
ASRTIME ..................................................................................................................................... 216
ABEND ....................................................................................................................................... 217
BIF DEEDIT ............................................................................................................................... 218
CANCEL ....................................................................................................................................... 219
CONNECT PROCESS ............................................................................................................... 220
CONVERSE ............................................................................................................................... 222
CONVERTTIME .......................................................................................................................... 225
DELAY ......................................................................................................................................... 226
DELETE ....................................................................................................................................... 227
DELETEQ TD .............................................................................................................................. 228
DELETEQ TS ............................................................................................................................... 229
DEQ ............................................................................................................................................. 230
DOCUMENT CREATE ............................................................................................................... 231
DOCUMENT INSERT ............................................................................................................... 235
DOCUMENT RETRIEVE ............................................................................................................ 238
DOCUMENT SET ....................................................................................................................... 240
DUMP ......................................................................................................................................... 243
ENDBR ....................................................................................................................................... 244
ENTER ......................................................................................................................................... 245
ENQ ........................................................................................................................................... 246
EXTRACT ATTRIBUTES ........................................................................................................... 247

© 2007 H.T.W.C. Srl
8 XFRAME User’s Guide

HTTP requests and response processing (as server) ............................................................... 426
HTTP requests and response processing (as client) ............................................................ 427
Session tokens ....................................................................................................................... 428
URLS for XCICS Web Support ............................................................................................. 429
Chunked transfer-coding ........................................................................................................ 430
Persistent connections ........................................................................................................... 431
XCICS HTTP/1.1 compliance .............................................................................................. 431
Planning the architecture for XCICS as HTTP server ......................................................... 434
Providing dynamic HTTP responses with Web-aware application programs .................... 434
Providing static HTTP responses with a XCICS document template or flat file .................. 435
Giving Web clients access to COMMAREA applications .................................................... 436
Writing Web-aware application programs for XCICS as an HTTP server ......................... 439
Examining the request line for an HTTP request ................................................................. 440
Examining the HTTP headers for a message ........................................................................ 441
Retrieving technical and security information about an HTTP request ............................... 441
Examining form data in an HTTP request ............................................................................ 442
Receiving the entity body of an HTTP request ..................................................................... 442
Writing HTTP headers for a response ................................................................................... 443
Producing an entity body for an HTTP message ................................................................. 444
Sending an HTTP response from CICS as an HTTP server ................................................. 444
Using chunked transfer-coding to send an HTTP request or response ............................. 445
Managing application state across an HTTP request sequence ........................................... 446
Resource definition for XCICS Web support ....................................................................... 448
Creating TCP/TPSERVICE resource definitions ............................................................... 448
Creating TRANSACTION resource definitions ................................................................. 449
Starting a URIMAP resource definition for any requests for XCICS as HTTP server .......... 450
Completing a URIMAP definition for an application response to an HTTP request for XCICS as HTTP server .............................................................. 451
Completing a URIMAP definition for a static response to an HTTP request for XCICS as HTTP server .............................................................. 452
Analyzer programs .............................................................................................................. 453
Writing an analyzer program .............................................................................................. 454
Sharing data between analyzer and converter programs ...................................................... 456
XCICS-supplied default analyzer program DFHWBAHX ...................................................... 457
Reference information for analyzer programs ....................................................................... 458
Converter programs ............................................................................................................ 462
Writing a converter program ............................................................................................... 463
Converter programs: Constructing an HTTP response ....................................................... 464
Reference information for converter programs .................................................................... 464
Security for XCICS Web support ....................................................................................... 468
HTTP client requests from a CICS application ..................................................................... 471
Making HTTP requests through CICS as an HTTP client ................................................... 471
Creating a URIMAP definition for an HTTP request by CICS as an HTTP client ... ......... 474
XCICS Web support and non-HTTP requests ..................................................................... 475
Handling non-HTTP requests .............................................................................................. 475
Document handler guide .................................................................................................... 477
Creating a document .......................................................................................................... 478
Setting up document templates ......................................................................................... 479
Programming with documents and document templates .................................................. 480
Symbols and symbol lists .................................................................................................... 480
Setting symbol values .......................................................................................................... 480
Embedded template commands .......................................................................................... 481
Contents

Part IV X4J 3270 Terminal Emulator 541

1 The menuing system .......................................................... 542
2 The toolbar ....................................................................... 546
3 Terminal Display Area ...................................................... 547
4 The Keyboard .................................................................... 548
5 Connect dialog ................................................................... 549
6 Sessions dialog .................................................................. 550
7 Options dialog ................................................................. 551
8 Using macro ...................................................................... 558
9 Miscellaneous .................................................................... 560
10 Configuration ..................................................................... 561
11 Running X4J ...................................................................... 569
12 Using Java WebStart ....................................................... 570
13 Session and settings deployment .................................... 572

Part V XVSAM File System 577

1 Concepts ............................................................................. 578
   File types ........................................................................ 579
   Alternate indexes ........................................................... 580
Part VI

1 Scripts logic ......................................................................................................................... 641
   EXEC ................................................................................................................................. 642
   DD & DLBL ....................................................................................................................... 643
   Data Cards ....................................................................................................................... 644
   PROC parameters substitution ....................................................................................... 646
   SORT ................................................................................................................................. 647
   FLAT & VSAM ................................................................................................................. 648
   GDG (Generation Data Group) ....................................................................................... 651

2 Using XVSAM .................................................................................................................... 585
   Functional commands ..................................................................................................... 588
   Creating a file .................................................................................................................. 589
   Browsing the repository ................................................................................................. 590
   Erasing and deleting ....................................................................................................... 591
   Import and Export ......................................................................................................... 592
   Repro ............................................................................................................................... 593
   Warm start and recovery ............................................................................................... 594
   Backup and restore ....................................................................................................... 595

3 Utilities .............................................................................................................................. 597
   Command line utilities ................................................................................................. 597
      xvsam ......................................................................................................................... 597
      Backward compatibility ............................................................................................ 601
      xvsamCreate ............................................................................................................. 601
      xvsamDelete ............................................................................................................. 603
      xvsamErase ............................................................................................................. 603
      xvsamImport ............................................................................................................. 604
      xvsamExport ............................................................................................................ 605
      xvsamLog .................................................................................................................. 607
      xvsamInfo ................................................................................................................ 608
      xvsamRepro .............................................................................................................. 608
      xvsamEdit ............................................................................................................... 611
      xvsamRecover ......................................................................................................... 612
      xvsamWarmStart ....................................................................................................... 612
      xvsamRts .................................................................................................................. 614
      xvsamAdm .................................................................................................................. 617
      Creating a cluster ...................................................................................................... 619
      Creation of an alternate index ................................................................................... 621
      Selection of a file ...................................................................................................... 623
      Cluster edit ............................................................................................................... 626
      Import of data ......................................................................................................... 628
      Exporting a file ....................................................................................................... 630
      Erasing a file ............................................................................................................ 631

4 Programmer’s reference .................................................................................................... 633
   XVSAM logical structure ............................................................................................ 633
   XVSAM programming interface .................................................................................. 634
   Debugging .................................................................................................................... 636
   V2R - Database storage system ................................................................................... 637

Part VI Batch Environment .................................................................................................. 640

© 2007 H.T.W.C. Srl
Part IX  XCONV Data Conversion Toolkit  784

1 XML generation .............................................................. 787
2 Converter generation ......................................................... 788
3 Rules definition ................................................................. 790
Part XI  XFRAME Unified Console  846

1 Installation ................................................................. 847
2 Desktop Elements .......................................................... 849
3 Connections .................................................................. 853
4 Options ........................................................................ 854
5 XCICS/TS administration module ...................................... 859
   Configuration .................................................................. 860
   Operations ....................................................................... 862
   Control panel - Status ......................................................... 863
   Terminals ........................................................................ 866
   Files .............................................................................. 868
   Users ............................................................................ 869
   Transactions .................................................................... 870
   Programs ........................................................................ 871
   Transient data .................................................................. 872
   Connection ...................................................................... 873
   Mapsets .......................................................................... 874
   Journals .......................................................................... 875
   Urimap ........................................................................... 876
   Documents template .......................................................... 877
   TCP/IP services ............................................................... 878
   TN3270 services .............................................................. 879
   MQ trigger monitor .......................................................... 880
   User shared memory ......................................................... 881
   Processes ......................................................................... 882
   Console log ...................................................................... 883
   Monitor utility .................................................................... 884
## Part XII  Programmer’s Reference  923

### 1 Utilities  924
- xcb  924
- xpli  931
- xpre  933
- xbms2mod  936

### 2 Debug  939
- Batch  939
  - Debugging Batch programs  940
- Online  943
  - Debugging COBOL  944
  - Debugging C  949
  - Debugging PL/I  951

### 3 Code changes  952
- EBCDIC Collating Sequence  955
- DB2 to Oracle  957

## Part XIII  Notices  963

## Index  0
Preface
1 Preface

This book's collection is intended to help you to operate in the XFRAME 4.1.1 environment. It contains guidance about configuring and using XCICS/TS, XVSAM, XSORT, XBM, XSPOOL and all the XFRAME products.

It contains information and reference guides for both programmers and system programmers who operate in the XFRAME environment.

**Who should read this book**

The book collection is intended primarily for use by system and application programmers, but will also be useful to migration analysts.

**What you need to know to understand this book**

For all the administrative tasks such installation, configuration and operation, we assume that you have some basic knowledge of the UNIX/Linux operating system. For XCICS/TS configuration, we assume that you have some knowledge about IBM CICS (TM) administration.

For the programming tasks, we assume that you have some experience in writing CICS programs in COBOL, C or PL/I.
Part II

Installation Guide
2 Installation Guide

2.1 Requirements

To install XFRAME, the following software is required:

<table>
<thead>
<tr>
<th>Category</th>
<th>Choices</th>
<th>Required(optional)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Hewlett-Packard HP-UX V.10.20 or 11.x</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>Sun Solaris V.2.7 or higher</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM AIX 4.x or 5.x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SuSE Linux Enterprise Server 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SuSE Linux Enterprise Server 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RedHat Linux ES 3</td>
<td></td>
</tr>
<tr>
<td>COBOL compiler</td>
<td>MicroFocus ServerExpress 2.0 or higher</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>ACUCOBOL Extend 7</td>
<td></td>
</tr>
<tr>
<td>C compiler</td>
<td>HP ANSI-C</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>Sun Forte’</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IBM VAC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GNU GCC</td>
<td></td>
</tr>
<tr>
<td>Java Development Kit</td>
<td>J2SE 1.4 (JDK 1.4)</td>
<td>required</td>
</tr>
<tr>
<td>Informix C-ISAM</td>
<td>Informix C-ISAM version 7.25 or higher</td>
<td>required</td>
</tr>
<tr>
<td>PERL interpreter</td>
<td>PERL 5.6 or higher</td>
<td>required</td>
</tr>
<tr>
<td>PL/I compiler</td>
<td>Liant OpenPL/I</td>
<td>optional</td>
</tr>
<tr>
<td>Oracle client libraries</td>
<td>Oracle 8i (8.1.x)</td>
<td>optional</td>
</tr>
<tr>
<td></td>
<td>Oracle 9i (9.0.x)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Oracle 9i release 2 (9.2)</td>
<td></td>
</tr>
<tr>
<td>Oracle precompiler</td>
<td>pro cob 8.x or 9.x</td>
<td>optional</td>
</tr>
<tr>
<td></td>
<td>proc 8.x or 9.x</td>
<td></td>
</tr>
<tr>
<td>IBM UDB Appl. Dev. Client</td>
<td>version 8.2</td>
<td>optional</td>
</tr>
</tbody>
</table>

XFRAME requires about 200 Mb of free disk space (user programs and data not included).
2.2 Installing

Once controlled that all the required software components have been correctly installed on your system, you may start with the XFRAME set up.

Each XFRAME installation should belong to a dedicated non-root user and contained in a dedicated directory.

By default, XFRAME is installed in /opt/xframe.

More than one XFRAME instances may be installed on one system; just install them in different users and directories. It means you may have different versions of XFRAME and same system, and let users to use the one they need.

A sample scenario: XFRAME v4.0.0 installed as user “xframeprod” in /opt/xframe-prod and used by the production environment, and XFRAME v4.0.5 installed as user “xframetest” in /opt/xframe-test and used by the testing team.

IMPORTANT: before to start with the XFRAME setup, please take a look to the README files deployed with the XFRAME distribution and contained in XFRAME setup tarball: they should contain important notices about your platform or some of the third-party tool you are going to use with XFRAME.

**Task to perform as root user**

In order to install XFRAME the following task must be done as root user:

1. create a group named “xframe”
2. create a user named “xframe”, belonging to the xframe group. We suggest to assign ”/opt/xframe” as the home directory. This user must use KSH or BASH and have umask set to 022.
3. check for the owner and group of the assigned home directory, to be sure they are consistent with the physical user and group identifiers.

**Task to perform as the xframe user**

1. log in as the “xframe” user
2. check if required software is available for the “xframe” user. To do this, try issuing the following command:

   ```
   # cc
   # perl -v
   # echo $JAVA_HOME
   # java -version
   # javac
   ```

   If the COBOL compiler is Microfocus COBOL:

   ```
   # cob
   # cobrun
   ```

   If the COBOL compiler is ACUCOBOL:

   ```
   # cobl
   # runcbl
   ```

   If ORACLE(tm) RDBMS is required, issue:

   ```
   # echo $ORACLE_HOME
   ```
If DB2™ RDBMS is required, issue:

```bash
# echo $DB2_HOME
# db2
```

3. Identify the XFRAME distribution archive required and copy it from your distribution media, or directly download it from http://www.htwc.it in a temporary directory on your system.

4. Decompress the archive using gzip and tar, and run the installation script 'install.ksh'.

```bash
# tar xf xframe.tar
# mkdir /tmp/xframe-setup-3.3.10
# cd /tmp/xframe-setup-3.3.10
# mv /some/where/xframe-AIX-PPC-32-3.3.10.tgz .
# gzip -d xframe-AIX-PPC-32-3.3.10.tgz
# tar xf xframe-AIX-PPC-32-3.3.10.tar
# ksh ./install.ksh
```

Then follow the install instructions provided by the installation procedure. Installation procedure will show the machine serial number. Provide this number to HTWC to obtain the correct license key.
2.3 Upgrading

The XFRAME upgrading process is almost equal to the install one:

- download the distribution you want to install
- decompress the tarball in a temporary directory
- run "ksh install.ksh"

The install procedure will ask for the configuration details, proposing the value you selected during the last installation as default values.

Please remember to verify than no process is running XFRAME software, before to start upgrading it.

Reconfiguring

Sometimes you need to reconfigure and relink XFRAME, because you need to change the S/W configuration (i.e. you want to relink with a different RDBMS system).

To reconfigure XFRAME, you simply have to re-install the version you have currently installed and provide different configuration details.
2.4 Runtime Configuration

The runtime configuration consist in the creation of all those executable files that require to be recompiled and/or linked on the target system: during this task some XFRAME components are rebuilt to get access to third party softwares (i.e. compilers, databases, etc.).

The runtime configuration is almost automated and it is divided in two phases:

- configuration
- link

During the first phase, the XFRAME configuration tool asks you for all the information it needs to rebuild the XFRAME components (i.e. which components must be linked, where to find some libraries, etc.). During the link phase the XFRAME rebuild tool analyzes the information collected in the first phase and recompiles and links whatever component must be rebuilt.

The runtime configuration may run automatically or manually.

**Automatic runtime configuration**

To execute it automatically, simply install (or re-install) XFRAME according to the installation instructions: during the setup procedure, the configuration tool is started and, when all the information have been collected, the link phase is automatically performed.

If any problem arises during this operation, the setup procedure stops and shows where to find the log of the procedure that caused the problem.

**Manual runtime configuration**

If you want to manually configure the runtime you may follow this procedure:

- login as the user that own XFRAME components (normally xframe)
- change directory to $XFRAMEHOME
- run the configuration tool with the command:
  
  ```
  perl etc/xframeConfig
  ```
- reply to all the questions, according to your environment
- run the relink tool with the command:
  
  ```
  ksh etc/xframeLink
  ```

If no error occurs during this phase, XFRAME is correctly rebuilt, otherwise check the xframeLink.log file to discover the cause of the failure.

**Third party software**

As described before, the runtime configuration is useful to add to the base XFRAME runtime all those components required to access the resources of third party softwares.

This components includes:

- COBOL compilers runtime
- PL/I compilers runtime
- Databases client libraries (i.e. Oracle, UDB)
• APPC Inter System Communication libraries (i.e. SNAP/IX, IBM/CS, HPSNAplus)
• MQ/Series client libraries
• LDAP client libraries

Depending on the component you are linking, the XFRAME configuration tools asks for the information it needs. The following paragraphs describe the information requested and the relative options.

**COBOL runtime**

XFRAME requires a COBOL runtime. This setting configures the type of COBOL runtime to be linked:

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>se10</td>
<td>the runtime is linked with Microfocus ServerExpress 1.0 runtime libraries. The environment variable COBDIR must be correctly set.</td>
</tr>
<tr>
<td>se20</td>
<td>the runtime is linked with Microfocus ServerExpress 2.0 runtime libraries. The environment variable COBDIR must be correctly set.</td>
</tr>
<tr>
<td>se40</td>
<td>the runtime is linked with Microfocus ServerExpress 4.0 runtime libraries. The environment variable COBDIR must be correctly set.</td>
</tr>
<tr>
<td>se50</td>
<td>the runtime is linked with Microfocus ServerExpress 5.0 runtime libraries. The environment variable COBDIR must be correctly set.</td>
</tr>
<tr>
<td>acu</td>
<td>the runtime is linked with ACUCOBOL runtime libraries. The environment variable ACUCOBOL must be correctly set.</td>
</tr>
</tbody>
</table>

This is the only mandatory component.

**Liant Open/PLI runtime**

The runtime must be linked with PL/I runtime if the user application contains programs written in PL/I.

If this option is set to "yes", the runtime is linked with the Liant Open/PLI runtime libraries. The environment variable LPI_PRODUCT_DIR must be correctly set.

**LU62/APPC**

If the user applications require to communicate over a SNA network, the runtime must be linked with a valid APPC APIs library. This setting configures the type of SNA communication product to be linked:

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no APPC communication is required.</td>
</tr>
<tr>
<td>snapix</td>
<td>the runtime is linked with DataConnection SNAP/IX APPC libraries</td>
</tr>
<tr>
<td>ibmcs</td>
<td>the runtime is linked with IBM Communication Server APPC libraries</td>
</tr>
</tbody>
</table>
If you specify a valid communication product, the tool will ask for the product installation base and for the product's include files path.

I.e.

```plaintext
using L/U2 ( none| snapix | snaplus | ibmcs | tpssna) [ snapix ] : snapix
SNAP/IX(TM) installation path [/opt/sna] : /opt/sna
SNAP/IX(TM) include path [/usr/include/sna] : /usr/include/sna
```

### IBM MQ/Series interface

If the user applications require to communicate with an MQ/Series (AKA WebSphere/MQ) system, the runtime must be linked with the MQ APIs libraries. This setting configures if and how the MQ libraries must be linked:

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>no MQ series library is linked</td>
</tr>
<tr>
<td>server</td>
<td>links the runtime with the MQ server libraries.</td>
</tr>
<tr>
<td>client</td>
<td>links the runtime with the MQ client libraries.</td>
</tr>
</tbody>
</table>

When the MQ/Series server runs on the same system where XFRAME runs, both "server" and "client" configuration may be used.

When MQ/Series server runs on a remote system only "client" configuration may be used: in this case the MQ/Series client component must be installed and configured on the system where XFRAME runs.

The environment variable MQSERIES_HOME must be correctly set up.

### RDBMS interface

If the user applications require to communicate with a database system, the runtime must be linked with the database client libraries. This setting configures if database libraries must be linked. If you reply "yes", the configuration tools asks for the type of database to use:

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ora805</td>
<td>the runtime is linked with Oracle 8.0.5 client libraries.</td>
</tr>
<tr>
<td>ora817</td>
<td>the runtime is linked with Oracle 8.17 client libraries.</td>
</tr>
<tr>
<td>ora901</td>
<td>the runtime is linked with Oracle 9.0.1 client libraries.</td>
</tr>
<tr>
<td>ora9i</td>
<td>the runtime is linked with Oracle 9iR2 client libraries.</td>
</tr>
<tr>
<td>ora10g</td>
<td>the runtime is linked with Oracle 10g client libraries.</td>
</tr>
<tr>
<td>udb8</td>
<td>the runtime is linked with IBM UDB 8.2 client libraries.</td>
</tr>
<tr>
<td>odbc</td>
<td>the runtime will use ODBC to connect the database</td>
</tr>
</tbody>
</table>
When the DB types has been chosen, you have to configure the interface type:

<table>
<thead>
<tr>
<th>option</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>native</td>
<td>the runtime will use the RDBMS native calls to connect and handle the LUW (Logical Unit of Work)</td>
</tr>
<tr>
<td>xa</td>
<td>the runtime will use the XA interface to connect and handle the LUW.</td>
</tr>
</tbody>
</table>

At the end a suitable database connection string is requested to compile the DB dependant objects. The string must be entered in the format user/password@database.

**LDAP Interface**

If you want to enable the inter-operation between XCICS/TS and an LDAP server, an LDAPv3 client must be installed on the system, to allow the XCICS LDAP interface compiling and to be linked to the XFRAME runtime.

If you choose to link with LDAP, you have to provide the full path to the directory where the LDAP include file(s) may be found (ldap.h) and all the libraries (or ld linker options) required to link the LDAP client.

These are some samples:

**Linking with OpenLDAP client:**

| LDAP authentication interface (yes|no) [no] : yes | LDAP include files path [] : /usr/local/include |
| LDAP libraries [] : /usr/local/lib/hpux32/libldap.so |

**Linking with Mozilla LDAP client:**

| LDAP authentication interface (yes|no) [no] : yes | LDAP include files path [] : /opt/mozilla/dist/public/ldap |
| LDAP libraries [] : /opt/mozilla/dist/lib/libldap50.so |

**Linking with Oracle LDAP client:**

| LDAP authentication interface (yes|no) [no] : yes | LDAP include files path [] : /opt/oracle/product/9.2.0/ldap/public |
| LDAP libraries [] : /opt/oracle/product/9.2.0/lib/libldapclnt9.a |

All the client libraries in the samples above make use of only one library for linking. If your client libraries requires more than one library, you may configure it adding more than one library with space as separator:

| LDAP libraries [] : /my/ldap/lib/libldap1.so /my/ldap/lib/libldap2.so |

or use "ld" options

| LDAP libraries [] : -L /my/ldap/lib -lldap1 -lldap2 |
2.5 Embedded web server

XFRAME delivers with an embedded Tomcat web server, which make accessible the XFRAME documentation shipped with the distribution and offers the possibility to install, via Java WebStart, the XFRAME client s/w (i.e. X4J, XBM remote Console, etc.).

The embedded web server must serve no other purpose than those specified herewith.

At installation time, during the configuration phase, the procedure will ask for the web server configuration.

By default the web server operates on port 8080.

The embedded webserver may be started with the command:

```
tomcat start
```

and stopped with :

```
tomcat stop
```

During the install procedure, the web server is automatically shut down.
2.6 Samples

XFRAME distribution contains sample programs, configurations and files to demonstrate the usage of specific features offered by the product.

These files are located in the $XFRAMEHOME/samples directory grouped by subject.
2.7 Third party tools

This a list of software which has been tested and certified to run with XFRAME.

**Operating Systems**
- IBM AIX 4.3
- IBM AIX 5.x
- HP HP-UX 11.11
- HP HP-UX 11.23
- Solaris 8
- Solaris 9
- RedHat Linux Enterprise Server 3
- SuSE Linux Enterprise Server 8
- SuSE Linux Enterprise Server 9

**COBOL Compilers**
- MicroFocus Server Express 2.0 and 4.0
- ACUCOBOL Extend 7

**PL/I compilers**
- Liant Open PL/I 7

**Java Development Kit**
- JDK 1.4.x

**Oracle database**
- Oracle 8.0.x
- Oracle 8.1.7
- Oracle 9.0.1
- Oracle 9.2
- Oracle 10.1
- Oracle 10.2

**IBM DB2 UDB database**
- IBM DB2 UDB 8.1
• IBM DB2 UDB 8.2

**IBM WebSphere/MQ**

• IBM WebSphere/MQ 6
2.8 Licensing

The XFRAME system is protected against unauthorized usage by means of a license file depending on your contract number and on the CPU identifier of your system.

Normally the license file will be communicated to you at the moment of installation. In case you have not received it, please contact HTWC.

The following instructions will explain how to set it up:

1. the installation procedure supplies the system information file. If you need to obtain it once again, you can get it by means of the following command:

   ```
   # xkey -q
   ```

2. send the system information file (system.inf) to HTWC: a new license file for XFRAME based on system information will be delivered

3. once obtained your license file (license.dat), put it in the directory:

   ```
   $XFRAMEHOME/license
   ```

4. log in once again to check if the password has been set up correctly and if the expiration date is set to unlimited;

Warning: In case of hardware updating or installation of a new system, do not forget to contact previously HTWC to receive the license file for your new system identifier.
3  XCICS Transaction Server

XCICS/TS is a software product which allows COBOL, C, Java and PL/I applications developed to operate with IBM CICS APIs, to run under UNIX/Linux operating systems.

XCICS/TS provides a cost-effective and manageable transaction processing system, allowing you to write your own applications or to choose from many existing vendor-written business products.

The XCICS Transaction Server supports on one hand the production environment of transactions based on CICS/COBOL, CICS/C and CICS/PLI applications under UNIX, which run without any source-code modification. On the other hand it offers an excellent development environment for new applications which can run either on UNIX systems or on mainframes.

Furthermore the product is suitable as a dedicated programming environment for further development and maintenance of existing applications. This reduces the workload for such tasks on mainframes considerably.

**Note**

This manual refers to XCICS/TS version 8, distributed as part of XFRAME version 4.
3.1 XCICS System Management Guide

This book is intended to help you operate XCICS/TS regions in an UNIX or Linux environment. It contains guidance about configuring and managing XCICS/TS regions in an UNIX/Linux environment. It also contains guidance about how to use the CICS batch utility programs.
3.1.1 The architecture

Each XCICS/TS region is composed a set of processes: when an instance is started up, the main process takes over and generates all the child processes that compose the instance.

**Main process (main)**

It is the central supervisor process. It manages and takes care about the resolution all conflicts among common resources (queue, etc.). It detects and handles every eventual failure on its child processes, and it provides to generate new processes whenever the needs arise.

**Terminal handler process (netd)**

This process provides to handle connections and communication requestes coming from terminals.

**Transaction server processes pool (engn)**

Every XCICS region has a pool of serving processes, so called "engines". The number of processes in this pool is defined in the XCICS configuration files and it may be dynamically changed by means of the `xcicsadm` utility.

Every time a transaction is requested by a terminal, the main process identifies the first available process and assign the request to this one, which will take care to run the user code and to provide XCICS services.

If the main process, do not find any process available to satisfy the request, this one is put in a FIFO queue waiting for the first available process in the pool.

The size of the processes pool strictly depends on:

- number of terminals connected
- transaction complexity and duration
- H/W characteristics

For user applications with an average duration, we suggest to define one engine process every 100 terminals connected, but, of course, only a correct tuning and sizing, may lead to an effective configuration.

**Background processes (task)**

Any time a background task is requested (i.e. a transient data trigger or a printer task), XCICS forks a new process to handle the background transaction.

Under some circumstances, like TCP and Web Services driven transactions, background task may be started also on engines pool processes, to avoid the initial delay of the fork system call.

**SNA session listeners**

This is a particular type of background process, which listens over SNA LU connection, to serve incoming request from remote region over SNA (CICS and XCICS).

**TCP/IP services**

This is a particular type of background process, which listens over a TCP/IP socket to handle all the incoming request for TCP/IP services (EZASOCKET, TCP ISC, etc.)
**TN3270 services**

This is a particular type of background process, which serve TN3270 connections over TCP/IP.

**File Access**

File accesses are controlled and supervised by the XVSAM subsystem, which simulates the IBM’s original VSAM storage access method. Every XCICS/TS embeds the XVSAM file access layer and perform I/O autonomously, without the need of an external I/O server.

**Inter-process Communication**

Communication between the active processes is provided through SEMAPHORES, QUEUES, SOCKETS, SHARED MEMORY and SIGNALS:

- semaphores are used to synchronize the access to the shared memory;
- main queues are used to send and receive messages from the Main Process (Q1);
- queues and/or sockets are used for the communication between terminals, engines and main processes and optionally between XCICS nucleus and XVSAM server when the latter is not linked-in (Q2, Qn).
3.1.2 Configuration reference

In order to run properly an application, XCICS/TS must be correctly configured.

An XCICS/TS application can be defined as a set of programs, datasets, maps and other resources, grouped together to run under the control of a single XCICS/TS main process.
3.1.2.1 The configuration file

The application profile, normally called xcics.conf and located under $HOME/etc, is a simple text file, grouping all information used by XCICS to start up and configure the main parameters of an application, and all its resources. Obviously it can be name differently than xcics.conf, and used by the –f option of xcicsctl.

I.e.

```bash
# xcicsctl -f /usr/local/mainappl.conf start
```

Will start up XCICS using a file named ‘mainappl.conf’ and located in ‘/usr/local’.

Parameters, resources and environment variables are specified using the following directives:

- set
- define
- bind
- add
- export
- link
- load
- include

Every directive is terminated using “;”. Directive parameters are comma (”,“) separated. If not explicitly required no quotas (“”) is required for values.

Lines started with “#” are treated as remarks.

**Include**

The "include" command let the user to include external files containing xcics configuration commands.

**Syntax**

```
include <filename>;
```

**Example**

```
include $HOME/etc/terminals.conf
include $HOME/etc/datasets.conf
```
3.1.2.2 Region parameters

The application parameters are used to control some XCICS behaviour concerning network, RDBMS, debug and application. Parameters are specified using the set directive in the configuration file:

**Set command**

This command sets the value of an XCICS/TS application parameter.

```
set <parm>=<value>;
```

3.1.2.2.1 General

**application_name**

This is the name of the application (APPLID).

No default is provided: this parameter must be specified.

**Syntax**

```
set application_name=<value>;
```

**local_sysid**

Specifies the local SYDID of the region, as returned by ASSIGN SYSID

**Syntax**

```
set local_sysid=<sysid>
```

**welcome_message**

Specifies the welcome message to be shown in the XCICS logon map.

No default is given and this parameter is optional. String MUST be quotes enclosed.

**Syntax**

```
set welcome_message="<text message>"
```

**jobname**

Specifies JOBNAME parameter, returned by INQUIRE JOBNAME.

No default is provided.

**Syntax**

```
set jobname=<job_name>
```
**default_transid**

Specifies default transaction code.
No default is provided.

Syntax
```
set default_transid=<TAC>
```

3.1.2.2 Region tuning

**server_pool_size**

Defines the number of serving processes for interactive sessions (it is a synonym of max_engines).

Syntax:
```
set server_pool_size=<value>
```

**cwa_size**

Specifies the size (in bytes) of application CWA (Common Work Area).

Syntax
```
set cwasize=<value>;
```

**table_increase_capacity**

Specifies the maximum number of runtime addable PCT, SNT, PPT, MPS entries. In other terms it defines the size of the space reserved to dynamically increase the number of items contained in the tables specified.

Syntax
```
set table_increase_capacity=<items>;
```

**max_sessions**

 Defines the maximum number of interactive sessions.
Default is unlimited.

Syntax
```
set max_sessions=<value>
```

**max_processes**

Defines the maximum number of concurrent processes in the application (both interactive and background).
Default is unlimited.
Syntax
set max_processes=<value>

**session_timeout**
Defines the connection timeout for inactive terminals.
Default is unlimited.

Syntax
set session_timeout=<value>

**vsam_timeout**
Defines the timeout for VSAM lock operation.
Default is 100 seconds.

Syntax
set vsam_timeout=<value>

**shut_timeout**
Defines the timeout before to proceed to a standard shutdown.
Default is 300 seconds.

Syntax
set shut_timeout=<value>

**kill_timeout**
Defines the timeout before to proceed to a forced shutdown after the normal shutdown has been started.
Default is 300 seconds.

Syntax
set kill_timeout=<value>

**memory_control**
Specifies if XCICS has to enforce memory allocation/deallocation check.

Syntax
set memory_control=<yes|no>

**local_getmain_table_size**
Defines the size, in bytes, of the internal memory table to reserve for task-only GETMAIN commands.
Syntax

set local_getmain_table_size=<value>

tmp_path

Defines the path for temporary files. If it is not set, XCICS uses the value of the environment $TMP as tmp_path. If $TMP is not set too, "tmp" is assumed.

Syntax

set tmp_path=<path>

temporary_storage_mode

Defines the storage type for temporary storage queue. By default TS are kept in memory.

Syntax

set temporary_storage_mode=(memory|flat|isam)

temporary_storage_path

Defines the path for ISAM and flat file based temporary storage. By default TS are kept in memory.

Syntax

set temporary_storage_path=<path>

ts_cache_size

Defines the value of the TS file descriptor cache size. Default is 128 entries.

Syntax

set ts_cache_size=<value>

document_tmp_path

 Defines the path for document temporary files.

Syntax

set document_tmp_path=<path>

reopen_std_fd

Enables XCICS to redirect STDOUT/STDERR to the XCICS trace log. Default is NO.
Syntax

```plaintext
set reopen_std_fd=<yes|no>
```

give_take_socket_table_size

Defines the size of the internal table used to simulate the TAKESOCKET/GIVESOCKET commands. Default is 32 entries.

Syntax

```plaintext
set give_take_socket_table_size=<value>
```

user_shared_memory_size

Defines the size, in Kbytes, of the shared memory to reserve for GETMAIN commands. Default is 256 Kbytes.

Syntax

```plaintext
set user_shared_memory_size=<value>
```

### 3.1.2.2.3 Networking

**network_port**

Specifies the TCP/IP port number on which XCICS must wait for incoming connections. Any free port can be selected. This variable is not needed, when a Microsoft Windows™ emulation for CXTERM is not used and whence XCLIENTSOCK variable is not set to YES.

The default value is 30585.

**WARNING:** To have more than one application running on the same machine, this port number must be different for each application.

Syntax

```plaintext
set network_port=<value>;
```

**bind_addr**

Specifies the IP address to bind. If not specified XCICS binds all available adapters.

Syntax

```plaintext
set bind_addr=<value>;
```

**default_inbound_lu**

Specifies the local LU alias used to listen for incoming SNA connections. By default it is the same as the application name.
**Syntax**

```plaintext
set default_inbound_lu=<lu alias>
```

**default_outbound_lu**

Specifies the local LU alias used outgoing SNA connections. By default it is the same as the application name.

**Syntax**

```plaintext
set default_outbound_lu=<lu alias>
```

**terminal_accept_latency**

Specifies the latency timeout for terminal connections.

**Syntax**

```plaintext
set terminal_accept_latency=<lu alias>
```

### 3.1.2.2.4 Programs and modules loading

**program_path**

Specifies the default path where to search for COBOL intermediate files.

XCICS will look for a `.gnt`, `.int`, `.acu` in this directory only in a case when no alternative path was specified for the appropriate program in the PPT table. Otherwise it will look for a `.int`, `.gnt`, `.acu` in the specified path.

The default value is "$HOME/tp/release".

**IMPORTANT:** This directory, as well as all alternate paths specified in PPT entries should contain only the results of a compilation without the `-a` option.

**N.B:** If using Microfocus Server Express 1.x both **program_path** and **program_debug_path** MUST refer to the debug directory (**program_debug_path**).

**Syntax**

```plaintext
set program_path=<absolute_path>
```

**program_debug_path**

Specifies the default path where to search for COBOL executable files compiled for debug, so that they can be animated.

This directory should contain the `.cbl`, `.acu` or `.int` and `.idy` files from your COBOL programs that are to be animated.

XCICS will look for the `.int`, `.acu` in this directory after start of the emulation while an animation session (described below) was specified. Otherwise - for a normal session - the executable code will be taken from the `program_path` directory or from an alternate path.

**N.B:** This directory should contain all results of compilations with the `-a` option.

The default value is "$HOME/tp/debug".

**Syntax**

```plaintext
set program_debug_path=<absolute_path>
```
**map_modules_path**

Points to the path containing BMS compiled (.mod) modules.

XCICS will look for a module in this directory only in a case when no alternate path was specified for the appropriate MAPSET in the MPS table. Otherwise the module will be searched in the specified path. BMS_MODULES is therefore a kind of a default value for a XSDF modules directory.

The default value is the value pointed by $XSDF_MODULES_PATH.

**Syntax:**

    set map_modules_path=<absolute_path>

**table_load_path**

Points to the path containing Assembler compiled (.lod) tables.

XCICS will look for a table in this directory when a LOAD command is issued.

**Syntax**

    set table_load_path=<absolute_path>

**program_asm_path**

Points to the path containing X370 compiled assembler modules.

**Syntax**

    set program_asm_path=<absolute_path>

**classpath**

Defines the CLASSPATH for the embedded Java Virtual Machine.

**Syntax**

    set classpath="<path>:<path>:...>

3.1.2.2.5 Files and databases

**recovery_log_path**

Specifies the directory used to write the XVSAM recovery logs. By default this path is set to $XVSAM

**Syntax**

    set recovery_log_path=<absolute_path>;

**use_rdbms**

Specifies if application is using an RDBMS. Allowed values are “yes” or “no”. No is the default.
Syntax
   set use_rdbms=(yes|no)

**hold_cursors_on_syncpoint**
If set to "yes", XCICS maintains cursors between different UOW.
Default is NO.

Syntax
   set hold_cursors_on_syncpoint=(yes|no)

**rdbms_date_format**
Specifies the date format returned by RDBMS
By default XCICS uses an european date format (dd.mm.yyyy)

Syntax
   set rdbms_date_format="<date format>"

**rdbms_timestamp_format**
Specifies the timestamp format returned by RDBMS. If set to "*" default database format is assumed.
By default XCICS uses an ISO timestamp format (yyyy-mm-dd hh.mm.sxxxx)

Syntax
   set rdbms_timestamp_format="<timestamp format>"

**force_disconnect**
Specifies if a COMMIT WORK RELEASE on RDBMS must be forced on XCICS syncpoints. Default is yes

Syntax
   set force_disconnect=(yes|no)

**use_dli**
Specifies if application is using H2R DLI gateway. Allowed values are "yes" or "no". No is the default.

Syntax
   set use_dli=(yes|no)

3.1.2.2.6 Compatibility

**bms_ascii_attributes**
Specifies whether XCICS BMS must consider the BMS attributes specified into User programs to be treated as EBCDIC or ASCII.
Allowed values are "yes" or "no". The default value is "no" and means that attributes are handled in EBCDIC format. When "yes" is specified, BMS attributes are translated into EBCDIC before to get handled.

**Syntax**

```
set bms_ascii_attributes=(yes|no)
```

**assume_getmain_shared**

If set to yes, XCICS assumes that all GETMAIN operation have to be allocated in shared memory, instead of having task-only allocation.

Default is no.

**Syntax**

```
set assume_getmain_shared=<yes|no>
```

**force_bms_comp_length**

If set to yes, XCICS assumes length field in BMS communication areas, have to be handled as big endian binary fields (COMP) instead of system dependent fields (COMP-5).

Default is NO.

**Syntax**

```
set force_bms_comp_length=<yes|no>
```

**default_date_format**

Defines the default format for the date/fulldate returned by FORMATTIME command.

Admittable values:

- EU (default)
- US
- ISO

**Syntax**

```
set default_date_format=<eu|us|iso>
```

**reset_trigger_on_error**

If set to yes, XCICS reset the TD trigger to 0 (zero) when an abend occurs on the triggered transaction. By default XCICS does not reset trigger on error.

**Syntax**

```
set reset_trigger_on_error=(yes|no)
```

**preserve_connection_on_dump**

If set to yes, XCICS maintains the connection with the terminal even if a signal or a COBOL error is detected.
Default is YES.

Syntax

```
set preserve_connection_on_error=(yes|no)
```

### 3.1.2.2.7 Security

**auth_mode**

Specifies the mode that XCICS will use to authenticate the users at SIGNON time.

XCICS supports 4 authentication modes:

<table>
<thead>
<tr>
<th>mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>Users are defined in the XCICS configuration file, with the &quot;define user&quot; directive and/or with the &quot;snt_loader&quot; exit_program. Clear text passwords are directly declared in the user definition statement with the &quot;password&quot; attribute. Users cannot change their passwords.</td>
</tr>
<tr>
<td>passwd</td>
<td>Similar to the basic mode, but passwords are encrypted stored in the file defined by the &quot;set passwd_file&quot; directive. Users can change their passwords.</td>
</tr>
<tr>
<td>ldap</td>
<td>Users are defined with &quot;define user&quot; directive and/or loaded from an LDAP server. The SIGNON is directly authenticated by the LDAP server. User may change their password according to the LDAP server settings. For further information, please refer to the &quot;Security management&quot; section.</td>
</tr>
<tr>
<td>user</td>
<td>Users are defined with &quot;define user&quot; in the configuration file and/or with the &quot;snt_loader&quot; exit_program. They are authenticated by a an user provided exit program of type &quot;user_signon&quot;. User may also change their password if an exit program of type &quot;user_chpasswd&quot; is defined. For further information, please refer to the &quot;Exit programs&quot; section and to the &quot;Customization Guide&quot;</td>
</tr>
</tbody>
</table>

Syntax

```
set auth_mode=(basic|passwd|ldap|user);
```

**passwd_file**

Specifies the path of the external password files. If defined the password are stored on the specified file and my be dynamically changed by users

Syntax

```
set passwd_file=<path>;
```

**single_signon**

Specifies if a single user may sign on from more than one terminal concurrently. By default multiple login are allow for a
single user.

**Syntax**

```
set single_signon=(yes|no);
```

**mixedcase_passwords**

Specifies if a mixed case passwords may be entered by users at sign-on time (CESN). The default value is "no".

**Syntax**

```
set mixedcase_passwords=(yes|no);
```

**default_user**

Specifies terminal default user. default_user and default_opid are mutually exclusive. default_user should be normally used.

No default is provided.

**Syntax**

```
set default_user=<username>
```

**utils_protection_level**

Specifies protection level for utility transactions, such CESN.

The default is 64.

**Syntax**

```
set utils_protection_level=<value>
```

**admin_protection_level**

Specifies protection level for administrative transactions.

The default is 64.

**Syntax**

```
set admin_protection_level=<value>
```

### 3.1.2.2.8 Logging and journaling

**logs_path**

Specifies the directory used to write the XCICS logs. By default this path is set to $HOME/logs.
Syntax

    set logs_path=<absolute_path>;

**core_trace**

Specifies the the core process (main) logging level. Admitted levels:

- none
- full

Syntax

    set core_trace=(none|full);

**task_trace**

Specifies the the user tasks (terminals) logging level. Admitted levels:

- none
- user
- full

Syntax

    set task_trace=(none|full);

**default_journal_path**

Specifies the deault path for journal log files.

If defined, XCICS invokes the specified routine to customize AID handling. The routine must be contained in a shared library to be loaded at XCICS startup using the "Load library" directive.

Syntax

    set default_journal_path=<path>

**terminal_control_journal**

If defined, the terminal activity is journaled on the specified journal. The journal name must be defined int JCT,

Syntax

    set terminal_control_journal=<filename>

**reopen_std_fd**

If set to "yes", XCICS reopens the stdout and stderr descriptor of the task, on the terminal log.

Syntax

    set reopen_std_fd=(yes|no)
**stat_system_tasks**

If set to "yes", XCICS includes in statistic measurement calculations system tasks too. By default this value is set to "no".

**Syntax**
```
set stat_system_tasks=(yes|no);
```

**allow_acu_thin**

If set to "yes", XCICS initialized the ACU COBOL runtime with the parameters to activate the COBOL debug on the ACU Thin client.

**Syntax**
```
set allow_acu_thin=(yes|no)
```

### 3.1.2.2.9 Printing and spooling

**spool_command**

Specifies the command used to execute the script generated from XCICS application when using SPOOL commands (SPOOLOPEN, SPOOLWRITE, SPOOLCLOSE).

By default the SPOOL_COMMAND is "csh".

**Syntax**
```
set spool_command=<command>
```

**spool_path**

Specifies the path to store both printing and scripts generated by XCICS application.

**Syntax**
```
set spool_path=<absolute_path>
```

**print_command**

Specifies the command to print files generated by XCICS application.

**Syntax**
```
set print_command=<command>
```
3.1.2.10 Customization

**default_terminal_height**
Sets the terminal default height in rows.

Syntax
```
set default_terminal_height=<value>
```

**default_terminal_width**
Sets the terminal default width in columns.

Syntax
```
set default_terminal_width=<value>
```

3.1.2.11 Debugging

**debug_acu_stub**
Specifies if ACU cobol stub loader should be debugged or not during debugger sessions.

Syntax
```
set debug_acu_stub=(yes|no)
```

**cedb_auto_mode**
If set to "yes" the CEDB transaction will automatically disconnect/reconnect to gain the reserved process.

Syntax
```
set cedb_auto_mode=(yes|no)
```

**timestamp_on_trace**
If set to "yes", lines in trace file, at level "full", will also contain a timestamp string.

Syntax
```
set timestamp_on_trace=(yes|no)
```
3.1.2.2.12 IMS/DC compatibility TK

**use_ims**
Specifies if application is using a the XIMS Compatibility toolkit

**Syntax**

```
set use_ims={yes|no}
```

**xims_format_path**
Specifies the path for XIMS formats

**Syntax**

```
set xims_format_path=<value>
```

**xims_spool_command**
Specifies the command to handle printout generated by program running under XIMS control.

**Syntax**

```
set xims_spool_command=<value>
```

**xims_spool_path**
Specifies the path where printout generated by program running under XIMS control are stored.

**Syntax**

```
set xims_spool_path=<path>
```

3.1.2.2.13 WebSphere/MQ

**use_mqseries**
Specifies if WebSphere/MQ connections must be allocated and kept under LUW for user transaction.

**Syntax**

```
set use_mqseries={yes|no}
```

**mqseries_xa_string**
Defines the XA string for WebSphere/MQ connections

The format of the XA string depends on the type of XFRAME configuration performed at setup for WebSphere/MQ (server or client).
For server configurations, only the qmname must be provided, for client configuration the whole XA string must be supplied. Please refer to the IBM WebSphere/MQ documentation.

Syntax

```bash
set mqseries_xa_string="<value>";
```

Samples

```
# server:
set mqseries_xa_string="qmname=venus.queue.manager";
# client:
set mqseries_xa_string="channel=CHANNEL.FC, trptype=tcp, connname=hpux02, qmname=venus.queue.manager"
```

`mqseries_queue_manager`

Defines the WebSphere/MQ default queue manager.

Syntax

```bash
set mqseries_queue_manager=<value>;
```

### 3.1.2.2.14 LDAP

`ldap_servers`

This parameter contains the name(s) (separated by space) of the LDAP servers from which XCICS/TS load the user list at startup or reconfiguration time.

Syntax

```bash
set ldap_servers="<value>";
```

`ldap_port`

The TCP port number where the LDAP service listens on LDAP servers. By default this value is the standard LDAP port number (389).

Syntax

```bash
set ldap_port=<value>;
```

`ldap_login_dn`

The full LDAP DN (Distinguish Name) used by XCICS/TS to login into the LDAP servers. (i.e. "uid=cicsuser,dc=example, dc=com");

Syntax

```bash
set ldap_login_dn="<value>";
```
**ldap_password**

The password of the DN used by XCICS/TS to login into the LDAP servers. It’s a plain text string.

**Syntax**

```
set ldap_password="<value>";
```

---

**ldap_base**

The LDAP search base used from where XCICS searches for valid users. It must be expressed as a standard LDAP search base (i.e. "dc=example,dc=com").

**Syntax**

```
set ldap_base="<value>";
```

---

**ldap_attribute_userid**

The name of the LDAP entry’s attribute where XCICS userid is stored. By default the ISO standard attribute "uid" is used.

**Syntax**

```
set ldap_attribute_userid="<value>";
```

---

**ldap_attribute_username**

The name of the LDAP entry’s attribute where XCICS username is stored. By default the ISO standard attribute "cn" is used.

**Syntax**

```
set ldap_attribute_username="<value>";
```

---

### 3.1.2.2.15 Deprecated settings

**max_engines**

Defines the number of serving processes for interactive sessions (obsolete).

**Syntax**

```
set max_engines=<value>
```

---

**run_logging_level**

Specifies the logging level for the XCICS child processes, and for their XVSAM accesses.
Syntax

```plaintext
set run_logging_level=<xcics level>:<xvsam level>
```

The xcics level indicates the debug level for the XCICS calls, while the xvsam level indicates debug level for XVSAM calls. The allowed level range goes from 00 to 15, where 00 means no debug, and 15 indicates the most detailed debugging level. Example:

```plaintext
set run_logging_level=15:15
```

The default value for this parameter is 00:00, no debug.

**main_logging_level**

Specifies the debug level for the XCICS main process, as `run_logging_level`.

If not specified the value of `run_logging_level` is used.

_N.B. The value of these two variables can be changed during runtime. You can also specify a particular level of logging for a specific terminal and/or for a single transaction (see XADM - ADMINISTRATION TOOLS below)._  

Syntax

```plaintext
set run_logging_level=<xcics level>:<xvsam level>
```

**max_commarea**

Defines the application maximum commarea size.

Syntax

```plaintext
set max_commarea=<value>
```

**use_cedf**

Enable CEDF printouts in logs.

Syntax

```plaintext
set use_cedf=<yes|no>
```

**slot_size**

Defines the size of the areas reserved for dynamic configuration. It specifies the maximum number of entries that may be added during an XCICS session for each reconfigurable table.

A default of 64 entries per table is defined.

Syntax

```plaintext
set slot_size=<value>
```
**default_opid**

Specifies terminal default OPID.

No default is provided.

**Syntax**

```plaintext
set default_opid=<OPID>
```

### 3.1.2.2.16 Web Interface

**server_ssl_certificate**

Full path of the server SSL pem certificate file used by TCP/IP services handling HTTP protocols (XCICS as HTTP server).

**Syntax**

```plaintext
set server_ssl_certificate=<path>;
```

**server_ssl_key**

Full path of the server SSL pem key file used by TCP/IP services handling HTTP protocols (XCICS as HTTP server).

**Syntax**

```plaintext
set server_ssl_key=<path>;
```

**client_ssl_certificate_path**

Path of the directory where the client SSL pem certificate files are stored. These files are used by user programs to open client HTTPS connections (XCICS as HTTP client).

**Syntax**

```plaintext
set client_ssl_certificate_path=<path>;
```

**client_ssl_key_path**

Path of the directory where the client SSL pem key files are stored. These files are used by user programs to open client HTTPS connections (XCICS as HTTP client).

**Syntax**

```plaintext
set client_ssl_key_path=<path>;
```

**cwi_timeout**

Timeout threshold, in milliseconds, after that WEB RECEIVE commands will be interrupted (returning TIMEDOUT). The default timeout value is 15000 milliseconds.
Syntax

    set cwi_timeout=<value>;}
3.1.2.3 Application resources

In the original mainframe CICS, resources are generally specified using a macro language, through the use of DFH macros or with the CEDA transaction. For XCICS/TS, the application resources are specified defining entries in the configuration file.

XCICS/TS resources contain information regarding:

- TCT - Terminal Control Table
- FCT - File Control Table
- DCT - Destination Control Table
- PCT - Program Control Table
- PPT - Program Processing Table
- JCT - Journal Control Table
- SNT - Signon Table
- MPS - Mapset Table
- CCT - Connection Control Table
- Memory buffer allocation

3.1.2.3.1 TCT - Terminal Control Table

This table defines names of the terminals allowed to connect to the application.

To define an entry in this table the directive `define terminal` is used. If a large set of terminals must be generated the directives `define terminalset` and `define browserset` can be used.

**define terminal**

Defines a terminal entry in the XCICS/TS terminal table (TCT).

```
define terminal name=<value>, device=<value>, [type=(standard|printer|virtual|web|session|remote)], netname=<value>, [address=x.x.x.x], [remote_name=<value>], [remote_system=<value>];
```

**name**

This is the logical name of the terminal as expected from the user program; this name is mandatory and it is passed back to the user program in the EIBTRMID field of the EIB control block; it must be unique within the application.

**type**

Different kind of terminals can be connected to XCICS:

- Network terminals (standard)
- Printers
- SNA sessions
- Remote terminals
- Virtual terminals
- WEB terminals

Network terminals come from the network, generated by a connection from X4J or XTND (both UNIX and Windows version).

The virtual terminals are used as resources by transactions generated by a START TRANSID command, or activated other background services (i.e. TCPIP services)

device

It is the device pool identifier of the terminal. This is the name that must be referred to when defining a connection in X4J or in the device attribute of the terminal element in the XTND configuration file.

More terminals can share the same device name: in such case XCICS assign the first terminal available with the requested device name.

If it is not defined, XCICS sets it equal to the netname. If also netname is not defined, the device is set equal to the terminal name.

physical

(deprecated) Alias name for device parameter.

IP address

It Is the network address of the terminal, specified in the format xxx.xxx.xxx.xxx (i.e. 10.1.1.50).

When set, XCICS reserves the terminal entry only for connection incoming from the IP address specified.

netname

For session type terminals, netname corresponds to the LU alias used in APPC communication.

For other terminal types it may be specified according to user preferences, otherwise XCICS automatically generates a netname for the terminal. If the terminal is served by an embedded tn3270 service (defined with "add tn3270server" command), the netname is also a constraint for the terminal connection.

remote_system

For remote terminals, specifies the name of the remote system from where the terminal can be used.

remote_name

For remote terminals, specifies the name the can be requested by remote systems. By default it is the same as the logical name.

**define terminalgroup**

Defines a group of size terminal with a fixed device name, a which logical name is composed by the un character prefix and an hexadecimal counter (Pxxx). The maximum group size is 4096.

**Syntax:**

```
define terminalgroup prefix=<value>,
device=<value>,
[type=standard,]
[size=<value>];
```

**define terminalset**

Defines a group of terminal with logical name prefixed by prefix, of the specified type and physical, in the network net in the specified range.
Syntax:

```
define terminalset prefix="<2 chars prefix>",
    physical=(WIN0-9|free),
    net=x.x.x,
    from=x,
    to=x;
```

**define browserset**

Defines a slot of webterm entry coming from a specified webserver.

**Syntax**

```
define browserset webserver=x.x.x.x size=<value>;
```

### 3.1.2.3.2 FCT - File Control Table

This table specifies the datasets to be used in the application.

To define an entry in this table the directive **define dataset** is used.

**define dataset**

Defines a file entry (dataset) in the XCICS/TS File Control Table (FCT).

```
define dataset name=<value>,
    cluster=<value>,
    catalog=<value>,
    [ sysid=<value> ],
    [ mode=( updt| rr.. ) ],
    [ logging=( on| off ) ],
    [ alternate ],
    [ immediate ],
    [ record_size=<value> ],
    [ key_length=<value> ],
    [ key_offset=<value> ],
    [ converter_function=<function> ],
    converter_lib=<library> ];
```

**name**

Name used in the application program when referring to such a file.

**cluster**

Name of the associated cluster entry in the XVSAM catalog.

**alternate**

This flag specifies if the dataset is a secondary index - Y - or a primary cluster - N -.

**catalog**

This is the name of the XVSAM catalog containing the specified cluster.
sysid

This parameter identifies the remote system that holds the file. It can be up to 4 characters long and it must correspond to one of the SYSIDs parameter defined in the Connection Control Table.

mode

The Open mode of the dataset or SYSID must be specified. The following modes are allowed:

- Update
- Exclusive update
- Load mode
- Exclusive protected retrieval
- Protected retrieval
- Protected retrieval reverse
- Exclusive retrieval
- Exclusive retrieval reverse
- Extended load

logging

If set to "on", the file is put under XVSAM transactional control, and all updates are tracked in the XVSAM recovery system. This means that transactional data integrity is preserved on this file and after image processing is also possible.

If set to "off", the file is directly updated without any transactional control. No after image is available. When logging is set to "off" the file is also set in "immediate" mode by default.

immediate

if specified, the file does not participate to the LUW (logical-unit-of-work), and it is therefore updated directly when the update command is issued, without waiting for the LUW completion.

record_length

This parameter specifies the record length of the file and must be provided only for remote files that must be dynamically converted.

key_length

This parameter specifies the key length of the file and must be provided only for remote files that must be dynamically converted.

key_offset

This parameter specifies the key offset of the file and must be provided only for remote files that must be dynamically converted.

converter_function

If specified, the specified routine is dynamically invoked by XCICS when the file is accessed on or by a mainframe system running IBM CICS/TS, to translate the commarea to and from EBCDIC. The routine must conform the XCICS dynamic conversion specs as documented in the XCICS Customization Guide.

converter_lib

specifies the name of the shared library containing the converter routine.
3.1.2.3.3 PPT - Program Processing Table

This table specifies the programs related to the application, their language, their processing type and location.

To define an entry in this table the directive `define program` is used.

**define program**

Defines a program entry in the XCICS/TS terminal table (TCT).

```
define program name=<value>,
    [language={cobol|c|pl1|java}>,</a>
    [sysid=<value>,]
    [path=<path>,]
    [sysid=sysdi,]
    [converter_function=<function>,
     converter_lib=<library>];
```

**name**

Name of the program.

**language**

Language of the program (COBOL, C language, PL/1 or X370 Assembler).

**path**

It indicates an alternative path to the program. Normally a program is searched in the directories specified as program_path by default (see configuration file). If a pathname is specified, the program is loaded from the specified directory. This path must be an absolute path.

For Java transaction classes the absolute class name must be specified in path.

For C and PL/1 programs, the shared library containing the module must be specified.

**sysid**

If specified, the program is remotely linked (DPL) on the CICS region identified by SYSID. The SYSID must be defined in CCT.

**converter_function**

If specified, the specified routine is dynamically invoked by XCICS when the server region is a mainframe system running IBM CICS/TS. to translate the commarea to and from EBCDIC. The routine must conform the XCICS dynamic conversion specs as documented in the XCICS Customization Guide.

**converter_lib**

specifies the name of the shared library containing the converter routine.

Example:

```
define program name=JAVAPGM1, language=java, path=com.hite.test.JTest01;
define program name=PLIPGM0, language=pl1, path=$HOME/lib/PLIPGM0.so;
```
3.1.2.3.4 PCT - Program Control Table

This table is used to specify transaction identifiers (TAC) and their relationship with programs.

To define an entry in this table the directive define transaction is used.

**define transaction**

Defines a transaction entry in the XCICS/TS terminal table (TCT).

```plaintext
define transaction name=<value>,
  program=<value>,
  twa=<value>,
  protection=<value>,
  backout=(on|off),
  [sysid=<value>],
  [remote_name=<value>],
  [converter_function=<function>,
   converter_lib=<library>];
```

**name**

This is the name associated with the transaction (TAC). The alias code may be used.

**program**

Name of user program assigned to the specified transaction code.

**twa**

Length in bytes of the transaction work area used by transactions.

**protection**

The Protection level, defines a security key for the transaction. Values between 01÷24 range are allowed.

**backout**

Specifies if the before image processing is required for the transaction. This value overrides what is specified in the FCTs related to data sets accessed during the transaction.

The default value, when nothing is entered, is yes, that means the file is to be backed up.

**sysid**

the name of the remote system where to execute the program

**remote_name**

the name of the transaction on the remote system

**converter_function**

If specified, the routine is dynamically invoked by XCICS when the transaction is started on or by a mainframe system running IBM CICS/TS, to translate the command area to and from EBCDIC. The routine must conform the XCICS dynamic conversion specs as documented in the XCICS Customization Guide.

**converter_lib**

specifies the name of the shared library containing the converter routine.
3.1.2.3.5 DCT - Destination Control Table

Each entry is used to describe a transient data (TD) destination afterwards referenced in the application. It may be either a file or a terminal.

To define an entry in this table the directive `define destination` is used.

**define destination**

Defines a destination entry in the XCICS/TS terminal table (TCT).

```sql
define destination name=<value>,
    type=<type>,
    format={fixed| variable},
    size=<value>,
    [ reusable=(yes|no), ]
    [ trigger=<value>, ]
    [ transid=<value>, ]
    [ device=(terminal| file), ]
    [ related=<value>, ]
    [ sysid=<value>, ]
    [ converter_function=<function>,
        converter lib=<library> ];
```

There are three possible types of destination:

- Extrapartitioned
- Intrapartitioned
- Indirect

**name**

It is the symbolic name of the element (max 8 chars). If destination type is Intrapartitioned the name of the resource can be up to 4 characters long.

**type**

This is the type of the entry and can be:

- External destination queue (extrapartitioned)
- Internal destination queue (intrapartitioned)
- Indirect destination queue (indirect)

**format**

Record format of the entry: fixed of variable.

**size**

Record length of the queue element.

**reusable**

This flag specifies if the resource can or cannot be reused.

**trigger**

This is the Trigger level counter. It is the value indicating if and when the related transaction is to be automatically started by the ATI (Asynchronous Transaction Interface).

**transid**
Identifier of the transaction to be started, when trigger level limit is reached.

**device**
Terminal or File.

**related**
Related resource name.

**converter_function**
If specified, the routine is dynamically invoked by XCICS when the transaction is started on or by a mainframe system running IBM CICS/TS, to translate the commarea to and from EBCDIC. The routine must conform the XCICS dynamic conversion specs as documented in the XCICS Customization Guide.

**converter_lib**
specifies the name of the shared library containing the converter routine.

---

### 3.1.2.3.6 MPS - Mapset Table

To define an entry in this table the directive **define mapset** is used.

**define mapset**
Defines a mapset entry in the XCICS/TS terminal table (TCT).

```
define mapset name=<value>, [path=<path>];
```

**name**
Name of the mapset.

**path**
Alternative path for the mapset. The mapset is searched in the default modules directory. If a path is given, the mapset is searched in the specified directory.

---

### 3.1.2.3.7 CCT - Communication Control Table

This table provides definitions used by the ISC (Inter System Communication) component to build up connections with remote systems, either UNIX, VSE or MVS.

To define an entry in this table the directive **define partner** is used.

**define connection**
Defines a connection entry in the XCICS/TS terminal table (TCT).

```
define connection  sysid=<value>, type=(tcp|sna|alias),
                  [ netname=<value>, ]
                  [ mode=<value>, ]
                  [ os=(ebcdic|ascii), ]
                  [ acquire,]
                  [ syncpoint=(simple|implied_forget|explicit_forget), ];
```
[symbolic_destination=<value>];

**sysid**

Is an identifier for intercommunication links and corresponds to the name specified in the SYSIDNT parameter of the DFHTCT TYPE=SYSTEM macro in the IBM CICS configuration.

**type**

It refers to the transport type for the connection:

- sna - SNA/APP C connection through APPC APIs
- tcp - APPC over TCP/IP

SNA connections are operated by means of a vendor dependant communication software such:

- Dataconnection SNAP/IX
- IBM Communication Server
- HP SNAplus 2

and can be established between XCICS/TS and another system speaking APPC (i.e IBM CICS regions, another XCICS/TS region or any other APPC software)

TCP connections do not require other third party software, but are supported only between XCICS/TS regions.

**symbolic_destination**

Only for alias connection, the CPI-C symbolic destination identifying the partner.

**netname**

Only for sna connections, Name of the LU identifying the partner: normally it should coincide with the name of the CICS region with which it wants to converse.

**mode**

Only for sna connections, Name of the Dialog Mode used in the conversation.

**acquire**

If specified, XCICS tries to acquire the connection automatically on startup.

**os**

Specifies OS encoding, for automated data conversion. Possible values:

- ascii
- ebcidic

By default, EBCDIC is assumed.

**syncpoint**

Specifies syncpoint transmission type. Possible values:

- simple
- implied_forget
- explicit_forget

By default, implied_forget is assumed.

For further information about syncpoint propagation, please refer to the "Distributed Transaction Processing" chapter in the "Intersystem Communication" section.
3.1.2.3.8 JCT - Journal Control Table

This table provides definitions for log files used by journaling sub-system.

To define an entry in this table the directive `define journal` is used.

**define journal**

Defines a journal entry in the XCICS/TS terminal table (TCT).

```
define journal name=<value>,
    number=<id>,
    [type=(native),]
    [filename=<value>;]
```

**name**

this is the 4 bytes symbolic identifier used in the EXEC CICS WRITE JOURNALNAME commands to reference the journal.

**number**

this is the numeric identifier for the journal (1-99) used in the EXEC CICS WRITE JOURNALNUM and EXEC CICS JOURNAL commands.

**type**

defines the type of the journal file. Actually only the "native" type is supported.

**filename**

spcifies the full path name to the journal log file. If not specified the files is created in the default journal path, with its name suffixed by ".log".

3.1.2.3.9 SNT - Sign-on Table

This table contains the information about the users attributes and access permission.

To define an user the directive `define user` is used.

**define user**

Defines a user entry in the XCICS/TS terminal table (TCT).

```
define user name=<value>,
    [userclass=<value>],
    [password=<value>],
    [opid=<value>],
    [auth="<value>"],
    [opkey=<hex-value>],
    [opclass=<hex-value>],
    [username="<value>"],
    [ldap_dn="<value>"],
    [sysuser|tduser];
```
name

It is the userid of an operator (max 8 chars).

userclass

If the user belongs to an userclass, it must specify the name of the userclass (up to 8 characters).

opid

This is the operator identifier, (max 3 chars). This parameter may be omitted if defined in the userclass.

password

Operator password (max 8 chars). This field is meaningful only when the region auth_mode is set to 'basic', otherwise it is ignored.

auth

Specifies all the protections level authorized for the user. The parameter must be quoted, and the levels must be separated by comma (i.e. "1,5,7"). Range values are admitted (i.e. "1,2,4-8,10-18,64"). Level value must be between 1 and 64. This parameter may be omitted if defined in the userclass.

opclass

It is a binary format number matrix representing the classes the operator belongs to (from 1 to 24). As for the operator keys, the matrix is structured in blocks of eight elements, which represent the classes sequence from 1 to 24. Positions containing 1 indicate classes the operator belongs to, while 0 represents classes unavailable for the operator. This parameter may be omitted if defined in the userclass.

opkey (deprecated)

It is a binary format number matrix (0/1) representing the operator keys (from 1 to 64). The matrix is structured in blocks of eight elements, which represent the keys sequence from 1 to 64. Positions containing 1 indicate available keys for the operator, while 0 represents not accessible keys.

username

specifies the 20 bytes character string identifying the user name.

ldap_dn

The LDAP DN (Distinguish Name) for the user. You should specify this only when auth_mode is set to "ldap" and you want to bind a certain user to a specific DN.

sysuser

If specified the user is used to start background transactions.

tduser

If specified the user is used to start TD.

Example

define user name=USERID00, opid=XXZ, opclass=FFFFF, auth="1,2,16-20,32-64", username="John Smith";
define user name=MASTER01, userclass=ADMINGRP, username="The Administrator";
3.1.2.3.10 TPL - Document Template Table

This table contains the information about the document templates. To define a document template the directive define doctemplate is used.

**define doctemplate**

Defines a document template entry in the XCICS/TS templates table (TPL).

```plaintext
define doctemplate name=<value>,
    [ type=(file|tsqueue) ],
    [ path=<value> ],
    [ tsqueue=<value> ];
```

**name**

It is the name of the document template (max. 48 chars).

**type**

defines the type of template.

<table>
<thead>
<tr>
<th>type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>the document template is stored in a flat file</td>
</tr>
<tr>
<td>tsqueue</td>
<td>the document template is stored in a temporary storage queue (this type is not currently supported).</td>
</tr>
</tbody>
</table>

**path**

Absolute path of the file containing the template.

**tsqueue**

Name of the Temporary storage containing the template

**Example**

```plaintext
define doctemplate name=MAIN, type=file, path=$HOME/doctemplates/main.html;
define doctemplate name=HOMEPAGE, type=file, path=$HOME/doctemplates/home_page.html;
```

3.1.2.3.11 URM - Urimap Table

This table contains the information about the URIMAPs. To define an URIMAP the directive define urimap is used.

**define urimap**

Defines an URIMAP entry in the XCICS/TS URIMAP table (URM). URIMAP options are strictly dependant on its usage.

```plaintext
define urimap name=<value>,
    [ enabled=(yes|no) ],
    [ usage=(server|client|pipeline) ],
    [ scheme=(http|https) ],
    [ host="<string>"],
    [ path="<string>"],
```
name
It is the name of the URIMAP.

enabled
Defines whether the URIMAP is enabled or not.

usage
defines the usage of the URIMAP.

| server | the URIMAP is used by XCICS acting as an HTTP server, to map the incoming requests |
| client | the URIMAP is used by XCICS acting as an HTTP client, to route the outgoing requests |
| pipeline | not yet supported |

Options for server usage

host
The virtual host name for which the URIMAP applies. If the value is the wildcard "*", the URIMAP applies to all the virtual hosts.

path
The resource path for which the URIMAP applies. The wildcard "*" may be used to complete the path.

scheme
The URIMAP matches only request on the scheme specified. If it is not specified, the URIMAP matches both HTTP and HTTPS schemes.

tcpipservice
Restricts the URIMAP only to the requests caught by the specified tcpipservice. If omitted or if the value is the wildcard "*", the URIMAP applies to all the tcpip services.

type
Defines how the request is managed.
The request will be handled by user application programs, which will dynamically generate the response

The request is mapped to a static resource (file) and will be handled by XCICS, which replies with the content of the file.

The response will redirect the client to the defined location

Defines whether the analyzer will be activated or not. This option is valid for application type URIMAPs only.

If specified, defines the name of the converter program to be invoked to encode/decode the request/response. This option is valid for application type URIMAPs only.

If specified, overrides the name of the default alias transaction. This option is valid for application type URIMAPs only.

The name of the application program to handle the request. This option is valid for application type URIMAPs only.

If specified, the USERID used to run the alias transaction. This option is valid for application type URIMAPs only.

The absolute path of the file bind the request. The "*" wild-card may be used to complete the path name. This option is valid for static type URIMAPs only.

The media type of the file bind the request. If the "*" wild-card is specified, XCICS tries to identify the file media_type according to its extension. This option is valid for static type URIMAPs only.

Defines the character set of the file. This option is valid for static type URIMAPs only.

Specifies if the redirect is of temporary or permanent type. This option is valid for redirect type URIMAPs only.

Specifies URL where to redirect the client. This option is valid for redirect type URIMAPs only.

The host name to address the request.

The port number where to address the request on the remote host. If omitted the value is set to the default according to scheme required.

The path
The resource path of the request.

scheme

The scheme used to address the request.

certificate

The name of the client certificate to use for HTTPS requests.

certificate

The name of the client certificate to use for HTTPS requests.

ciphers

The label of the ciphers suite to use for HTTPS requests.

Example

```plaintext
# server static urimaps
#
define urimap name=URL1, usage=server, type=static, host="\*", path="/home/\*",
    file="$HOME/request/web/\*";
define urimap name=URL2, usage=server, type=static, scheme=https, host="\*",
    path="/static/";
define urimap name=URL3, usage=server, type=static, tcpipservice=", scheme=http,
    host="\*",
    path="/documents/\*",
    file="/home/galletti/prova_sito_pwi/\*",
    media_type="text/html";
#
# server application urimaps
#
define urimap name=URJA,
    usage=server,
    type=application,
    analyzer=yes,
    host="\*",
    program="CMDURL",
    path="/prova/are\*";
define urimap name=URJB,
    usage=server,
    type=application,
    scheme=http,
    analyzer=no,
    tcpipservice=HTTP,
    host="\*",
    transaction=CMDA,
    program="CMDURL",
    path="/prova/are\*";
#
# server redirect urimaps
#
define urimap name=RED1,
    usage=server,
    type=redirect,
    scheme=http,
    host="\*",
    path="/gif",
    url="http://www.google.com";
#
# client urimaps
```
3.1.2.3.12 Memory buffers allocation

Memory buffers are the areas where XCICS memory operations are performed (TS, GETMAIN etc.). Five buffer sets are required by XCICS to work: they can be defined using the `add bufferset` directive.

**define bufferset**

Defines a memory buffer entry in the XCICS/TS terminal table (TCT).

```
add bufferset buffers=<value>, size=<value>
```

This directive adds a set of buffers of the specified size (512 bytes blocks).

- **buffers**
  
  number of buffers in the bufferset.

- **size**
  
  buffer dimension in 512-bytes unit.

**WARNING:** a maximum of 5 buffers can be added.
3.1.2.4 RDBMS interface

In order to interface an RDBMS (Oracle, DB/2, SyBase, etc), XCICS requires some additional information such asDBC and binding.

**DBC (DataBase Connection)**

A DBC entry is an object identifying the single connection RDBMS/USER, and can be defined using define dbc.

**Syntax:**

```plaintext
define dbc
  name=<value>,
  database=<value>,
  user=<value>,
  password=<value>,
  [connect_type=(1 2)],
  [sql_rules=(db2 std)],
  [syncpoint_type=(onephase twophase none)],
  [disconnect_type=(expl cond auto)],
  [deferred_prepare=(no yes all)];
```

- **name**
  - name is the dbc identifier (used in bind)

- **database**
  - the database identifier

- **user**
  - the database user

- **password**
  - the user password.

**Native UDB only parameters**

The following parameters apply only to "native" type connections to UDB database. The native UDB connection type is selected at the moment of XFRAME installation and runtime configurations.

These options configure the UDB client API layer (SET CLIENT API), and they have no effect on other connections types (i.e. Oracle or UDB/XA).

For further information on UDB client API settings please refer to the IBM UDB documentation.

**connection_type**

Specifies how to set up the CONNECTION type. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type 1 CONNECTs enforce the single database per unit of work semantics of older releases, also known as the rules for remote unit of work (RUOW).</td>
</tr>
<tr>
<td>2</td>
<td>Type 2 CONNECTs support the multiple databases per unit of work semantics of DUOW.</td>
</tr>
</tbody>
</table>
Default value is "1".

**sql_rules**

Specifies how to set up the SQLRULES. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2</td>
<td>Enable the SQL CONNECT statement to switch the current connection to an established (dormant) connection.</td>
</tr>
<tr>
<td>std</td>
<td>Permit only the establishment of a new connection through the SQL CONNECT statement. The SQL SET CONNECTION statement must be used to switch the current connection to a dormant connection.</td>
</tr>
</tbody>
</table>

Default value is "db2".

**syncpoint_type**

Specifies how to set up the coordination among multiple database connections during commits or rollbacks. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onephase</td>
<td>Uses one-phase commits to commit the work done by each database in multiple database transactions. Enforces single updater, multiple read behavior.</td>
</tr>
<tr>
<td>twophase</td>
<td>Requires a Transaction Manager (TM) to coordinate two-phase commits among databases that support this protocol.</td>
</tr>
<tr>
<td>none</td>
<td>Uses one-phase commits to commit work done, but does not enforce single updater, multiple read behavior.</td>
</tr>
</tbody>
</table>

Default value is "twophase".

**disconnect_type**

Specifies how to set up the scope of disconnection to databases at commit. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expl</td>
<td>Removes those connections that have been explicitly marked for release by the SQL RELEASE statement at commit.</td>
</tr>
<tr>
<td>cond</td>
<td>Breaks those connections that have no open WITH HOLD cursors at commit, and those that have been marked for release by the SQL RELEASE statement.</td>
</tr>
<tr>
<td>auto</td>
<td>Breaks all connections at commit.</td>
</tr>
</tbody>
</table>

Default value is "expl".
**deferred_prepare**

Specifies when to execute the PREPARE statement. The valid values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>no</td>
<td>The PREPARE statement will be executed at the time it is issued.</td>
</tr>
<tr>
<td>yes</td>
<td>Execution of the PREPARE statement will be deferred until the corresponding OPEN, DESCRIBE, or EXECUTE statement is issued. The PREPARE statement will not be deferred if it uses the INTO clause, which requires an SQLDA to be returned immediately. However, if the PREPARE INTO statement is issued for a cursor that does not use any parameter markers, the processing will be optimized by pre-OPENing the cursor when the PREPARE is executed.</td>
</tr>
<tr>
<td>all</td>
<td>Same as YES, except that a PREPARE INTO statement which contains parameter markers is deferred. If a PREPARE INTO statement does not contain parameter markers, pre-OPENing of the cursor will still be performed. If the PREPARE statement uses the INTO clause to return an SQLDA, the application must not reference the content of this SQLDA until the OPEN, DESCRIBE, or EXECUTE statement is issued and returned.</td>
</tr>
</tbody>
</table>

Default value is "no".

**Binding**

Binding is the relationship between XCICS users, transactions and dbc. Using bind it is possible to associate a specific dbc to some transaction or some users.

A default binding must be provided. Binding is specified using `bind`.

**Syntax**

```
bind dbc=<name> default;
```

or

```
bind dbc=<name>, user=(<name>|*), transaction=(<name>|*);
```

When specifying a default it means that all users and transaction are using that dbc, if not furthermore specified.

Specific bind can be done using the second format of bind directive.

**user**

name of XCICS user using that dbc. If astersik (*) is specified all user can use the dbc.

**transaction**

TAC of the transaction using that dbc. If astersik (*) is specified all transaction can use the dbc.

For Example:

```bash
# defining the production DB..
# both for standard and administration functions
define dbc name=PROD, database=ORAPROD, user=CED01, password=CES01;
```
define dbc name=ADMPROD, database=ORAPROD, user=SYSTEM, password=MANAGER;
# ...and the Test DB
define dbc name=TEST, database=ORATEST, user=CEDO2, password=XXX
# application default dbc is PROD
bind dbc=PROD default;
# user DEVELO must use TEST DB for ALL transactions
bind dbc=TEST, user=DEVEL, transaction=*;
# transaction DBA1 uses on TEST DB
bind dbc=TEST, user=*, transaction=DBA1;
# user ROOR works on PROD with DBA1
bind dbc=PROD, user=ROOT, transaction=DBA1
3.1.2.5 Alternate character sets

**Overriding Codepage**

Alternate codepages may be loaded by XCICS using the `load` directive.

Specified codepage must be an absolute path name to a table file.

This table file must be a 512 bytes length sequential file, containing on the first 256 bytes the ASCII to EBCDIC translation values and on the second 256 bytes the EBCDIC to ASCII values. I.e. the file at offset 0x20 (ASCII blank) contains 0x40 (EBCDIC blank), while at offset 0xFF+0x40 contains 0x20.

Codepages may be loaded for whole CICS (load codepage) or only for BMS routines (load bms_codepage).

**Syntax**

```
load codepage=<table>
load bms_codepage=<table>
```

**Example**

```
load codepage=$XFRAMEHOME/etc/IBM930.tbl;
```

Sample XFRAME codepage tables are deployed with XFRAME and may be found in $XFRAMEHOME/etc with ".tbl" extention.

The human readable sources for the codepage tables deployed with XFRAME may be found in $XFRAMEHOME/tools with .cpg extention.

If the need for a new codepage arise, user may create its own codepage source starting from one of those provided, editing it and the converting it to an XFRAME codepage table using the script $XFRAMEHOME/tools/cpg2tbl.ksh.

**Specifying DBCS conversion table**

If application requires a DBCS conversion table, it may be loaded using `load dbcs_table` directive

Specified DBCS table must be an absolute path name to a table file.

**Syntax**

```
load dbcs_table=<table>
```

**Example**

```
load dbcs_table=$XFRAMEHOME/etc/IBM-Kanji.tbl;
```

Sample XFRAME DBCS tables are deployed with XFRAME and may be found in $XFRAMEHOME/etc with ".tbl" extention.

The human readable sources for the codepage tables deployed with XFRAME may be found in $XFRAMEHOME/tools with .dbcs extention.

If the need for a new DBCS conversion table arise, user may create its own codepage source starting from one of those provided, editing it and the converting it to an XFRAME DBCS table using the script $XFRAMEHOME/tools/dbcs2tbl.ksh.
3.1.2.6 Environment Variables

Application may need some environments set. Obviously environment variables set by the starting shell can be read, but it is possible to force some environment to be defined and exported at Xcics startup.

**export**

Using export directive it possible to define and export environments.

**Syntax:**

```
export VARIABLE=value
```
3.1.2.7 User classes

Whenever the region administrator wants to define more users sharing the same attributes and access rights, it is possible to define an userclass.

An userclass is a collection of user attributes: all the users belonging to the userclass inherit the attributes defined in it. Those attributes defined in the user's definition too (define user) override the userclass's ones.

To define an userclass, use the directive define userclass.

**Syntax**

```
define userclass name=<value>,
    auth="<value>",
    [opid=<value>],
    [opclass=<hex-value>],
    [ldap_filter="<value>"];
```

**name**

This is the class identifier, eight alphanumeric chars to be referenced in the "define user".

**opid**

This is the operator identifier, three alphanumeric chars.

**auth**

Specifies all the protections level authorized for the user. The parameter must be quoted, and the levels must be separated by comma (i.e. "1,5,7"). Range values are admitted (i.e "1,2,4-8,10-18,64"). Level value must be between 1 and 64.

**opclass**

It is a binary format number matrix representing the classes which the operator belongs to (from 1 to 24). The matrix is structured in blocks of eight elements, which represent the classes sequence from 1 to 24. Positions containing 1 indicate classes the operator belongs to, while 0 represents classes unavailable for the operator.

**ldap_filter**

When XCICS/TS auth_mode is set to "ldap", all the LDAP entries that satisfy the filter are defined as XCICS users and bind to the userclass. The filter must be coded according to LDAP filters rules (i.e. "(&(objectClass=inetOrgPerson) (ou=Product Development))").

**A practical example**

```bash
# using external password repository
# set passwd_file=$HOME/etc/xpasswd
set default_user=GUEST;
# users classes
# define userclass name=CLSGUEST, opid=GST, auth="1", opclass=000000;
define userclass name=CLSCOPER, opid=USR, auth="1-16,32,33,40-50", opclass=00FFFF;
define userclass name=CLADMIN, opid=ALL, auth="1-64", opclass=FFFFFF;
# users
# guest user definition
# define user name=GUEST, userclass=CLSGUEST, username="Guest user";
# # standard users definition
```
define user name=LSKYWALK, userclass=CLSOPER, username="Luke Skywalker";
define user name=JKIRK, userclass=CLSOPER, username="Cpt. James Kirk";
define user name=DECKARD, userclass=CLSOPER, username="Rick Deckard", opid=USEG;
#
# super-user
#
define user name=FABRIZIO, userclass=CLSADMIN, username="Administrator user";
3.1.2.8 Shared library loading

Shared libraries may be dynamically loaded by XCICS using the `load` directive.

Specified library may be an absolute path name or a simple library name. If no path is specified library is searched in the standard library loading path (LD_LIBRARY_PATH, SHLIB_PATH or LIBPATH depending on O.S.).

Also extension must be specified (.so or .sl depending on O.S.)

**Syntax**

```
load library=<libname>
```

**Example**

```
load library=libprograms.so;
load library=/usr/lib/libloadable.so;
```
3.1.2.9 Exit programs

Some aspects of the configuration as well as some run-time behaviour of XCICS/TS may be influenced and/or driven by user exit programs.

An exit program is a C language routine, written by the user and declared at configuration time.

XCICS/TS supports the following type of exit programs

<table>
<thead>
<tr>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odc_ts</td>
<td>the routine is invoked by XCICS/TS to convert the temporary storage records, whenever a TS queue is accessed on or by remote EBCDIC systems. This routine must conform to the rules defined by the XCICS/TS Online Data Conversion System (ODCS)</td>
</tr>
<tr>
<td>snt_loader</td>
<td>the routine is invoke by XCICS/TS at configuration time (startup and reconfigure) to feed the SNT tables. User may provide this routine whenever he need to define users programmatically (i.e. users are read from a DB and automatically defined)</td>
</tr>
<tr>
<td>user_signon</td>
<td>the routine is invoked by XCICS/TS when the operator tries to SIGNON (CESN transaction or EXEC CICS SIGNON command). The routine receives the USERID and the PASSWORD entered by the operator and decide if the operator is authorized to sign on.</td>
</tr>
<tr>
<td>user_signoff</td>
<td>the routine is invoked by XCICS/TS when user signs off (CESF or EXEC CICS SIGNOFF).</td>
</tr>
<tr>
<td>user_chpasswd</td>
<td>the routine is invoked by XCICS/TS when the operator tries to change his password (CESN transaction or EXEC CICS SIGNON NEWPASSWORD command). The routine receive the USERID and the PASSWORDs entered by the operator.</td>
</tr>
<tr>
<td>dfhaid</td>
<td>the routine is invoked by XCICS/TS when the operator enters an AID, and has the capability to alter the transaction startup, i.e changing the behaviour of the AID or the RETURN TRANSID.</td>
</tr>
</tbody>
</table>

For further information about exit programs creation and usage please refer to the "Customization Guide".

Exit programs are declared with the **define exit_program** directive

**define exit_program**

**Syntax**

```c
define exit_program
    type=(odcs_ts|snt_loader|user_signon|user_signoff|user_chpasswd|dfhaid),
    function=<value>,
    [library=<value>];
```

**type**
This is the type of exit program being defined.

**function**

The name of the routine.

**library**

The name shared library containing the routine. If the library is not in the linker loading path (LD_LIBRARY_PATH, SHLIB_PATH, LIBPATH) the absolute path must be provided. If the library parameter is omitted, the function is searched in the libraries previously loaded with the "load library" directive.

**Example**

```plaintext
define exit_program type=user_signon, function=dfhep_signon, library=libuserid.so;
define exit_program type=user_signoff, function=dfhep_signoff, library=$HOME/lib/libsignoff.so;
```
3.1.2.1 Java Virtual Machine options

Options may be passed to the embedded Java Virtual Machine.

To define options to the JVM the `add javaopt` directive must be used:

**Syntax**

```
add javaopt value="<value>"
```

**Example**

```
add javaopt="-verbose";
add javaopt="-Djava.compiler=NONE";
```
3.1.2.1 SNA and TCP/IP APPC sessions

In order to allow remote regions to connect the local XCICS region to perform inter system communication (i.e. starting transactions for DTP or to request services such as DPL or RFA), some server sessions must be defined in the XCICS configuration file.

Depending on the transport layer, two different directives are used.

**SNA sessions**

SNA sessions are defined by means of the `define sessions` directive.

Each session is background terminal defined in TCT, which name is composed by the one-byte prefix followed by a counter.

```plaintext
define session
```

**Syntax**

```
define sessions size=<value>,
[luname=<value>],
[prefix=<value>],;
```

- **size**
  - This is the number of parallel session to be activated.

- **luname**
  - Specifies the LU alias for the incoming connections. If not specified, XCICS assigns the name specified by the general setting `default_inbound_lu`, if declared, or the application name itself.

- **prefix**
  - If specified, overrides the default character ("-") for terminal name generation.

**TCP/IP sessions**

TCP/IP sessions are activated by a TCP server and are defined by means of the `add tcpserver` directive.

The TCP server listens for incoming TCP/IP connections and activates a background task to serve the request.

```plaintext
define session
```

**Syntax**

```
add tcpserver name=<value> , port=<value> [,terminal=<value>],
isct;
```

- **name**
  - Name of the TCP server.

- **port**
  - Number of the TCP port, where the server is waiting for connections.

- **terminal**
  - (Optional) Name of the virtual terminal where the task is to be started.
isc

It is mandatory.
3.1.2.1 TN3270 services

XCICS accepts terminal connection only over its native protocol, from X4J terminals. In order to access to tn3270 terminals (i.e. IBM Personal Communication, Tn3270+, eXtra, etc), XFRAME provides an additional tool, named XTND, which is a TN3270 / XCICS protocol converter.

XTND can be run as an external service or directly embedded by XCICS. In this case XCICS takes care to define the XTND configuration and to start/stop the service whenever required.

This section covers the configuration aspects of tn3270 services embedded in the XCICS region.

By means of the command `add tn3270server` one or more tn3270 services may be added and started to an XCICS region. All the services defined will be activated at XCICS start up, and they will provide tn3270 access to all the terminals defined as "standard" type.

The connection of terminals defined in TCT with the "address" parameter will be allowed only for tn3270 connection coming from the specified IP address.

If the TCT entry contains the "netname" clause, the connection will be allowed only for those tn3270 terminal requiring an LUNAME equal to the one specified in the "netname" clause.

When XCICS region is alive, TN3270 services may be controlled using `xcicsadm`.

For each of these tn3270 services some parameters must be supplied.

For additional information about XTND, please refer to the XTND Tn3270 Server guide.

### Syntax

```
add tn3270service [name=<value>],
  [port=<value>],
  [address=<value>],
  [admin_port=<value>],
  [timeout=<value>],
  [ssl, keystore=<value>, keystorepassword=<value>],
  [enabled|disabled];
```

### Name

This is the name of the service. If it is not specified it is automatically set by XCICS.

### Port

This is the port number where the service will be listening for incoming tn3270 connections and where tn3270 emulators should connect to. If it is not specified it is XCICS automatically assigns it. In this case the number assigned can be read on the console log or using xcicsadm utility.

### Address

This is the IP address bind by the tn3270 service. It must be specified when you want the service to listen on a specific IP. If it is not set the service is bind on the default IP.

### Admin Port

This is the port number where the service will be listening for admin connections. If it is not specified it is XCICS automatically assigns it. Normally there is no need to specify this parameter.

### Timeout

This is the time, in minutes, after that inactive terminals are automatically disconnected. If not specified or set to zero, no timeout is managed.

### SSL

If set, the tn3270 service will accept only SSL communication. When set, keystore and keystorepassword must also be configured.
keystore

Full path of the Certificate store file.

keystorepassword

Password to access the key store file

For further information about tn3270 over SSL and its configuration, please refer to the XTND documentation, in the "Using SSL" section.

enabled

If specified the service is automatically activated at XCICS start up time. This is the default behaviour.

disabled

If specified the service is not automatically activated at XCICS start up time.
3.1.2.1 TCP/IP services

One or more TCP/IP services may be defined, with the `add tcpipservice` directive.

When XCICS region is alive, TCP/IP services may be controlled using `xcicsadm`.

**Syntax**

```plaintext
add tcpipservice name="<value>",
    port=<value>,
    [ terminal=<value> ],
    [ server_pool=(yes|no) ],
    [ protocol=(direct|classic|webservices|eci|isc|http|user) ],
    [ urm=<value> ],
    [ ssl=(yes|no) ],
    [ close_connection=(yes|no) ],
    [ close_timeout=<value> ],
    [ auth=(none|basic|certificate|automatic) ],
    [ enabled|disabled ]
```

**name**

Name of the TCP service.

**port**

Number of the TCP port, where the service waiting for connections.

**terminal**

(Optional) Name of the virtual terminal where the task is to be started.

**server_pool**

If set to yes, the service forwards the transaction directly on the engines pool, instead of forking as every normal background process;

**protocol**

defines the protocol handled by the tcpip service. The protocol may be one of the following:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direct</td>
<td>the service handles the direct ezalistener protocol</td>
</tr>
<tr>
<td>classic</td>
<td>the service handles classic ezalistener protocol</td>
</tr>
<tr>
<td>webservices</td>
<td>the service handles the XCICS/S Web Service interface protocol</td>
</tr>
<tr>
<td>isc</td>
<td>the service handles the XCICS/TS Intersystem communication with APPC over TCP/IP protocol</td>
</tr>
<tr>
<td>eci</td>
<td>the service handles XECI (XCICS External Call Interface) requests</td>
</tr>
<tr>
<td>http</td>
<td>the service handles HTTP requests for the XCICS Web Interface</td>
</tr>
<tr>
<td>user</td>
<td>the service handles user protocols for non-HTTP requests</td>
</tr>
</tbody>
</table>

**enabled**

If specified the service is automatically activated at XCICS start up time. This is the default behaviour.
disabled
If specified the service is not automatically activated at XCICS start up time.

urmm
For HTTP and USER protocol only. Defines the name of the analyzer program to be started to handle HTTP and non-HTTP requests.

ssl
For HTTP and USER protocol only. If set to yes, the communication with the service will be encrypted over SSL (Secure Socket Layer)

close_connection
For HTTP and USER protocol only. Controls the usage of permanent connection. By default XCICS uses permanent connections.

close_timeout
For HTTP and USER protocol only. Defines the close timeout, in seconds, for permanent connections.

auth
For HTTP and USER protocol only. Defines the authorization method for the service.

<table>
<thead>
<tr>
<th>none</th>
<th>no authorization is required to the client</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>requires BASIC HTTP authentication to the client</td>
</tr>
<tr>
<td>certificate</td>
<td>requires the client to provide a certificate, and use it to authorize.</td>
</tr>
<tr>
<td>automatic</td>
<td>if the client sends a certificate, the service will use it to authenticate the client, otherwise BASIC authentication will be requested.</td>
</tr>
</tbody>
</table>

For further information about the EZA-like and SOAP TCP services modes, please refer to the TCP Services. For further information about XCICS Web Interface, HTTP and user protocols, please refer to the XCICS Internet Guide.

If no mode is specified, the BASIC mode is assumed by default.

Example:

```
add tcpipservice name=DEFAULT, port=8023;add tcpipservice name=WEBSVC, port=8090, terminal=WEB, protocol=web_services;
add tcpipservice name=HTTPD, port=80, protocol=HTTP, close_timeout=25, urm=ANALYZ01, auth=none;
add tcpipservice name=HTTPD, port=443, protocol=HTTP, ssl=yes, server_pool=no, auth=automatic;
```
3.1.2.1 MQ Trigger Monitor services

XCICS may monitor WebSphere queues, to replicate the behaviour of the IBM MQ/SERIES CICS Adapter.

To do that one or more trigger monitors must be defined.

By means of the command `add mqtrgmon` one or more trigger monitoring services may be added and started to an XCICS region.

All the services defined will be activated at XCICS start up, and they will provide a CKTI-like monitoring and task activation services.

When XCICS region is alive, Trigger Monitoring services may be controlled using `xcicsadm`.

For each of these trigger monitors some parameters must be supplied.

Syntax

```
add mqtrgmon [name=<value>],
[queue=<value>],
[{enabled|disabled}];
```

**name**

This is the name of the service. If it is not specified it is automatically set by XCICS.

**queue**

This is the name of the initiation queue monitored by the services. If it is not set, XCICS automatically assign the name `"<APPLID>.INITIATION.QUEUE"`, where `<APPLID>` is the XCICS region application identifier (application_name).

**enabled**

If specified the service is automatically activated at XCICS start up time. This is the default behaviour.

**disabled**

If specified the service is not automatically activated at XCICS start up time.
3.1.2.1 Transaction keybinding

Transaction keybinding command enables the automatic installation of default user exit routines that controls the AID handling. This program starts the specified transaction code when a certain key pressed on terminal.

To define a key bind use the link directive.

An alternative to this directive may be the definition and installation of an user written exit program (refer to the XCICS Customization Guide).

**Syntax**

```
link <AID>=<transid>;
```

The following AIDs are supported:

```
DFHNULL
DFHENTR
DFHCLEAR
DFHPEN
DFHPID
DFHPA1
DFHPA2
DFHPA3
DFHPF1
DFHPF2
DFHPF3
DFHPF4
DFHPF5
DFHPF6
DFHPF7
DFHPF8
DFHPF9
DFHPF10
DFHPF11
DFHPF12
DFHPF13
DFHPF14
DFHPF15
DFHPF16
DFHPF17
DFHPF18
DFHPF19
DFHPF20
DFHPF21
DFHPF22
DFHPF23
DFHPF24
```

**Example:**

```
link DFHPF11=SAVE
link DFHPF12=BACK
```
3.1.3 Region control

Each XCICS TS region may be controlled through different tools. Some of these tools simply take care to start, stop the region or inquire its status; other tools provide a wider set of features to control every aspect of the running instance.
3.1.3.1 Command line utilities
XCICS/TS may be controlled from the system shell or from scripts using its command line utilities.

3.1.3.1.1 xcicsctl
An XCICS region can be easily controlled using the xcicsctl shell script, which provides an user friendly interface for the most frequently used administrative tasks.

Syntax
```
xcicsctl [-cyf<configuration file>] <command>
```

Options
- `-c` console mode
- `-v` verbose mode
- `-i` immediate mode for shut
- `-h` hard mode for report (just system report)
- `-s` soft query for status (uses xcicsadm to determine application status)
- `-f<file>` uses <file> configuration
- `-j<file>` execute the start-up job
- `-9` forces kill to invoke signal 9 (SIGKILL)
- `-t` skips transient data erasure on cold start

Return Code

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>1</td>
<td>error on options</td>
</tr>
<tr>
<td>2</td>
<td>configuration error</td>
</tr>
<tr>
<td>8</td>
<td>network busy</td>
</tr>
<tr>
<td>9</td>
<td>application already active</td>
</tr>
</tbody>
</table>

When the application starts as a background process, no information log is displayed on the console terminal and the operator may close the shell used to enter the xcicsctl command. In this case, the information log will be written on the `lastlog` file in the logs directory (LOG_PATH if defined in the XCICS profile or $HOME/logs).

WARNING: if XCICS is started up as foreground process, every interruption sent to terminal will kill the application

The following command can be given to xcicsctl:
## Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>cold start up application</td>
</tr>
<tr>
<td>warmstart</td>
<td>warm start up application</td>
</tr>
<tr>
<td>stop</td>
<td>shutdown application</td>
</tr>
<tr>
<td>status</td>
<td>briefly inquiry application status</td>
</tr>
<tr>
<td>reload</td>
<td>forces engines re-generation</td>
</tr>
<tr>
<td>reconfigure</td>
<td>forces xcics reconfiguration</td>
</tr>
<tr>
<td>kill</td>
<td>terminate all application processes by signal 15</td>
</tr>
<tr>
<td>report</td>
<td>report information about application</td>
</tr>
</tbody>
</table>

## Start up of XCICS

The first starting process is CXMAIN, which initializes the application environment.

To startup XCICS, after the configuration and tables files have been created, type at the shell prompt:

```bash
# xcicsctl start
```

During the initialization phase CXMAIN loads the environment variables and reads the configuration file in order to create a temporary file named "CX_<application_name>" in the HOME directory. This file contains all information relevant to the XCICS execution environment such as PID number, Queue identifiers and other information. It will be deleted by CXMAIN at the end of the session.

The terminal where the `xcicsctl` shell has been executed operates as the XCICS console for the current session, so that messages displayed by all running transactions or informing the operator on the normal/abnormal processing, error messages and notifications are collected on this terminal.

All what is displayed on the XCICS console will be also saved in the session log file, whose name is "xcicslog.<datetime>".

Different opportunities are available to choose whether or not a message is to be displayed on the console and/or on the session-log file, or if it must simply be ignored. Such a directive is memorized in the first byte (ocb - output control byte-) of each message record on the standard xlikeMsg message file.

No input is needed except the interrupt key to abort the XCICS session, only in the case that it is not possible to use any normal XCICS shutdown procedures.

It is possible to customize the message file, through adding user messages, deciding which message to activate or simply translating the messages in other languages. User messages will be displayed using the standard output routine "xlikeMsg" contained in the XCICS libraries. This routine may also be used by other UNIX processes which desire to send a message to the XCICS console or to store it in the session-log file.

During startup, CXMAIN may display information, warning or error messages, caused by wrongly defined or unavailable resources. In this eventuality, depending on the reason and weight of the error, XCICS will continue the loading or it terminates its execution. A complete list of all messages that may be output on the XCICS console with their meaning, can be found in paragraph 1.5.

If no severe error occurs, the initialization phase terminates when the following message appears:

```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
In case that a previous session was terminated due to a system crash or a power failure, XCICS can fail while starting up. In such a case it is advisable to perform a warm restart by means of the `-w`:

```
# xcicsctl -w start
```

which executes a dummy cleaning of its communication areas, and forces XCICS to start up.

**Shutting down XCICS**

Two shutdown procedures are available to close an XCICS application:

- Online procedure
- Command line procedure

In the online procedure, the XCICS system can be terminated normally by a terminal session (XCLIENT) with administration authorization, by entering the administrator reserved transaction code “SHUT”.

From command line, the `xcicsctl` shell can be used to close an application:

```
# xcicsctl stop
```

In this case, you are requested to confirm the session close request by the question:

```
Shut application (y/n)?
```

If “Y” or ‘y’ is replied, the XCICS session specified application will be closed.

If `-y` is specified to xcicsctl no confirmation is required, and XCICS is automatically closed. This option is useful when XCICS has to be closed automatically.

**N.B.: Please consider that XCICS will be terminated in both cases: online procedure and command line IMMEDIATELY. This has for consequence that eventual transactions still “in flight” are closed and rollbacked.**

In case you have more than one XCICS application running on your server, you will have more than one XCICS configuration files, located in different directories. The xcicsctl will close the application pointed by the xcics.conf associated to the path of directories in which you are currently working.

**Querying XCICS status**

To control if a certain XCICS application is currently active and running, enter:

```
# xcicsctl status
```

XCICS will return the application status.

**Reporting XCICS status**

To report about open connection a running processes, enter:

```
# xcicsctl status
```

XCIs will return a complete list of processes and network connections.
Killing XCICS

XCICS should never been killed, but if operator needs to shut XCICS immediatly, the kill command provides killing for each XCICS process.

```
# xcicsctl kill
```

Reloading XCICS

Due to different reasons (i.e. refresh copies of programs loaded with CALL), XCICS engines may be restarted. When a restart is invoked, XCICS immediatly respawns free engines, and schedules the respawn of busy engines, that will be restarted as soon as they complete the currently running transaction.

```
# xcicsctl restart
```

Reconfiguring XCICS

XCICS allow to change run-time its configuration, allowing changements for:

- PCT
- PPT
- MPS
- SNT

In order to change entries above, user only have to update its configuration file, and issue the "reconfigure" command.

When a reconfigure is invoked, XCICS reloads its configuration, immediatly respawns free engines, and kill busy engines and backround task, in order to avoid transaction corruption.

```
# xcicsctl reconfigure
```

3.1.3.1.2 xcicsadm

xcicsadm performs all the XCICS administrative tasks. It may be employed by system administrators to perform queries on the XCICS status and to control its functionalities, directly from the UNIX/Linux command shell or scripts.

Featured operations:

- activity & performance monitoring
- resource tables and services reports
- shutdown, reload and reconfiguration
- terminal status control
- file status control
- transaction status control
- system acquisitions
- region tuning
- mapset new-copy
XCICS Transaction Server

- URIMAPs control
- trace levels and debug control
- TCP, HTTP and MQ services control

Resource and service reports

xcicsadm allows the reporting of the following resources and services.

<table>
<thead>
<tr>
<th>tct</th>
<th>Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>pct</td>
<td>Transactions</td>
</tr>
<tr>
<td>fct</td>
<td>Files</td>
</tr>
<tr>
<td>ppt</td>
<td>Programs</td>
</tr>
<tr>
<td>dct</td>
<td>Transient data (destinations)</td>
</tr>
<tr>
<td>cct</td>
<td>Connection to remote systems</td>
</tr>
<tr>
<td>mps</td>
<td>Mapsets</td>
</tr>
<tr>
<td>ushm</td>
<td>User shared memory (GETMAIN)</td>
</tr>
<tr>
<td>tcpip</td>
<td>TCP/IP services</td>
</tr>
<tr>
<td>tn3270</td>
<td>TN3270 services</td>
</tr>
<tr>
<td>mqtrgmon</td>
<td>MQ trigger monitors (CKTI)</td>
</tr>
<tr>
<td>proclist</td>
<td>Processes of the region</td>
</tr>
<tr>
<td>tpl</td>
<td>Document templates</td>
</tr>
<tr>
<td>urm</td>
<td>URIMAPs</td>
</tr>
</tbody>
</table>

Syntax

xcicsadm [options] application_name

Options

- -T, --timeout <secs> sets the timeout
- -s, --stats shows performances stats
-l, --list <table>  list the specified resources table or services:
  - tct: terminal control table (by default only active terminals unless -a specified)
  - fct: file control table (datasets)
  - pct: program control table (transactions)
  - ppt: program processing table
  - dct: destination control table (transient data)
  - mps: mapsets
  - cct: connection control table
  - ushm: user shared memory/GETMAIN status
  - tcpip: TCPIP services
  - tn3270: TN3270 services
  - mqtrgmon: MQ trigger monitors
  - proclist: processes of the region
  - tpl: document templates table
  - urm: URIMAPs table

-X, --set-tunable <tunable>=<value>  sets the specified tunable to the provided value. Admittable tunables are:
  - server_pool_size
  - max_processes
  - max_terminals

-a, --all  reports all items of requested table(s) shows contents of shared memory when listing ushm
-x, --extended  reports extended information on requested table, if available. The extended mode is allowed for TCT and URIMAP only
-N, --resolve  resolves IP addresses
-k, --kill <terminal>  kill specified terminal
-n, --new-copy <mapset>  performs the new-copy of specified mapset
-e, --enable-transid <transid>  enables specified transid
-d, --disable-transid <transid>  disables specified transid
-o, --open-file <dataset>  opens specified dataset
-c, --close-file <dataset>  closes specified dataset
-t, --trigger <level>  set specified trigger level for selected TD queue (-q)
-q, --td-queue <queue>  specifies TD queue to handle (to use with -t)
-R, --reconfigure  requestes XCICS to reconfigure
-r, --reload  requestes XCICS to reload engine processes
-S, --shutdown
requests XCICS to shutdown

-i, --immediate
forces an immediate shutdown (with -S)

-A, --acquire <sysid>
acquires connection to specified system

--set-core-trace <level>
sets the core trace level:
  • none
  • full

--set-task-trace <level>
sets the task trace level:
  • none
  • user
  • full

--set-debug-display <display>
sets the debugger display for background terminal or transactions. Must be used in conjunction with --transaction or --terminal

--set-debug-display
resets the debugger display for background terminal or transactions. Must be used in conjunction with --transaction or --terminal

--transaction
set specified trace level for specified transaction only (use with --set-task-trace)

--terminal
set specified trace level for specified terminal only (use with --set-task-trace)

--show-trace-info
displays trace/debug data for tct list (use with -l tct)

--tn3270-status
displays the status of tn3270 services (xtnd)

--tn3270-service <name>
restricts operations on specified tn3270 service

--enable-tcpip <name>
enables the specified tcpip service

--disable-tcpip <name>
disables the specified tcpip service

--enable-tn3270 <name>
enables the specified tn3270 service

--disable-tn3270 <name>
disables the specified tn3270 service

--enable-mqtrgmon <name>
enables the specified MQ trigger monitor service

--disable-mqtrgmon <name>
disables the specified MQ trigger monitor service

--enable-urimap <name>
enables the specified URIMAP

--disable-urimap <name>
disables the specified URIMAP

--set-synctype-simple <sysid>
sets the syncpoint type to simple for <sysid>

--set-synctype-implied <sysid>
sets the syncpoint type to implied_forget for <sysid>

--set-synctype-explicit <sysid>
sets the syncpoint type to explicit_forget for <sysid>

--cx-file <file>
connects the region qualified by the CX_<file>

-?, --help
shows help text
3.1.3.1.3 xpasswd

The xpasswd utility is used to manage an XCICS external password file (xpasswd file).

With xpasswd, it is possible to perform the following operation on users:

- set user's password
- reset user's password
- enable or disable the user
- check the user password

Furthermore, it is also possible to configure the password handling features, for the XCICS regions that make use of the xpasswd file being managed, and in particular:

- set passwords minimum and maximum length
- set password management activity log file
- set password expiration period
- set the maximum allowed number of login failures

Unless overridden by the "-f" option, xpasswd manages by default the file $HOME/etc/xpasswd

Syntax:

xpasswd [options] username password

Options

General options:

- v, --verbose turns verbose mode on
- f, --file <file> uses <file> instead of default file $HOME/etc/xpasswd
- ?, --help shows the help

User management options (username must be provided):

- p, --password <password> updates the password of the user
- e, --enable enables the user account
- d, --disable disables the user account
- r, --reset resets the user's password
- c, --check <password> checks username and password

Configuration options:

- s, --show-conf shows the current configuration
- l, --conf-log <file> activates password management logging on <file>
- n, --conf-nolog deactivates password management logging
- P, --conf-password-validity <N> sets the password expiration period to <N> days If set to 0 (zero), passwords will never expire
-A, --conf-alert-expiring <N> sets the number of days in advance the user will begin receiving warnings that his password will expire. If set to 0 (zero), user will never receive warning in advance

-L, --conf-max-login-failures sets the number of consecutive login failures after that the user’s account will be disabled. If set to 0 (zero), user account will never be disabled

-m, --conf-min-password-len <N> sets the minimum admissable password length in chars

-M, --conf-max-password-len <N> sets the maximum admissable password length in chars

3.1.3.1.4 tsqctl

**tsqctl** performs inquiry and update operation on flat-file base temporary storage (TS) queues. It may list, control the status, delete, read and write all the local temporary storage queue hold by the region

**Syntax**

tsqctl [options] [queue_name]

**Options**

- `-r, --read` reads the items in the queue
- `-w, --write` reads STDIN and writes an item in the queue
- `-d, --delete` deletes the queue
- `-q, --query` check the status of the queue
- `-l, --list` list the available queues
- `-L, --lines` with --write, reads from STDIN and writes each line read as a separate item
- `-i, --item <i>` with --read starts reading from item <i>
  with --write rewrites item <i>
- `-u, --update-position` updates the queue current position indicator
- `-a, --region <name>` handles the TS queues of the region <name>, if active. Otherwise gets the first region active for the current user.
- `-p, --path <path>` handles the TS queues in <path>, even if a region is not active
- `-?, --help` shows the help text

**Example**

Listing the temporary storages of the active regions
# tsqctl --list
connected to shared memory of region CICSLX02
using repository: /home/fabrizio/tmp

<table>
<thead>
<tr>
<th>queue</th>
<th>cur.item</th>
<th>num.item</th>
<th>size file</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_S00L2</td>
<td>1</td>
<td>1</td>
<td>65048 435f53304f304c32000000000000000000.tsq</td>
</tr>
<tr>
<td>C_S00L3</td>
<td>1</td>
<td>1</td>
<td>65048 435f53304f304c33000000000000000000.tsq</td>
</tr>
<tr>
<td>TSQTEST</td>
<td>0</td>
<td>5</td>
<td>195144 5453515554555354000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010001</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303031000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010002</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303032000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010003</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303033000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010004</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303034000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010005</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303035000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010006</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303036000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010007</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303037000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010008</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303038000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010009</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303039000000000000000000.tsq</td>
</tr>
<tr>
<td>TN010010</td>
<td>100</td>
<td>100</td>
<td>3284924 544e303130303030000000000000000000.tsq</td>
</tr>
</tbody>
</table>

Reading the content of a queue:

# tsqctl --read --item=1 TN010004
connected to shared memory of region CICSLX02
using repository: /home/fabrizio/tmp

queue TN010004:
item leng. content (first 80 bytes)

1  80  TN010004.0001
2  80  TN010004.0002
3  80  TN010004.0003
4  80  TN010004.0004
5  80  TN010004.0005
6  80  TN010004.0006
7  80  TN010004.0007
...

Writing an item at bottom of the queue:

# tsqctl --write TN010008
connected to shared memory of region CICSLX02
using repository: /home/fabrizio/tmp
adding item(s) at bottom
TEST ENTRY
<CTRL-D>

1 item(s) added (numitems=101)

Rewriting an item in the queue:

# tsqctl --write --item=2 TN010008
connected to shared memory of region CICSLX02
using repository: /home/fabrizio/tmp
rewriting item 2
Deleting a queue:

tsqctl --delete TN010008
connected to shared memory of region CICSLX02
using repository: /home/fabrizio/tmp
queue deleted

3.1.3.1.5 Other

xstart

The xstart utility is used to start transactions on XCICS from script shell.

Syntax

```
xstart [-s][-h][-t<termid>][ -a<applid>] [ <transid>]
```

If no transid is specified input is taken from standard input.
When passing iput from standard input additional data may be specified.

Options

```
-s   silent mode
-t <termid>   task is started on specified termid (must be defined as virtual)
-a <applid>    specifies application name
-v   verbose mode is turned on
```

Example

```
# xstart -a TESTCICS ATR0
# xstart -a TESTCICS
ATRO DATA FOR RETRIEVE
```

xqio

The xqio utility is used to upload/download data to/from temporary storage.

Syntax

```
xqio [options] ( get| put)
```

Commands

get    download specified file from XCICS temporary storage
put    upload specified file to XCICS temporary storage
Options

- `v`         verbose mode
- `a`         interacts with specified
- `t`         use specified temporary storage
- `T`         run loading on specified virtual terminal
- `l`         specifies temporary storage record length
- `s`         script name to be activated when download is completed
- `P`         program name to be linked after when upload is completed
- `i`         forces XCICS to use a temporary storage named CFTR

Transaction

xqio is backed up by an XCICS transaction called XQIO.

To configure it add the following lines to the xcics.conf:

```
define program name=xqio;
define transaction code=XQIO, program=xqio, twa=512, protection=64;
```

Called program

XQIO may link a use program. This program is linked passing a commarea as follow:

```
01 COMMAREA.
   02 FILLER PICTURE XX.
   02 FILLER PICTURE XX.
   02 CFTR-FILE PICTURE X(8).
   02 CFTR-QUEUE.
       03 CFTR-NAME PICTURE X(4).
       03 CFTR-TERM PICTURE X(4).
   02 RET-CODE PICTURE X(2).
   02 FILLER PICTURE X(6).

CFTR-QUEUE contains the TS name.
```

RET-CODE contains the return code passed back by linked program. If user program returns "DL" the TS is rewritten to the file.

Example

```
# xqio -a TESTCICS -l 100 put $HOME/file01.dat
# xqio -a TESTCICS -t A01DA3BB -l 80 get $PWD/ts01.txt
```
3.1.3.2 XCICS supplied transactions

XCICS is deployed with a set of transactions to perform special functions during the execution of an XCICS session. All functions are performed interactively, by means of administration's reserved transactions executed during the XCICS session.

**Configuration**

No special setting must be performed to define the following transactions. Their protection level can be controlled using the `set admin_protection_level` directive. By default, a protection level of 64 is assumed for XCICS provided transactions.

**CESN**

CESN allows to sign on and sign off in the XCICS application. If region's auth_mode has been set to "passwd" or "ldap", CESN will allow the operator to manage his password.

Usage: refer to its online help.

**CEBR**

CEBR allows to browse the Temporary Storage queues currently present in the system.

Usage: refer to its online help.

**CEB2**

Similar to CEBR, CEB2 allows to browse the Temporary Storage queues with names longer than 8 characters, currently present in the system.

Usage: refer to its online help.

**CETD**

CETD allows to browse, edit and set trigger level Transient Data queues defined in the system.

Usage: refer to its online help.

**CEDF**

CEDF allows to trace the execution of user programs, command by command.

Usage: refer to its online help.

**CEFH**

CEFH allows to browse and edit XCICS datasets.

Usage: refer to its online help.

**CESQ**

CESQ allows to easily monitor the status of XCICS displaying:
• engine occupation
• wait queue length
• average trx response time
• average and maximum wait queue length & time

Usage:
CESQ

**CEDB**

CEDB allows to activate and change the debug information for a terminal or a transaction

Usage:

CEDB (ON|OFF)[,TERM=<termid>| TRAN=<transid>][,DISPLAY=<display>][,NOAUTO]

Parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ON</strong></td>
<td>activate debug system</td>
</tr>
<tr>
<td><strong>OFF</strong></td>
<td>deactivate debug system</td>
</tr>
<tr>
<td><strong>TERM</strong></td>
<td>specifies the terminal for which debug must be activate. If omitted current terminal is assumed.</td>
</tr>
<tr>
<td><strong>TRAN</strong></td>
<td>specifies the transaction name for which debug must be activate</td>
</tr>
<tr>
<td><strong>DISPLAY</strong></td>
<td>sets the X/server DISPLAY address where debug system (both Microfocus and ACUCOBOL) must be started. It is mandatory for:</td>
</tr>
<tr>
<td><strong>TTY</strong></td>
<td>sets the target terlnet (device) to debug with ACUCOBOL</td>
</tr>
</tbody>
</table>

- transaction-oriented debugging
- background terminal debugging
- ACUCOBOL debugging

If set to "*" (asterisk) the display 0 (zero) on the IP address of the connected terminal is used

i.e.:
CEDB ON,DISP=192.168.1.32:0
CEDB ON,DISP=*,TRAN=ACCT

i.e.:
CEDB ON,TTY=pts/4
**THIN**
sets the ACUCOBOL thin client address to debug with ACUCOBOL thin client.
XCICS must be configured with "allow_acu_thin=yes"
I.e.
CEDB ON, THIN=192.168.1.32:8000

**NOAUTO**
If specified, debug requestes for current terminal, automated disconnect/reconnect flow is disabled, and let the user to perform the disconnect/reconnect operation according to his preferences.

If no parameter is provided, CEDB display the current status of debugging for the terminal.

**CETR**
CETR allows to activate and change the trace level for the whole application or for a single transaction.

CETR [ CORE=(NONE|FULL) ][ , TASK=(NONE|USER|FULL) ][ , TERM=<TERM> ][ , TRAN=<TRAN> ]

<table>
<thead>
<tr>
<th>CORE</th>
<th>set the core trace level. Admitted values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• NONE</td>
</tr>
<tr>
<td></td>
<td>• FULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TASK</th>
<th>sets the trace level for all terminals. Admitted values:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• NONE</td>
</tr>
<tr>
<td></td>
<td>• USER</td>
</tr>
<tr>
<td></td>
<td>• FULL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TERM</th>
<th>restricts the trace level set-up to the specified terminal</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TRAN</th>
<th>restricts the trace level set-up to the specified transaction</th>
</tr>
</thead>
</table>
3.1.3.3 Obsolete administrative functions

XADM is a suite of administration functions provided in XFRAME 3.x and used to display the XCICS activity or to perform special functions during the execution of an XCICS session. All functions are performed interactively, by means of administration’s reserved transactions executed during the XCICS session.

Configuration

In order to make use of XADM transactions inside XCICS, they must be correctly set up. The following include must be defined in the configuration file:

```
include $XFRAMEHOME/etc/xadm.conf
```

Then the XADM programs must be related to corresponding transaction names and operator security information:

```
define transaction code=XSSN, program=xssn, twa=512, protection=64;
define transaction code=SHUT, program=xshut, twa=512, protection=01;
define transaction code=XLOAD, program=xload, twa=512, protection=01;
define transaction code=XADM, program=xadm, twa=512, protection=01;
define transaction code=XQUERY, program=xque, twa=512, protection=01;
define transaction code=XDB, program=xdbg, twa=512, protection=01;
define transaction code=XFILE, program=xfile, twa=512, protection=01;
define transaction code=XCP, program=xcp, twa=512, protection=01;
define transaction code=STAT01, program=xstat01, twa=512, protection=01;
define transaction code=STAT02, program=xstat02, twa=512, protection=01;
define transaction code=STAT, program=xstat, twa=512, protection=01;
define transaction code=XTCT, program=xtct, twa=512, protection=01;
define transaction code=XTBR, program=xtbr, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
define transaction code=XTDR, program=xtmdmu, twa=512, protection=01;
```

XADM transaction

The XCICS administration facility comprehends a number of functions useful to control and display normal activity of online sessions, and to correct unexpected error situations.

You have also the possibility to open and close dynamically data sets which have been defined for XCICS, or manipulate records, without interrupting other transactions, processing other datasets.

The administration facility can be requested from every authorized user by entering the special transaction code XADM.

Each function can be started by entering its corresponding function code in the selection field. Supported function codes are:

- To activate the dataset inquiry/update utility;
- To monitor terminal activities;
- To obtain current session overviews;
- to load a fresh copy of mapsets;
- To modify the debugging levels for XCICS /XVSAM;
To open and close datasets dynamically;

The usage of all available function keys will be explained for each map. In general PF3 is used to exit or return to the previous menu.

**XSSN transaction**

XSSN allows to sign on and sign off in the XCICS application.

If an external password file has been provided to XCICS, user may use XSSN to change their password.

**XQBR transaction**

XQBR allows to browse the Temporary Storage queues actually present in the system.

**XTDR transaction**

XTDR allows to browse, edit and set trigger level Transient Data queues defined in the system.

**XQUE transaction**

XTDR allows to easily monitor the status of XCICS displaying:

- engine occupation
- wait queue length
- average trx response time
- average and maximum wait queue length & time
3.1.4 Tuning

The tuning of a system running XCICS/TS involves on:

- IPC resources
- shared memory

**Inter-Process Communication resources**

XCICS/TS processes inter-communicate by IPC resources (semaphores and message queues). To ensure a correct IPC communication into the region some system parameters must be set.

The values are strictly dependant on OS and H/W architecture. This table contains only suggested values.

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>semmsl</td>
<td>256</td>
</tr>
<tr>
<td>semmens</td>
<td>32000</td>
</tr>
<tr>
<td>semopm</td>
<td>100</td>
</tr>
<tr>
<td>semmni</td>
<td>128</td>
</tr>
<tr>
<td>msgseg</td>
<td>32767</td>
</tr>
<tr>
<td>shmmni</td>
<td>4096</td>
</tr>
</tbody>
</table>

**Shared memory availability**

A sufficient amount of shared memory must be available for each XCICS/TS region.

The amount of shared memory required by a region depends on:

- number of entries in the resource tables (PCT, PPT, etc)
- size of the main storage reserved for user programs (GETMAIN), as requested in the region configuration file.

When the region starts up, it shows, in the main console log, the amount or requested shared memory and tries to allocate it. If the allocation fails, the region startup fails.

In this situation, the kernel of the system must be configured to allow the allocation of the amount of memory requested.
3.1.4.1 Region performances

The performances of an XCICS/TS region may be tuned regulating two factors:

- the server pool size
- size and number of memory buffers

Server pool size

The server pool size is number of transaction server processes which carry out all the transactional work activated by the on-line connections (terminals) and deeply affects the performance of the region.

When a transaction is requested, XCICS/TS assigns the task to the first process in pool available. If no process is available, it puts the request in the "Wait Queue" (WQ), waiting for a server process to become available.

Since the number transaction server processes (pool size) determines the number of concurrent task that can run on a region, underestimating it can cause online transaction to wait in the WQ, and, if transactions are waiting, so are users. Therefore, you need to allocate enough transaction server processes to keep the WQ at a maximum of 2 or 3 most of the time.

Of course over allocation of the pool size keeps WQ empty, but it also over allocates the system virtual memory, which may produce unnecessary paging or swapping.

This number of transaction server processes is tuned by the "set server_pool_size" in the configuration time and may be dynamically changed at run time.

Tuning the parameter

The tuning of this parameter depends a number of factors. The most obvious factor is the expected workload of the server.

You need to consider the frequency with which transaction requests arrive, the required response time, and the type of application programs:

I/O-bound programs

Programs that perform a lot of I/O requests are often dormant while they are running, while they wait for the responses to their requests. However, processing I/O requests can sometimes make considerable demands on the CPU. You need to consider whether you might need more transaction server processes for this type of service.

CPU-bound programs

Programs that perform few I/O requests are usually constrained by the use they make of the central processor. With CPU-bound services, fewer transaction server processes usually work better, because the central processor switches between the running tasks, and this is an overhead.

Monitoring the region

The exact tuning of the server process pool may achieved monitoring the WQ when the region is alive and serving.

You may monitor the WQ with the command line utility "xcicsadm" or with the on-line transaction CESQ.

These tools show:

<table>
<thead>
<tr>
<th>current wait queue length</th>
<th>the number of transaction currently waiting to be processed</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum wait queue length</td>
<td>the maximum length of the queue detected</td>
</tr>
<tr>
<td>maximum wait queue time</td>
<td>the maximum time a transaction has been waiting in the queue</td>
</tr>
</tbody>
</table>
average wait queue time | the average time the transactions wait in queue
---|---
total transactions performed | the total number of transactions performed by the region

Additionally, XCICS/TS collects different information for each transaction executed, including:

- elapsed time
- CPU time
- I/O count

These information may be programmatically analyzed or converted into a report by the "xreport" tool. For further reading on transaction statistics, please refer to the "Logging, Accounting & Journaling" section.

**Background tasks**

Background tasks normally do not run in the process shared pool, but they use a dedicated process created when the task starts and dismissed when the task end.

**TCP/IP services**

Tasks activate as response to a request coming through TCP/IP services may be run as standard background task in dedicated process, or in a process of the server process pool (use_engine parameter on the service definition). It depends on the service definition at configuration time.

Normally you may decide the execution method depending on the type of the request:

**Short-running requests**

Short-running requests are requests where there is just one interaction between the client and the service; the client request arrives, the service runs, and a response is returned to the client. Requests from Web services clients are normally short-running. With short-running requests, you only need to consider whether or not the service itself is CPU-bound and handle it in the process pool, or I/O-bound and handle the service as a standard background task.

**Long-running Client Requests**

Long-running client requests are requests where the same client can make repeated requests of a service and needs data to be preserved between invocations of the service. In this case the service must be handle a standard background task.

**SNA sessions**

A SNA session is a process waiting for requests coming from the SNA network via APPC (i.e. DTP, DPL, etc.). This process receives and handle the request. When the request is satisfied, it becomes available for a new request.

The number of SNA sessions depends on the expected number of contemporary requests and on the their type, as described for TCP/IP services.

SNA sessions are defined with one or more "define session" directive.

**Memory buffers**

Memory buffers are the areas used by the XCICS/TS core process to exchange data with the transaction server processes.

Each time the main process needs to exchange data with a transaction server process, a free buffer of the requested size is used: if no suitable buffer is free, the communication is hold until one becomes available.

Therefore buffers size and number regulates the speed of the internal communication, and a larger number and size of
buffers speeds up the internal communication.

This setting is very important especially for those application which make a large usage of "in-memory" temporary storage, transient data and background task activation.

Memory buffers may be tuned using the "add bufferset" directive in the configuration file.
3.1.5 Security management

Very often, in an OLTP system, not all the users are entitled to execute all the defined operations. The system administrators may desire that only a certain group of authenticated users could be allowed to use a specific set of resource.

I.e. the administrators wants to the invoice department users to execute only invoicing transactions, and marketing department users to execute only CRM transactions.

XCICS/TS security system is based on the concepts of transaction protection level and users.

The transaction protection level is numeric identifier, varying from 1 to 64, associated with a transaction identifier (PCT entry). More transactions may share the same protection level.

The user is the identifier of the operator connected to the system. Each user has an one authorization set that allows him to execute those transactions with a compatible protection level.
3.1.5.1 Transaction protection

As described before only the users with an authorization set that matches the protection level requested by a transaction definition, may execute the it.

The protection level is number varying from 1 to 64, and it is simply an unique identifier. This means that there is no relationship between a level and its predecessors or successors and therefore level N has no more or less rights that N-1 or N-2.

Each transaction is defined in the XCICS/TS configuration file with the "define transaction" directive and it is provided with a protection level, like in this sample:

```plaintext
define transaction code=TAB1, program=TAB1PGM, twa=512, protection=3;
define transaction code=TAB2, program=TAB2PGM, twa=512, protection=3;
define transaction code=LOGO, program=LOGOPGM, twa=512, protection=25;
define transaction code=TYYY, program=TYYYPGM, twa=512, protection=8;
```

This means that only those users with an authorization set including level 3 can execute the transaction TAB1 and TAB2, only those users with an authorization set including level 25 can execute the transaction LOGO and only users with authorization set including level 8 can execute TYYY.

The users are defined with the "define user" directive and, by means of the "auth" attribute, an authorization set is bind to them:

```plaintext
define user name=GUEST, auth="1,25";
define user name=OPERATOR, auth="1-3,25";
define user name=EMPLOYEE, auth="8-12";
define user name=MASTER, auth="1-64";
```

In this sample XCICS/TS will allow the execution of:

- LOGO for GUEST
- TAB1, TAB2 and LOGO for OPERATOR
- TYYY for EMPLOYEE
- all transactions for MASTER
3.1.5.2 Users and userclass definition

Users are defined with the "define user" directive. Each "define user" creates an user entry in the SNT, providing all the attributes and authorizations for the users.

When a large number of users must be defined, administrators may need to group users into classes sharing the same authorizations and/or attributes (i.e. OPID, OPKEY).

This task is achieved by means of the userclass definition.

An userclass is a collection of user attributes: all the users belonging to the userclass inherit the attributes defined in it. Those attributes defined in the user's definition too (define user) override the userclass's ones.

The userclass is defined with the "define userclass" directive.

Default user

XCICS/TS requires a default user to identify the operator when connected, before he does a sign-on operation. This user must be defined as usual with a "define user" directive and declared to XCICS/TS with the "set default_user" directive (see sample below).

System users

XCICS/TS requires two system users: the sysuser to execute the background task, and the tduser to execute the task triggered by Transient Data queues.

These users may be manually defined with a standard "define user" directive including the "sysuser" or "tduser" parameter in the declaration.

If they are not defined, XCICS/TS automatically creates the $$DFHSYS and $$DFHTDQ users with full authorizations.

A practical example

```
# using external password repository
# set passwd_file=/HOME/etc/xpasswd
set default_user=GUEST;
# users classes
#
define userclass name=CLSGUEST, opid=GST, auth="1", opclass=000000;
define userclass name=CLSOUSER, opid=USR, auth="1-16,32,33,40-50", opclass=00FFFF;
define userclass name=CLSAADMIN, opid=ALL, auth="1-64", opclass=FFFFFFFF;
#
define user name=GUEST, userclass=CLSGUEST, username="Guest user";
# standard users definition
#
define user name=LSKYWALK, userclass=CLSOUSER, username="Luke Skywalker";
define user name=JKIRK, userclass=CLSOUSER, username="Capt. James Kirk";
define user name=DECKARD, userclass=CLSOUSER, username="Rick Deckard", opid=USR;
#
# super-user
#
define user name=FABRIZIO, userclass=CLSAADMIN, username="Administrator user";
```
3.1.5.3 Authentication modes

XCICS/TS supports 4 authentication system to verify the credentials of users performing a SIGNON:

- basic
- passwd
- ldap
- user

The mode is set using the "set auth_mode" directive in the XCICS/TS configuration file.

**BASIC mode**

In the BASIC mode, users are defined in the configuration file with the "define user" directive and/or with the "snt_loader" exit_program.

The user’s password is directly contained in the configuration file as clear text.

Users cannot change their password.

**PASSWD mode**

In PASSWD mode, users are defined in the configuration file with the "define user" directive and/or with the "snt_loader" exit_program.

The encrypted passwords are stored in an external file. By default XCICS/TS uses the file name $HOME/etc/xpasswd; to provide an alternative path the "set passwd_file" directive may be used.

With passwd mode, users may change their password online (with CESN or EXEC CICS SIGNON NEWPASSWORD): when required, XCICS/TS will update the passwd file with the new encrypted password.

Additionally, with xpasswd mode, XCICS/TS allows to the administrators to improve the security handling, providing:

- minimum and maximum password length constraints: users cannot change the new passwords match the specified that violate the specified constraints are automatically rejected
- password expiration period: users are obliged to re-new their password after a certain period of time
- password expiration warning: XCICS/TS warns users when the expiration date of their password is incoming
- login failures handling: after a certain number of login failures, the user account is disabled
- password reset: administrators may reset the user's password, obliging him to re-new the password
- password management logging: password changes, login failures and configuration changes are tracked in a log file

The "xpasswd" command line utility may be used to activate and configure all the security settings above (using the its "configuration options") as well as to manage the user account status (enable and disable) and password (reset or definition).

The passwd file must contain the password of all users declared: when an user is declared in the configuration file but it is not found in the passwd file, its SIGNON is denied. Therefore, whenever an user is added to the configuration file, the administrators must take care to put its credentials in the passwd file too (with xpasswd).

The same xpasswd file may be use by more than one region: in this case all the regions will use the same security settings and the user's passwords will be the same on all regions. Note: all regions must have read and write permissions on the file.

**LDAP mode**

With LDAP, user are may be defined in the configuration file with the "define user" directive or taken by an LDAP server.
The authentication of users credentials is demanded to the LDAP server, so, when an user request a SIGNON, XCICS/TS binds the LDAP server, passing the user DN (LDAP Distinguish Name) and its password: if the LDAP server authenticates the user, XCICS/TS completes the SIGNON successfully.

Users may change their password according to the LDAP server settings and definitions.

For further information on XCICS/TS and LDAP cooperation, please refer to "LDAP integration" section.

**USER mode**

In USER mode, users are defined in the configuration file with the "define user" directive and/or with the "snt_loader" exit program.

The SIGNON is verified by an exit program of type "user_signon". This is an user written program the communicates to XCICS/TS if the credentials provided by the operator are authorized.

In this scenario, the password change may be allowed by the presence of an "user_chpasswd" exit program.
3.1.5.4 LDAP integration

XCICS/TS may use LDAP servers to define and identify its users, through an LDAPv3 interface.

When a region is configured to use LDAP authentication mode (parameter auth_mode set to "ldap"), XCICS/TS will connect, during the startup and the reconfiguration phase, one or more LDAP servers to populate its SNT (Signon table) with the users.

So, first of all, an LDAP entry (user) must be created for the XCICS/TS region.

When the region starts (or is being reconfigured) it connects the LDAP server with the its own credentials, and it searches the users that will populate its SNT.

XCICS/TS recognizes the LDAP users to add in its SNT according to three elements:

- the search base
- the userclass filters
- the presence of the USERID attribute

The search base is the path in the LDAP directory, from which the research of users starts. So, only those users who have a Distinguish Name located under the search base branch may be potential users of the running region.

The research is then driven by the LDAP filters of the XCICS userclasses: each userclass provided with an ldap_filter parameter, causes an LDAP research from the search base described above and filtered by the its own filter.

Additionally, XCICS/TS refines these searches forcing the presence of the USERID attribute, which is the field of the entry whose value is used as CICS USERID.

Each entry returned by a single userclass filter creates in SNT an XCICS user belonging to the class and identified by the value contained in the USERID attribute. The entry's DN is also stored in the SNT to be used later for user sign-on.

If the same user is returned by more than on search, it is defined (and therefore bind) to the first userclass that find it.

Afterwards, whenever an operator requests a sign-on (by CESN or EXEC CICS SIGNON), XCICS/TS will connect the LDAP server to authenticate the user's DN with the password provided by the operator.

Moreover, according the LDAP server configuration, XCICS/TS permits to the users to manage their passwords, and, and if the LDAP client library supports the necessary APIs, it may notify the (through CESN of EXEC CICS SIGNON) the following states:

- password expiring
- password reset
- password expired
- user locked/revoked

For further information on LDAP filters and search bases, please refer to your LDAP server documentation or to RFC2254 (The String Representation of LDAP Search Filters).

Runtime configuration

To make use of LDAP authentication mode, an LDAP client must be installed on the system. The XCICS/TS LDAP interface must be re-compiled with the headers of the installed LDAP implementation and then linked with the LDAP client API libraries.

The LDAP client must support the LDAPv3 protocol, and its APIs must conform to RFC1823 (The LDAP Application Program Interface).

For further information on runtime configuration and linking, please refer to the "Installation Guide".

LDAP configuration

To allow a region to connect an LDAP server and retrieve users, an LDAP user for the region must be defined. The user
must be able to connect the LDAP server and to browse entries. The password for this user should not be bind to any password expiration policy, to avoid unexpected situations.

A sample LDAP user for the region:

```
ldapsearch -h linux64 -D "uid=cicslx02,ou=People,dc=example,dc=com" -w secret 
-b "dc=example,dc=com" "(uid=cicslx02)"
```

```
# extended LDIF
#
# LDAPv3
# base <dc=example,dc=com> with scope subtree
# filter: (uid=cicslx02)
# requesting: ALL
#
# cicslx02, People, example.com
dn: uid=cicslx02,ou=People,dc=example,dc=com
uid: cicslx02
givenName: XCICS/TS
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetorgperson
sr: Linux02
cn: XCICS/TS Linux02
```

**Region configuration**

When the LDAP user is created, the region must be configured to access the LDAP server and retrieve the users. Therefore, the region authentication mode must be set to "ldap" and the following information must be provided:

- the LDAP server: hostname and port (optional)
- the region login credentials: DN & password
- LDAP search base

I.e.

```
set auth_mode= ldap ;
set ldap_servers=" linux64 ";
set ldap_login_dn= "uid=cicslx02,ou=People,dc=example,dc=com ";
set ldap_password= " secret ";
set ldap_base= " ou=People,dc=example,dc=com ";
```

**Userclass configuration**

As described before, the LDAP users retrieval is driven by XCICS userclasses. This means that only those LDAP users whose attributes satisfy the userclass filter are defined in SNT and therefore will be able to sig-on in the region.

For each userclass you want to bind LDAP users, you have to provided a valid LDAP filter in the ldap_filter parameter.

I.e.
The USERID attribute

As well known, CICS (and so XCICS) identifies each operator by an 8 bytes unique identifier, called USERID. When the users are retrieved from LDAP, XCICS/TS requires to know their USERID: the USERID attribute is the LDAP field whose value contains the user’s USERID.

Due to this reason, the searches for users defined in the userclasses are restricted only to those entries containing an USERID attribute.

By default, XCICS/TS retrieves the USERID from the LDAP attribute named "uid", according to the ISO definition of the LDAP class "inetOrgPerson".

If you want to store the user's USERID in a different attribute, you may set the parameter ldap_attribute_userid to the name of the your desired attribute.

So, if the parameter ldap_attribute_userid is not defined in the configuration field, XCICS will extract from LDAP all the users under the configured search base, who satisfy the userclass filters and containing the attribute "uid", whose value will be use as USERID.

If, for example, the USERID is contained in an attribute that is not "uid", you have to set the ldap_attribute_userid parameter with the name of the attribute that contains it.

I.e.

```bash
set ldap_attribute_userid="myXcicsUserId";
```

In this situation, XCICS will extract from LDAP all the users under the configured search base, who satisfy the userclass filters and containing the attribute "myXcicsUserId": the value of this attribute will be the USERID of the user.

Anyway, the value of the USERID attribute must be 8 bytes long: if the value should be longer, XCICS will use only the first 8 bytes.

The USERNAME attribute

By default, the USERNAME of the users (that is the value returned by ASSIGN USERNAME) is retrieved by the value of the attribute named "cn" (Common Name).

If you want to get the USERNAME from a different attribute you may set the parameter ldap_attribute_username.

I.e.

```bash
set ldap_attribute_username="sn";
```

Passwords

Normally XCICS handles only uppercase passwords up to 8 characters long.

When in LDAP authentication mode, XCICS/TS may handle password up to 32 characters.

Additionally, by means of the mixedcase_passwords, you may decide to allow the usage of mixed case passwords.

I.e.
Configuration verification

To verify the exactness of the configuration as well as to discover eventual problems, the best system is to use an LDAP searching tool to perform the same search XCICS/TS would do, and check if the returned entries satisfy the userclass you are analyzing.

The search should be qualified by:

<table>
<thead>
<tr>
<th>Description</th>
<th>xcics.conf parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP server</td>
<td>ldap_server</td>
</tr>
<tr>
<td>bind DB</td>
<td>ldap_login_dn</td>
</tr>
<tr>
<td>bind password</td>
<td>ldap_password</td>
</tr>
<tr>
<td>base dn for search</td>
<td>ldap_search_base</td>
</tr>
<tr>
<td>filter</td>
<td>(userclass ldap_filter) AND (uid=*)</td>
</tr>
</tbody>
</table>

So, if using the OpenLDAP search tool:

```bash
ldapsearch -h linux64 
  -D "uid=cicslx02,ou=People,dc=example,dc=com" 
  -xw secret 
  -b "dc=example,dc=com" 
  "(&(objectClass=inetOrgPerson) (ou=Product Development) (uid*))"
```

A practical example

This is a sample XCICS/TS configuration for LDAP:

```bash
set auth_mode= ldap ;
set ldap_servers= "linux64 ";
set ldap_login_dn=" uid=cicslx02,ou=People,dc=example,dc=com " ;
set ldap_password=" secret ";
set ldap_base=" ou=People,dc=example,dc=com " ;
set mixedcase_password=yes;

define userclass name=DEVELOPE, opid=OVP, auth="1-32", opclass=FFFFFE, 
  ldap_filter="/ & objectClass/inetOrgPerson (ou=Product Development) " ;
define userclass name=ACCOUNT, opid=ACT, auth="33-64", opclass=FFFFFE, 
  ldap_filter="/ & objectClass/inetOrgPerson (ou=accounting) " ;
```

This configuration will cause the following:

1. Authentication mode is LDAP
2. XCICS/TS will use the LDAP server "linux64" on the standard LDAP port
3. XCICS/TS will login to the LDAP server identified by the DN "uid=cicslx02,ou=People,dc=example,dc=com" and by the password "secret"

4. XCICS/TS will search its users from the search base "ou=People,dc=example,dc=com"

5. The userclass "DEVELOPE" will contain all those users satisfying the filter "&(objectClass=inetOrgPerson) (ou=Product Development))" and containing the attribute "uid"

6. The userclass "ACCOUNT" will contain all those users satisfying the filter "&(objectClass=inetOrgPerson) (ou=accounting))" and containing the attribute "uid"

7. At sign-on time (CESN) users will be able to use mixed case passwords.

In terms of LDAP search, the userclass "DEVELOPE" is the result of the search shown in the previous paragraph.
3.1.6 Data integrity

XCICS ensures data integrity on file through the XVSAM logging features.

All operations on files are tracked on the recovery log file. Every XCICS instance has its own recovery log containing all information concerning both committed and "in-flight" transactions.

The recovery log is used to perform:

- Warm start
- After-Image processing

The recovery log is by default named RECOVERY.LOG and using the configuration directive recovery_log_path, it is possible to define the path where XCICS will create the file. For safety reasons, it's better to keep this file not on the same disk where XVSAM data are stored.

**XCICS Warm start**

Normally, when XCICS is cold started (xcicsctl start), it generates a new recovery log. In case of abnormal XCICS termination (i.e. hardware fault with cluster switch-over), it is possible to restart XCICS in "warm" mode (xcicsctl warmstar).

When XCICS is warm-started, it reads the existing recovery log, and backs-out all the "in-flight" transactions of the previous instance. In other words it removes all uncommitted transaction from XVSAM files.

Additionally, it preserves also existing Transient Data records, to continue eventual uncompleted background processing.

Warm start processing on XVSAM data is also possible by command line using the xvsam utility (--warm-start option).

**After-Image processing**

On some situations, i.e. serious hardware faults, it is necessary to rebuild the XVSAM information starting from a backup.

With the after-image processing, it is possible to restore an existing backup and use the XCICS recovery log to apply all changes occurred on files tracked in the log.

In this case, user have to:

1. restore the latest XVSAM backup
2. re-apply the changes tracked on files using the xvsam utility (--recover option)

Of course, backup data and recovery log must be congruent.
3.1.7 Print services

On-line applications have obviously printing capabilities, and, normally, there are 2 ways to print files under a CICS monitor:

- using the BMS facility, to print directly on a printer terminal
- using the SPOOL facility, to print through the spooling system
3.1.7.1 Using BMS facility

The first method to print data under XCICS is to use BMS commands (SEND, SEND TEXT, SEND MAP, etc.) in a transaction started on a terminal defined as "printer" in the XCICS configuration file (xcics.conf).

This transaction, once started, will issue BMS commands to write the print file, which is a standard UNIX flat file containing control characters as new-line, carriage-return an new-page.

This file is temporary stored in the path defined in the configuration parameter spool_path.

When transaction completes XCICS will provide to spool the generated file through the command specified in the configuration parameter print_command.

The print_command is a user defined script/executable which takes care of forwarding the file to the printing subsystem or to do whatever the user needs.

XCICS invokes the print_command providing the following arguments:

1. the absolute path name of the printout file
2. the device name of the printer terminal which generated the printing
3. the transaction name of the transaction which generated the printing

An example

We suppose to have an XCICS printer terminal named PRT0 and a print_command named tplp.

The xcics.conf will contain the following lines:

```
define terminal name=PRT0, type=printer, device=PRINTER0;
set print_command=$HOME/bin/tplp
set spool_path=$HOME/tmp
```

When the transaction started on PRT0 completes its work, XCICS will issue the command:

```
$HOME/bin/tplp $HOME/tmp/PRT0_<PID>.dat PRT0 PRINTER0
```

The "tplp" script could be defined as follow (an example):

```
#!/bin/ksh
#
# sample printing shell which
# assumes that every XCICS terminal has
# a corresponding lp class with the same name
# filename=$1
terminal_name=$2
transid=$3
lp -d $terminal_name $filename
```
3.1.7.2 Using SPOOL facility

The second method to print data under XCICS is to use SPOOL commands (SPOOLOPEN, SPOOLWRITE, SPOOLCLOSE.).

This transaction, once started, will issue SPOOL commands to write the print file.

The SPOOL commands provide a long list of parameter to identify and describe the attributes of the printout (FORM, FC; DESTINATION, etc.) and because of it XCICS will include at the beginning of the generated file a list of all the parameters supplied in the SPOOLOPEN command, prefixed by the tag /spoolopen/.

This list starts with /spoolopen/begin-of-properties tag and will end with the tag /spoolopen/end-of-properties.

I.e.

```
/spoolopen/begin-of-properties
/spoolopen/title=/spoolopen/class=C/spoolopen/asa=true/spoolopen/copies=3/spoolopen/sep=false/spoolopen/nsep=false/spoolopen/form=A6XG/spoolopen/fo=f/spoolopen/line length=0/spoolopen/lines=0/spoolopen/destination=/spoolopen/record length=133/spoolopen/node=NODE1/spoolopen/end-of-properties

********************************************
* Sample SPOOL printout
********************************************
Surname: Smith
Name: Mary
```

This file is temporary stored in the path defined in the configuration parameter spool_path.

When transaction completes XCICS will provide to spool the generated file through the command specified in the configuration parameter spool_command.

The spool_command is a user defined script/executable which takes care of forwarding the file to the printing subsystem or to do whatever the user needs.

XCICS invokes the spool_command providing the following arguments:

1. the absolute path name of the printout file

Writing JCLs

Sometimes SPOOL facility is used on mainframe to write directly to the SYSREADER, in order to dynamically write and start JCLs.

Under XCICS application may do the same: the spool_command will have to be in this case "intelligent" enough to understand (normally from the file prefix name, or from its contents) if a spooled file is a printing or a JCL, and consequently decide if to print it or handle it as a JCL.

A rehosted online application which dynamically writes JCL should be converted in order to write directly UNIX scripts, but sometimes user may decide to do not this activity. Therefore a dynamic conversion is required (see XMVSCONV, the JCL converter).

An example

We suppose to have a spool_command named tpsh.
The \texttt{xcics.conf} will contain the following lines:

\begin{verbatim}
set print\_command=$\texttt{HOME/bin/tpsh}
set spool\_path=$\texttt{HOME/tmp}
\end{verbatim}

If a transaction invokes SPOOL commands, when completed, XCICS will issue the command:

\begin{verbatim}
$\texttt{HOME/bin/tpsh $HOME/tmp/<REPORT>.<DATE>.<PID>.<ID>}
\end{verbatim}

Where <\texttt{REPORT}> is the name specified in the \texttt{REPORT(\textbackslash{}rparm of SPOOLOPEN (REPORT is the default)}.

The \texttt{"tpsh"} script could be defined as follow (an example):

\begin{verbatim}
#!/bin/ksh
#
# sample spooling shell which
# extract the destination from the spool file
# and forwards it to corresponding ip class
# if destination is not defined assumes $PRINTER as default
#
filename=$1
$temporary\_file=/tmp/spool.$$
$destination=$($\texttt{grep "\textbackslash{}spoolopen/destination\textquoteright\ "$filename | sed 's/.*$/\'} $if ( -z "$destination\textquoteright\ fu
then
 $destination=$\texttt{PRINTER}
 fi
$\texttt{grep -v "\textbackslash{}spoolopen/" < $filename > $temporary\_file
lp -c -d $destination $temporary\_file
rm -f $temporary\_file
\end{verbatim}
3.1.8 Intersystem Communication

XCICS TS supports LU6.2 APPC inter system communication, and provides:

- CICS APIs, the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands that allow the user to code its own Distributed Transaction Processing (DTP) application
- Remote function shipping services:
  - DPL - Distributed Program Link
  - Remote Function Shipping
  - Asynchronous Processing
  - Transaction Routing (TOR & AOR)

**Supported protocols**

XCICS/TS supports LU6.2 APPC communication on two protocols:

- SNA
- TCP/IP

APPC over SNA protocol allows communication between XCICS and any other system supporting the SNA LU6.2 communication protocol, such IBM CICS/TS or another XCICS/TS region.

Communication over the SNA protocol requires a third-party product, hardware dependant, providing the SNA transport layer (SNA over IP, SDLC etc.) and APIs. These products are described in the next chapter.

APPC over TCP/IP protocol is an XCICS native protocol, and allows the communication only between XCICS regions. It does not require any additional software.

**Note:** APPC over TCP/IP is not SNA over IP. Therefore if you need to communicate with SNA over IP (Enterprise Extender) you have to use APPC over SNA: the communication product will provide the SNA transport over the IP protocol.

**Software requirements**

The communication over the SNA protocols is carried out by means of third-party products, that provide the UNIX/Linux system with SNA functionalities. This means than a SNA server product must be installed on the system.

Actually XCICS/TS supports the SNAP-IX(tm) products family:

- DataConnection SNAP-IX for Solaris
- IBM Communication Server for AIX
- IBM Communication Server for Linux
- HP SNAPlus 2 for HP-UX
3.1.8.1 Configuration for SNA

The setup and configuration of the inter system communication services over the SNA protocol, requires the following steps:

1. SNA server product installation and configuration
2. XFRAME configuration and relink
3. SNA Node configuration and LU definition
4. XCICS region configuration
5. Partner region configuration

**SNA server product setup**

First of all, the SNA server product (DataConnection SNAP-IX, IBM Comm. Serv. or HP SNAPplus2), must be installed on the system and correctly configured.

Some flavours of these products have the possibility to select the components to install: It is mandatory to install the core and APPC API components.

Please refer to your SNA server product documentation.

**XFRAME setup**

When the SNA server products is installed, XFRAME APPC components must be built with the product libraries.

To do that, simply (re)install XFRAME, and specify the SNA product type during configuration phase.

**SNA node configuration**

SNA node must be configured and an active link to the partner system must be created.

Of course, this link requires a correspondent definition in the VTAM of the partner system.

Please refer to your SNA server product documentation to configure the node and the link, and refer to the corresponding documentation to define the node in the partner system (VTAM documentation for mainframes and SNA server product for other UNIX systems).

Furthermore at least one LU to be used by the XCICS region must be defined.

This LU should normally have the same name as the XCICS application name. XCICS identifies the LU by its alias therefore remember to correctly defined the alias too. The easiest way to configure the LU is to provide the same 8 byte identifier to the LU, to its alias and to the XCICS application name.

Finally an alias identifying the remote partner must be added.

**XCICS configuration**

First of all, the LU alias for incoming and outgoing connections must be declared in xcics.conf. By default XCICS uses an LU alias equal to its name. Alternatively it is possible to override this behaviour specifying the `default_inbound_lu` and `default_outbound_lu` in xcics.conf.

Then one or more connection entries (CCT) must defined using the `define connection` directive. A connection identifies the partner system, specifying the SYSID used in the application and the remote LU alias.

Finally, sessions to serve incoming requestes must be defined, using the `define sessions` directive.
A sample configuration

Following some sample configuration describing a connection between an XCICS TS region and an IBM CICS TS on SNA over ethernet.

Unix side

in the UNIX/Linux SNA server product configuration file:

```
[define_node]
 cp_alias = UNIX01
 description = ""
 fgcp_name = MINNET:UNIX01
 node_type = LEN_NODE
 mode_to_coe_map_supp = NO
 mds_supported = YES
 node_id = <05F00002>

 # (3) as defined in the VTAM
 V|H|I|D
 max_locates = 1500
 dir_cache_size = 255
 max_dir_entries = 0
 locate_timeout = 0
 reg_with_nn = YES
 reg_with_cds = YES
 mds_send_alert_q_size = 100
 cos_cache_size = 24
 tree_cache_size = 40
 tree_cache_use_limit = 40
 max_tdm_nodes = 0
 max_tdm_tgs = 0
 max_isr_sessions = 1000
 isr_sessions_upper_threshold = 900
 isr_sessions_lower_threshold = 800
 isr_max_ru_size = 16384
 isr_rvr_pmo_window = 8
 store_endpt_rscvs = NO
 store_isr_rscvs = NO
 store_dlr_rscvs = NO
 dlr_support = NO
 pu_conc_support = YES
 nn_rar = 128
 max_ls_exception_events = 0
 ptf_flags = NONE
 cplu_syncpt_support = NO

[define_local_lu]
 lu_alias = XCMS001
 description = ""
 application name
 list_name = ""
 lu_name = XCMS001
 definition
 lu_session_limit = 0
 p1_name = <0000000000000000>
 nul_address = 0
 default_pool = NO
 syncpt_support = YES
 lu_attributes = NONE
 sscp_id = 0
 disable = NO

# (1) the same as local XCICS

# (3) the same as in remote VTAM
```
sys_name = ""
timeout = 60
back_level = NO

[define_partner lu]
plu_alias = IMBCICS                        # (2) partner lu alias
description = ""
frplu_name = MNET.IBMICCS1                  # remote CICS LU name
plu_un_name = DBDCCICS
parallel_sess_supp = YES
max_mc_ll_send_size = 0
conv_security_ver = NO

in xcics.conf:

set application_name= XICCS001 ;                # (1) the same as local LU alias
define connection sysid=P390,
type=sna,
netname=IMBCICS ,                         # (2) partner lu alias
mode=LU62APPC,
acquire, os=ebcdic;
define sessions size=4;                     # number parallel sessions available

Mainframe side

LU definition in the partner VTAM for UNIX system

**XCASNET VBUILD TYPE=SWNET**
HPUX02  PU  PUTYPE=2, ADDR=C1,
       IDBLK=0SF, INNM= 0000Z ,
       MODTAB=ISTINCLM,
       DISCNT=NO,
       MAXDATA=265, MAXOUT=7, MAXPAXT=1, SSCPMT=FSS,
       PACING=0, VPACING=0, ISTATUS=ACTIVE,
       CONNTYPE=APPN, CFCP=YES
HP02   PATH  GRPMM=XGE401E,
       DIALNO=01400306EF3C2FB,
       GID=1, PID=1, USE=NO
*/ (3) the following is the same XCICS LU name
XICCS001 LU  LOCADDR=0

CICS CEDA definitions :

```
DEFINE CONNECTION (XCX1) GROUP (XICCS001) NETNAME (XICCS001) ACCESSMETHOD (VTAM) PROTOCOL (APPC) SINGLESESS (NO)
DEFINE SESSION (XCXLSESSION) GROUP (XICCS001) CONNECTION (XCX1) ACCESSMETHOD (VTAM) PROTOCOL (APPC) MAXIMUM (10,6)
```
3.1.8.2 Configuration for TCP/IP

The setup and configuration of the inter system communication services over the SNA protocol, requires no special setup. You simply have to configure the two XCICS regions involved in the communication, defining for both:

- an ISC oriented TCP servers
- a connection describing the partner

A sample

Configuration of the region called CICSLX01:

```plaintext
set application_name=CICSLX01;
set local_sysid=LX01;
#
# TCP server for incoming ISC request
#
add tcpserver name=APPCSVC, port=8094, isc;
#
# connection to the CICSLNX1 system
#
add tcpserver name=APPCSVC, port=8094, isc;
declare connection sysid=SUN1, type=tcp, netname=CICSSUN1,
    hostname=192.168.1.30, port=8094;
```

Configuration of the region called CICSSUN1:

```plaintext
set application_name=CICSSUN1;
set local_sysid=SUN1;
#
# TCP server for incoming ISC request
#
add tcpserver name=APPCSVC, port=8094, isc;
#
# connection to the CICSLNX1 system
#
declare connection sysid=LNX1, type=tcp, netname=CICSLX01,
    hostname=192.168.1.14, port=8094;
```
3.1.8.3 Distributed Transaction Processing

When XCICS arranges function shipping, distributed program link (DPL), asynchronous transaction processing, or transaction routing for you, it establishes a logical data link with a remote system. A data exchange between the two systems then follows. This data exchange is controlled by CICS-supplied programs, using APPC, LUTYPES 1, or MRO protocols.

The XCICS-supplied programs issue commands to allocate conversations, and send and receive data between the systems. Equivalent commands are available to application programs, to allow applications to converse. The technique of distributing the functions of a transaction over several transaction programs within a network is called distributed transaction processing (DTP).

In the intercommunication facilities, DTP is the most flexible and the most powerful, but it is also the most complex. This section introduces you to the basic concepts.

Why to use DTP

In a multisystem environment, data transfers between systems are necessary because end users need access to remote resources. In managing these resources, network resources are used. But performance suffers if the network is used excessively. There is therefore a performance gain if application design is oriented toward doing the processing associated with a resource in the resource-owning region.

DTP lets you process data at the point where it arises, instead of overworking network resources by assembling it at a central processing point.

There are, of course, other reasons for using DTP. DTP does the following:

- Allows some measure of parallel processing to shorten response times
- Provides a common interface to a transaction that is to be attached by several different transactions
- Enables communication with applications running on other systems, particularly on non-CICS systems
- Enables batching of less urgent data destined for a remote system.

DTP Programming

In DTP, transactions pass data to each other directly. While one sends, the other receives. The exchange of data between two transactions is called a conversation. Although several transactions can be involved in a single distributed process, communication between them breaks down into a number of self-contained conversations between pairs. Each such conversation uses a CICS resource known as a session.

Conversation initiations

A transaction starts a conversation by requesting the use of a session to a remote system. Having obtained the session, it causes an attach request to be sent to the other system to activate the transaction that is to be the conversation partner.

A transaction can initiate any number of other transactions, and hence, conversations. In a complex process, a distinct hierarchy emerges, with the terminal-initiated transaction at the very top.

The structure of a distributed process is determined dynamically by program; it cannot be predefined. Notice that, for every transaction, there is only one inbound attach request, but there can be any number of outbound attach requests.

The session that activates a transaction is called its principal facility. A session that is allocated by a transaction to activate another transaction is called its alternate facility. Therefore, a transaction can have only one principal facility, but any number of alternate facilities.

When a transaction initiates a conversation, it is the front end on that conversation. Its conversation partner is the back end on the same conversation. (Some books refer to the front end as the initiator and the back end as the recipient.) It is normally the front end that dominates, and determines the way the conversation goes. You can arrange for the back end to take over if you want, but, in a complex process, this can cause unnecessary complication. This is further explained in the discussion on synchronization later in this section.

A conversation transfers data from one transaction to another. For this to function properly, each transaction must know what the other intends. It would be nonsensical for the front end to send data if all the back end wants to do is print out the weekly sales report. It is therefore necessary to design, code, and test front end and back end as one software unit. The
same applies when there are several conversations and several transaction programs. Each new conversation adds to the complexity of the overall design.

**Conversation state and error detection**

As a conversation progresses, it moves from one state to another within both conversing transactions. The conversation state determines the commands that may be issued. For example, it is no use trying to send or receive data if there is no session linking the front end to the back end. Similarly, if the back end signals end of conversation, the front end cannot receive any more data on the conversation.

Either end of the conversation can cause a change of state, usually by issuing a particular command from a particular state. XCIICS tracks these changes, and stops transactions from issuing the wrong command in the wrong state.

**Synchronization**

There are many things that can go wrong during the running of a transaction. The conversation protocol helps you to recover from errors and ensures that the two sides remain in step with each other. This use of the protocol is called synchronization.

Synchronization allows you to protect resources such as transient data queues and files. If anything goes wrong during the running of a transaction, the associated resources should not be left in an inconsistent state.

**APPC sync levels**

The APPC architecture defines three levels of synchronization (called sync levels):

- Level 0 - none
- Level 1 - confirm
- Level 2 - syncpoint

All these levels are supported by XCIICS/TS.

At sync level 0, there is no system support for synchronization. It is nevertheless possible to achieve some degree of synchronization through the interchange of data, using the SEND and RECEIVE commands.

If you select sync level 1, you can use special commands for communication between the two conversation partners. One transaction can confirm the continued presence and readiness of the other. The user is responsible for preserving the data integrity of recoverable resources.

The level of synchronization described earlier in this section corresponds to sync level 2. Here, system support is available for maintaining the data integrity of recoverable resources.

XCIICS implies a syncpoint when it starts a transaction; that is, it initiates logging of changes to recoverable resources, but no control flows take place. XCIICS takes a full syncpoint when a transaction is normally terminated. Transaction abend causes rollback. The transactions themselves can initiate syncpoint or rollback requests. However, a syncpoint or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and if sync level 2 has been selected for the conversation between them.

Remember that syncpoint and rollback are not peculiar to any one conversation within a transaction. They are propagated on every sync level 2 conversation that is currently in bracket.

**SYNCPOINT transmission**

XCIICS/TS supports three different modes of SYNCPOINT transmission over APPC:

**simple**

SYNCPOINT is activated with a two-phases only PSH exchange:

- REQUEST COMMIT
- COMMITTED

**implied_forget**

SYNCPOINT is managed with a tree-phases PSH handshake, using implied forget algorithm:

- PREPARE
- REQUEST COMMIT
- COMMITTED

Implied processing is the default.

**explicit_forget**

SYNCPOINT is managed with a four-phases PSH handshake:

- PREPARE
- REQUEST COMMIT
- COMMITTED
- FORGET

The type of SYNCPOINT processing may be defined for each single connection at configuration time or modified at runtime using xcicsadm. If syncpoint processing is not configured in the connection definition, XCICS/TS handles syncpoint with the "implicit" mechanisms by default.

I.e.

```
define connection sysid=P390,
    type=sna,
    netname=IBMCICS1,
    mode=LU62APPB,
    syncpoint=explicit,
    acquire, os=ebcdic;
```

Or runtime:

```
# xcicsadm --set-synctype-implied P390
```

**XLN processing**

XCICS/TS supports the exchange log name (XLN) processing: during the connection acquisition phase it exchanges data with the partner to identify itself and to get information about the capabilities and the status of the remote system.

The "acquire" processing may be activated from XCICS/TS (either automatically at region startup or manually using xcicsadm utility) or by the remote system.

**Known limitations**

XCICS only supports APPC links. No MRO link is possible with remote regions.

XCICS only supports LU6.2 communication.

XCICS only supports mapped APPC. EXEC CICS GDS commands are not currently available.
Other docs

XCICS uses the same APPC APIs of IBM CICS.

For further information about APPC programming, please refer to the following docs:

- IBM CICS Intercommunication Guide
- IBM CICS Distributed Transaction Programming Guide
3.1.8.4 Distributed Program Link

XCICS distributed program link enables XCICS application programs to run programs residing in other XCICS or CICS regions by shipping program-control LINK requests.

An application can be written without regard for the location of the requested programs; it simply uses program-control LINK commands in the usual way. Typically, entries in the XCICS program definition tables specify that the named program is not in the local region (known as the client region) but in a remote region (known as the server region).

XCICS can act as both client and server region for XCICS and IBM CICS systems.

The XCICS recovery and restart facilities enable resources in remote regions to be updated, and ensure that when the client program reaches a syncpoint, any mirror transactions that are updating protected resources also take a syncpoint, so that changes to protected resources in remote and local systems are consistent.

A client program can run in a XCICS intercommunication environment and use DPL without being aware of the location of the server program.

**Static and dynamic routing**

The location of the server region can be defined statically or dynamically.

Static routing means that the location of the server program is specified at design time, rather than at run-time. DPL requests for a particular remote program are always routed to the same server region. Typically, when static routing is used, the location of the server program is specified in the installed program resource definition.

Dynamic routing means that the name of the server region is specified at run-time, using the SYSID parameter.

**Limitations of DPL server programs**

A DPL server program cannot issue the following kinds of commands:

- Terminal-control commands referring to its principal facility
- Commands that set or inquire on terminal attributes
- BMS commands
- Signon and signoff commands
- Batch data interchange commands
- Commands addressing the TCTUA
- Syncpoint commands (except when the client program specifies the SYNONRETURN option on the LINK request).

If the client specifies SYNONRETURN:

- The server program can issue syncpoint requests.
- The mirror transaction requests a syncpoint when the server program completes processing.

Attention: Both these kinds of syncpoint commit only the work done by the server program. In applications where both the client program and the server program update recoverable resources, they could cause data-integrity problems if the client program fails after issuing the LINK request.
3.1.8.5 Function Shipping

XCICS function shipping enables XCICS (command-level) application programs to:

- Access on-line files owned by other XCICS/CICS regions by shipping file control requests.
- Transfer data to or from transient-data and temporary-storage queues in other XCICS/CICS regions by shipping requests for transient-data and temporary-storage functions.
- Initiate transactions in other XCICS/CICS regions (see Asynchronous processing section)

Applications can be written without regard for the location of the requested resources; they simply use file control commands, temporary-storage commands, and other functions in the same way. Entries in the XCICS resource definition tables allow the system programmer to specify that the named resource is not on the local (or requesting) system but on a remote (or owning) system.

The XCICS recovery and restart facilities enable resources in remote systems to be updated, and ensure that when the requesting application program reaches a synchronization point, any mirror transactions that are updating protected resources also take a synchronization point, so that changes to protected resources in remote and local systems are consistent.

Design considerations

User application programs can run in a XCICS intercommunication environment and use the intercommunication facilities without being aware of the location of the file or other resource being accessed. The location of the resource is specified in the resource definition.

The resource definition can also specify the name of the resource as it is known on the remote system, if it is different from the name by which it is known locally. When the resource is requested by its local name, XCICS substitutes the remote name before sending the request. This facility is useful when a particular resource exists with the same name on more than one system but contains data peculiar to the system on which it is located.

Although this may limit program independence, application programs can also name remote systems explicitly on commands that can be function-shipped, by using the SYSID option. If this option is specified, the request is routed directly to the named system, and the resource definition tables on the local system are not used. The local system can be specified in the SYSID option, so that the decision whether to access a local resource or a remote one can be taken at execution time.

Remote File Access

Function shipping allows access to XVSAM/VSAM files located on a remote XCICS/CICS system. INQUIRE FILE, INQUIRE DSNAME, SET FILE, and SET DSNAME are not supported. Both read-only and update requests are allowed, and the files can be defined as protected in the system on which they reside. Updates to remote protected files are not committed until the application program issues a syncpoint request or terminates successfully. Linked updates of local and remote files can be performed within the same unit of work, even if the remote files are located on more than one connected CICS system.

RFA is only supported for KSDS files. No support is provided for RRN and RBA accesses.

Temporary storage

Function shipping enables application programs to send data to, or retrieve data from, temporary-storage queues located on remote systems. A temporary-storage queue is specified as being remote by specifying the SYSID on the access APIs.

Transient data

An application program can access transient-data queues on remote systems. The definition of the queue in the requesting system defines it as being on the remote system. The definition of the queue in the remote system specifies its recoverability attributes, and whether it has a trigger level and associated terminal. Extrapartition queues can be defined (in the owning system) as having records of fixed or variable length.
If a transient-data queue has an associated trigger level transaction, the named transaction must be defined to execute in the system owning the queue; it cannot be defined as remote.
3.1.8.6 Asynchronous Processing

Asynchronous processing provides a means of distributing the processing that is required by an application between systems in an intercommunication environment. Unlike distributed transaction processing, however, the processing is asynchronous.

In distributed transaction processing, a session is held by two transactions for the period of a "conversation" between them, and requests and replies can be directly correlated.

In asynchronous processing, the processing is independent of the sessions on which requests are sent and replies are received. No direct correlation can be made between a request and a reply, and no assumptions can be made about the timing of the reply.

In general, asynchronous processing is applicable to any situation in which it is not necessary or desirable to tie up local resources while a remote request is being processed.

Asynchronous processing is not suitable for applications that involve synchronized changes to local and remote resources; for example, it cannot be used to process simultaneous linked updates to data split between two systems.

Asynchronous processing methods

In XCICS, asynchronous processing can be done in either of two ways:

1. By using the interval control commands START and RETRIEVE.

   You can use the START command to schedule a transaction in a remote system in much the same way as you would in a single XCICS system. This type of asynchronous processing is in effect a form of XCICS function shipping, and as such, it is transparent to the application. The systems programmer determines whether the attached transaction is local or remote.

   If you use the START command for asynchronous processing, you can communicate only with systems that support the special protocol needed for function shipping; that is, XCICS and CICS.

   An XCICS transaction that is initiated by a remotely-issued start request can use the RETRIEVE command to retrieve any data associated with the request. Data transfer is restricted to a single record passing from the initiating transaction to the transaction initiated.

2. By using distributed transaction processing (DTP).

   This is a cross-system method and has no single-system equivalent. You can use it to initiate a transaction in a remote system that supports one of the DTP protocols.

   When you use DTP to attach a remote transaction, you also allocate a session and start a conversation. This permits you to send data directly and, if you want, to receive data from the remote transaction. Your transaction design determines the format and volume of the data you exchange. For example, you can use repeated SEND commands to pass multirecord files.

   When you have exchanged data, you terminate the conversation and quit the local transaction, leaving the remote transaction to run on independently.

Whatever protocol you decide to use, you must observe the rules it imposes. However short the conversation, during the time it is in progress, the processing is synchronous. In terms of command sequencing, error recovery, and syncpointing, it is full DTP.

In both forms of asynchronous processing (and also in synchronous processing), an XCICS transaction can use the EXEC CICS ASSIGN STARTCODE command to determine how it was initiated.

Asynchronous processing programming

The interval control commands that can be used for asynchronous processing are:

- START
- RETRIEVE

For further information, please refer to the XCICS Programming Guide.
3.1.8.7 Dynamic Conversion System

Whenever XCICS perform a function shipping with a CICS remote region running on a mainframe, the EBCDIC-ASCII conversion problem arises. To bypass this inconvenient, XCICS provides a On-line Data Conversion System to transform properly the data exchanged with the remote region.

Generally, XCICS allows the definition of a conversion routine for each resource accessed on or by a remote CICS region.

For further information, please refer to the On-line Data Conversion section of the XCICS Customization Guide.
**3.1.8.8 Transaction Routing**

XCICS transaction routing allows terminals connected to one XCICS/CICS system to run with transactions in another connected XCICS/CICS system. This means that you can distribute terminals and transactions around your XCICS/CICS systems and still have the ability to run any transaction with any terminal.

The system that owns the terminal is called the terminal-owning region or TOR, and the system that owns the transaction is called the application-owning region or AOR. These terms are not meant to imply that one system owns all the terminals and the other system all the transactions, although this is a possible configuration.

The terminal-owning region and the application-owning region must be by APPC links.

The user transaction can use the terminal control, BMS, of XCICS to communicate with the terminal, as appropriate for the terminal or session type. Mapping and data interchange functions are performed in the application-owning region.

The transaction routing feature can be successfully employed in the following scenarios.

**Multiple server systems (Cluster)**

The architecture of XCICS does not require the creation of a pool of regions to make use of all the power provided by a single SMP computer, because every XCICS region can use effectively more than one CPU.

In some situation, you want to use the power of more than system: this is the case of a cluster. In this scenario one ore more XCICS regions runs on every machine of the cluster, each one serving a part of the application. The terminal users will connect to one region (TOR), which will route the transaction requested to the corresponding server.

**Partial or progressive re-hosting**

In the case of a partial re-hosting of a mainframe system (only a part of the applications running on the mainframe are move to UNIX/Linux) or in the case of a progressive re-hosting (applications are moved one next the other to the UNIX/ Linux system), the transaction routing can be successfully used.

In this situations XCICS normally acts as AOR, serving transactions for the TOR (the old mainframe application), reducing the work-load of the mainframe system.

Of course a reverse scenario could also be possible, using XCICS as TOR and IBM CICS as AOR.

**3.1.8.8.1 Configuration**

To make use of transaction routing, both TOR and AOR must be properly configured.

In the TOR:

- a working connection (CCT entry) to the AOR
- transactions to be run on the AOR must be configured as remote providing the SYSID of AOR and, eventually, the remote name

In the AOR:

- a working connection (CCT entry) and serving sessions for the TOR
- terminal entries for the TOR, defined as remote

**Important note**

Terminal entries used to route transactions (both local and remote) must be defined with a TCTUA of 255 bytes.
3.1.8.8.2 Configuration samples

This section describes some sample configurations in different scenarios, where XCICS/TS or IBM CICS may act as TOR (Terminal Oriented Region) and/or AOR (Application Oriented Region).

**IBM CICS TOR to XCICS AOR**

This sample describes the case of an IBM CICS region acting as TOR, routing transaction to an XCICS/TS region acting as AOR. In this scenario users will connect to the IBM CICS region (TOR), while the transactions will be remotely executed on the XCICS/TS region (AOR).

**CICS definitions:**

```
DEFINE CONNECTION(SUN1)  GR(XCICS001) NETNAME(XCICS001) ACCESSMETHOD(VTAM)
  PROTOCOL(APPC) SINGLESESS(NO)
DEFINE SESSION(XCICS001) CONNECTION(SUN1) ACCESSMETHOD(VTAM) PROTOCOL
  (APPC) MAXIMUM(10,6)
DEFINE TERMINAL(900) GR(MYGRP) TYPETERM(MYTYPTRM) ......
DEFINE TYPETERM(MYTYPTRM) GR(MYGRP) USERNAMES(255) ...
DEFINE TRANSACTION(RSUN) GR(XCICS001) REMOTESYSTEM(SUN1)
```

**XCICS definitions:**

```
define connection sysid=P390, type=enq, netname=CICS, mode=LU6.2APPB, acquire,
  os=ebcdic;
define session size=4;
define terminal name=L900, type=remote;
define transaction code=RSUN, program=ROUTEST, twa=512, protection=64;
```

**XCICS TOR to IBM CICS AOR**

This sample describes the case of an XCICS/TS region acting as TOR, routing transaction to an IBM CICS region acting as AOR. In this scenario users will connect to the XCICS/TS region (TOR), while the transactions will be remotely executed on the IBM CICS region (AOR).

**CICS definitions:**

```
DEFINE CONNECTION(SUN1)  GR(XCICS001) NETNAME(XCICS001) ACCESSMETHOD(VTAM)
  PROTOCOL(APPC) SINGLESESS(NO)
DEFINE SESSION(XCICS001) CONNECTION(SUN1) ACCESSMETHOD(VTAM) PROTOCOL
  (APPC) MAXIMUM(10,6)
DEFINE TERMINAL(TROU) GR(MYGRP) TYPETERM(MYTYPTRM) NETRAN(TROU)...
DEFINE TYPETERM(MYTYPTRM) GR(MYGRP) USERNAMES(255) REMOTENAME(TROU)
  REMOTESYSTEM(XCICS001)
DEFINE TRANSACTION(RIBM) GR(XCICS001) PROGRAM(RIBMPGM) ...
```

**XCICS definitions:**

```
define connection sysid=P390, type=enq, netname=CICS, mode=LU6.2APPB, acquire,
  os=ebcdic;
define session size=4;
```
XCICS TOR to XCICS AOR over SNA

XCICS TOR definitions:

```plaintext
define connection sysid=HP02, type=sna, netname=CICSHP02, mode=LU62APPB, acquire, os=ascii;
define session size=4;
define terminal name=TROU, type=standard, physical=TROU;
define transaction code=RHP2, sysid=SUN1, remote_name=RHP2, twa=512, protection=64;
```

XCICS AOR definitions:

```plaintext
define connection sysid=SUN1, type=sna, netname=CICSSUN1, mode=LU62APPB, acquire, os=ascii;
define session size=4;
define terminal name=TROU, type=remote;
define transaction code=RSUN, program=ROUTETST, twa=512, protection=64;
```

XCICS TOR to XCICS AOR over TCP/IP

XCICS TOR definitions:

```plaintext
define connection sysid=SU1T, type=tcp, netname=CICSSUN1, hostname=solaris01, port=8094;
add tcpserver name=APPCSVC, port=8094, terminal=TISC, isc;
define terminal name=TROU, type=standard, physical=TROU;
define transaction code=RLX1, sysid=LNX1, remote_name=RLX1, twa=512, protection=64;
```

XCICS AOR definitions:

```plaintext
define connection sysid=LNX1, type=tcp, netname=CICSLX01, hostname=linux01, port=8094;
add tcpserver name=APPCSVC, port=8094, terminal=TISC, isc;
define terminal name=TROU, type=remote;
define transaction code=RLX1, program=ROUTETST, twa=512, protection=64;
```
3.1.8.9 ISSUE PASS command

The EXEC CICS ISSUE PASS command can be used to disconnect a terminal from an XCICS region and transfer it to another XCICS application specified on the LUNAME option.

For example, to transfer a terminal from this XCICS to another terminal-owning region, you could issue the command:

```
EXEC CICS ISSUE PASS
   LUNAME( applid)
```

where `applid` is the 8 byte identifier of an application connection to which the terminal is to be transferred.

The application connection must be defined with the specified `applid` in:

- X4J sessions list
- XTND configuration file

Additionally, the ISSUE PASS can forward a data area to the other application, which can be extracted using the EXTRACT LOGONMSG command.

Please note that only XCICS to XCICS pass is available. No link with IBM CICS and/or IBM VTAM can be performed.
3.1.9 TCP/IP services

XCICS TS may be accessed from external applications, by means of TCP/IP based services. These services allows to remote application to connect and exchange data with an XCICS/TS region over TCP/IP.

Four different communication protocols are currently supported by XCICS TCP Services:

- Classic EZASOKET Listener
- Direct link EZASOKET Listener
- Web Services Connector
- ECI
- HTTP
- User

The first two protocols (classic and direct) emulates the behaviour of the IBM EZASOKET listener, delivered with IBM CICS.

The Web Services Connector allows to activate CICS programs through from every web services enabled application, the SOAP protocol.

The ECI mode, allow to serve remote application which communicates with XCICS using the XECI interface.

HTTP allows to XCICS to act as an HTTP server and make user of the XCICS Web Support interface. User protocols are an extension to the Web Interface and allow to the user to exchange non-HTTP requests. For further information about XCICS Web Interface, HTTP and user protocols, please refer to the XCICS Internet Guide.
3.1.9.1 TCP/IP listeners

Client applications can communicate with a remote XCICS region using sockets, either via direct socket API calls and EZASOKET APIs.

Users may write their own socket listener, or take advantage of the listeners supplied by XCICS/TS: They emulate the behaviour of the IBM EZASOKET Listener provided by IBM and may be used to activate and communicate with a transaction from a remote application.

In order to enable activate the predefined socket listeners, a TCP service definition must be added to the XCICS/TS configuration file (see TCP/IP services. XCICS/TS will provide to automatically start up the defined TCP service(s). Each TCP service instance will be listening on the pre-defined port for incoming requestes, in order to forward them to the transaction server, where requests will be serviced.

The work mode of a TCP listener is specified in its configuration in the XCICS configuration file. The Direct Link mode is the default.

i.e.

```
add tcpipservice name=DEFAULT, port=8888, terminal=TCPD, direct, use_engines;
add tcpipservice name=IBMEZ, port=8099, terminal=TCP, classic;
```

For further information on how to use the predefined socket listeners please refer to the TCP/IP Sockets Programming Guide.
3.1.9.2 Web Services (SOAP)

XCICS Web Services area a set of APIs to deploy XCICS functionalities to an heterogenous network of clients. In fact, by using the SOAP protocol, XCICS Web Services are available for clients on platforms like J2EE, .NET, CORBA, etc.

XCICS Web Services make use of the powerful AXIS framework, the Apache Jakarta implementation, of SOAP protocol. This ensure stability, power and portability.

**Functionalities**

By using the XCICS Web Services, remote clients may easily access to the following XCICS functionalities:

- LINK PROGRAM
- START TRANSID
- WRITEQ TD
- READQ/WRITSEQ TS

To get a detailed description on APIs, please refer to the XJE Java APIs for the class com.hite.connectors.XCicsConnector.

### 3.1.9.2.1 Installation

Some software and preliminary activities are requested to install XCICS Web Services Connector.

**Requirements**

THE XCICS Web Services framework requires:

- Apache Jakarta Tomcat 4.1.29 or higher
- Apache Jakarta AXIS 1.1

ANT building tool is suggested too.

**Installation**

**Preliminary tasks**

Before to install and configure the XCICS Web Services framework, both Tomcat and AXIS must be correctly installed and configured. Please refer to their documentation on their respective web sites. Only when AXIS is correctly installed and tested, you may proceed with the XCICS Web Services framework setup.

Please do not forget to define and export an environment variable named AXIS_HOME, that paths to the AXIS installation directory.

**Installing the framework**

All the services provided by the XCICS Web Services framework, are made available through the deployment of the class named com.hite.connectors.XCicsConnector.

System administrator should have a small background on administration and deployment/undeployment of web services with AXIS. In the same manner, to make use of deployed services, programmer must have knowledge of AXIS programming.

To speed-up services deployment and use, some sample files are provided in XFRAME distribution. In the directory $XFRAMEHOME/jxe/samples/webservices the following files are available:
Follow the procedure below, to set up the XCICS Web Service in the AXIS web app.

**Tomcat & AXIS setup**

First of all, Tomcat must be installed. After that, AXIS must be set up and tested. Default settings are normally enough to set up XCICS Web Services.

Do not forget to export CATALINA_HOME and AXIS_HOME.

**XCICS configuration**

XCICS must be configure to accept Web Services request, from the web server. To do that a "webservice" RCP connector must be declared. In the "xcics.conf" add the following line:

```
add tcpserver name=LISTENER, port=8093, webservice;
```

This will add a TCP Server running in Web Service mode on the port 8093. Please refer to TCP Server docs for more information about it.

**Deployer configuration**

First of all, deployment system must be configured. Edit the file build.xml and change the "editable properties" if needed.

By default the build.xml provided with XFRAME, supposes that the tomcat webserver containing the AXIS application, is running on localhost at port 8080 and that AXIS has been configured with defaults.

**Web Service configuration**

edit the file deploy.wsdd and change the values

- xcics.ws.host
- xcics.ws.port

proving the hostname and port of the XCICS Web Service server for the XCICS application. In the sample above the TCP Server is running at port 8093.

**Web Service Deployment**

Now, the web service is ready to be deployed. From the directory $XFRAMEHOME/xje/samples/webservices issue the command:

```
ant deploy
```
If the command completes successfully, you should be able to see the XCicsConnector in the list of deployed services, in the AXIS application.

ATTENTION: this step copies some files in the AXIS webapp (usually $CATALINA_HOME/webapps/axis), so be sure to have write permission on the target directory.

### Stub generation

Now it's time to transform the WSDL of deployed service into a Java stub for clients. Issue the command:

```
ant wsdl2java
```

If the command completes successfully, a set of Java sources for the stub interfaces are created in the sub-directory "generated"

### Java samples compiling

In order to compile sample programs, issue:

```
ant compile
```

It will compile 2 test program, one using dynamic invocation and another one using stubs.

### CICS samples compiling

In order to serve samples, a sample server COBOL programs has be provided: WEBSRV.pre. To compile it issue:

```
xcob -ua WEBSRV.pre
```

Do not forget to declare the program in PPT, by ading in xcics.conf:

```
delare program name=WEBSRV;
```

### Running samples

Samples may be run issuing:

```
runc TestStub
runc testClient http://localhost:8080/axis/services/XCicsConnector
```
3.1.9.3 XJGATE APIs

XJGATE is a set of APIs that allow a program to connect XCICS in order to activate transactions and to exchange data with them.

XJGATE APIs are available both for the following programming languages:

- C language
- Java.

The use of the XJGATE APIs is discouraged and maintained only for backward compatibility reasons. Programmers that want to communicate with programs running under XCICS/TS should consider the opportunity to use other communication protocols like SNA ISC, TCP/IP sockets or web services.

For further information on how to program with XJGATE API please refer to the XJGATE APIs Guide.

The gateway transaction

In order to provide an easier gatewaying interface a gateway program has been provided: XJGT.

To activate XJGT, XCICS must be configured with the following entries.

```
define program name=XJGT, language=java, path=com.hite.xjgate.XJGateServer;
define transaction name=XJGT, program=XJGT, twa=512, protection=64;
```
3.1.10 Logging, Accounting & Journaling

XCICS/TS has the capability to track, log and keep accounting of many aspects of the region operations and resources usage, by means of:

- master logging
- accounting
- journaling

Due to compatibility reasons, the journaling is performed according to the IBM CICS standards, while logging and accounting are achieved by proprietary mechanisms, to better fit the Unix/Linux systems standards.
3.1.10. Logging

The XCICS/TS main logging facility is the **console log**, which contains all the information, warning and alert messages coming from both XCICS/TS core and user programs.

This log is normally named *xcicslog.latest* and it is backup on *xcicslog.<datetime>*. Its location, depends on the instance configuration.

It is also possible to modify, enable and disable all messages shown on the console log; by default XCICS TS loads messages from a file called *xcicsmsg.txt*, which is normally located under the directory $XFRAMEHOME/etc.

This file contains the message code and the corresponding text for each message displayed.

To redefine XCICS messages, copy the $XFRAMEHOME/etc/xcicsmsg.txt and edit it. Once done, set the environment XCICS_MESSAGES_FILE to the new file path and restart XCICS.

**Message file syntax**

The syntax for the *xcicsmsg.txt* file is simple: each message is contained in one line and is composed by an output control byte, a message code and a message text.

The first byte on each line identifies the activity status for the message:

- blank message is active either on logs and console
- + message is active only on logs

An asterisk (*) on the first byte means that the line is processed as a comment.

The output control byte is followed by a four characters message code and by the message text, separated by a comma (,). The message text, whose length may be up to 60 characters, can contain formatting sequences too (in the same way as for the printf function) that can be updated at the moment of display with variable data (i.e. a data set name).

User messages codes should be identified by a message code beginning by the 'U' letter, and distinguished from the messages displayed by XCICS, that initiate with a 'C'.

Example.

```
* This is an example * and this line is a comment
C001, XCICS VERSION %s +C001,INITIALIZATION STARTED AT %s
+C002, ERROR OPENING LOGFILE +C003, LOADING TABLE FILE '%s' & Only on logs
+C004, MEMORY ALLOCATION ERROR & Only on logs
C005, TABLE FILE '%s' NOT AVAILABLE
C006, XCICS SHUTDOWN COMPLETED
C007, XCICS APPLICATION '%s' ALREADY RUNNING
#C008, STARTUP OF APPLICATION '%s' IN PROGRESS & Disabled
C009, ALLOCATED SHARED MEMORY SIZE %d KBYTEs
U001, USER TRANSACTION '%s' STARTED WITH PARAMETERS %s & user message
```
### 3.1.10.1.1 XCICS Messages

The following table reports the XCICS messages and their description.

<table>
<thead>
<tr>
<th>Message Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C000</td>
<td>XCICS version display</td>
</tr>
<tr>
<td>C001</td>
<td>Initialization started</td>
</tr>
<tr>
<td>C002</td>
<td>error opening the XVSAM recovery log file</td>
</tr>
<tr>
<td>C003</td>
<td>XCICS is loading the displayed configuration file</td>
</tr>
<tr>
<td>C004</td>
<td>memory allocation error detected</td>
</tr>
<tr>
<td>C005</td>
<td>configuration file cannot be found</td>
</tr>
<tr>
<td>C006</td>
<td>XCICS region shutdown completed normally</td>
</tr>
<tr>
<td>C007</td>
<td>an XCICS region with identical name is already running</td>
</tr>
<tr>
<td>C008</td>
<td>startup of region in progress</td>
</tr>
<tr>
<td>C009</td>
<td>size of shared memory allocated</td>
</tr>
<tr>
<td>C010</td>
<td>shared memory allocation error detected</td>
</tr>
<tr>
<td>C011</td>
<td>display of the number of entry for each configuration table</td>
</tr>
<tr>
<td>C012</td>
<td>display of the size of the internal buffers</td>
</tr>
<tr>
<td>C013</td>
<td>XCICS begins to load mapset data in shared memory</td>
</tr>
<tr>
<td>C014</td>
<td>XCICS successfully completed the mapset data loading</td>
</tr>
<tr>
<td>C015</td>
<td>XCICS is opening the XVSAM files</td>
</tr>
<tr>
<td>C016</td>
<td>specified transaction started by PLT</td>
</tr>
<tr>
<td>C017</td>
<td>XCICS is creating the main message queue</td>
</tr>
<tr>
<td>C018</td>
<td>XCICS is initializing the signal handler subsystem</td>
</tr>
<tr>
<td>C019</td>
<td>XCICS initialization is complete</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>C020</td>
<td>A terminal tried to connect a the device shown, but the device is already occupied by another terminal: the connection is refused</td>
</tr>
<tr>
<td>C021</td>
<td>A server process (XRUN) started</td>
</tr>
<tr>
<td>C022</td>
<td>A server process terminated</td>
</tr>
<tr>
<td>C023</td>
<td>A background server process started</td>
</tr>
<tr>
<td>C024</td>
<td>A background server process terminated</td>
</tr>
<tr>
<td>C025</td>
<td>A server process aborted</td>
</tr>
<tr>
<td>C026</td>
<td>A background server process aborted</td>
</tr>
<tr>
<td>C027</td>
<td>A terminal requested a connection for a device undefined: the connection is refused</td>
</tr>
<tr>
<td>C028</td>
<td>A server process detected a signal (dump): rollback is being issued</td>
</tr>
<tr>
<td>C029</td>
<td>The mapset shown cannot be found in the specified path</td>
</tr>
<tr>
<td>C030</td>
<td>A failure has been detected while opening the file name shown</td>
</tr>
<tr>
<td>C031</td>
<td>A server process detected a severe error while communicating with terminal: connection is closed</td>
</tr>
<tr>
<td>C032</td>
<td>A background task has been started by ATI facilities</td>
</tr>
<tr>
<td>C033</td>
<td>A background task has been started on the specified terminal</td>
</tr>
<tr>
<td>C034</td>
<td>Generic system error</td>
</tr>
<tr>
<td>C035</td>
<td>Operator security violation detected at specified terminal</td>
</tr>
<tr>
<td>C036</td>
<td>The specified map cannot be found in the specified mapset</td>
</tr>
<tr>
<td>C037</td>
<td>Licensing error detected</td>
</tr>
<tr>
<td>C043</td>
<td>The XCICS region is ready to serve</td>
</tr>
<tr>
<td>C044</td>
<td>XCICS region has shut down abnormally</td>
</tr>
<tr>
<td>C045</td>
<td>XCICS region has shut down normally</td>
</tr>
<tr>
<td>C046</td>
<td>A background task is waiting for the availability of specified terminal to start</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>C047</td>
<td>the specified dataset cannot be open due to an ISAM error</td>
</tr>
<tr>
<td>C048</td>
<td>XVSAM remote server core started</td>
</tr>
<tr>
<td>C049</td>
<td>XVSAM remote server child started</td>
</tr>
<tr>
<td>C050</td>
<td>wrong dataset definition (name mismatch)</td>
</tr>
<tr>
<td>C051</td>
<td>the shutdown of the region has been requested by specified terminal</td>
</tr>
<tr>
<td>C052</td>
<td>XVSAM lock timeout detected</td>
</tr>
<tr>
<td>C053</td>
<td>XCICS network server detected an error on socket</td>
</tr>
<tr>
<td>C054</td>
<td>XCICS network server detected an error on socket</td>
</tr>
<tr>
<td>C056</td>
<td>wrong dataset definition (information mismatch)</td>
</tr>
<tr>
<td>C057</td>
<td>the specified dataset is locked by another process: impossible to check</td>
</tr>
<tr>
<td>C058</td>
<td>the specified dataset is locked by another process: impossible to check</td>
</tr>
<tr>
<td>C059</td>
<td>the concurrent connection limit has been reached: connection refused</td>
</tr>
<tr>
<td>C060</td>
<td>region shutdown started</td>
</tr>
<tr>
<td>C061</td>
<td>region shutdown is in progress</td>
</tr>
<tr>
<td>C062</td>
<td>an abend has been detected on the specified terminal/transaction</td>
</tr>
<tr>
<td>C063</td>
<td>database error detected</td>
</tr>
<tr>
<td>C064</td>
<td>system's resource exhausted: fork failed</td>
</tr>
<tr>
<td>C065</td>
<td>specified terminal connected</td>
</tr>
<tr>
<td>C066</td>
<td>specified terminal disconnected</td>
</tr>
<tr>
<td>C067</td>
<td>unknown message for main task discarded</td>
</tr>
<tr>
<td>C068</td>
<td>XTDSTOR dataset has not been defined in the configuration: region startup refused</td>
</tr>
<tr>
<td>C069</td>
<td>specified dynamic link library has been loaded successfully</td>
</tr>
<tr>
<td>C070</td>
<td>specified dynamic link library cannot be loaded</td>
</tr>
<tr>
<td>Code</td>
<td>Message</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>C071</td>
<td>region immediate shutdown started</td>
</tr>
<tr>
<td>C072</td>
<td>region is waiting for the shutdown timeout to close</td>
</tr>
<tr>
<td>C073</td>
<td>the specified TD queue trigger has been set to zero</td>
</tr>
<tr>
<td>C074</td>
<td>the displayed operation has occurred on the transid triggered for the specified queue</td>
</tr>
<tr>
<td>C075</td>
<td>the specified terminal requested a reload of the region server processes (engines)</td>
</tr>
<tr>
<td>C076</td>
<td>engine reload message: the message is auto-explanatory</td>
</tr>
<tr>
<td>C077</td>
<td>specified user changed its password</td>
</tr>
<tr>
<td>C078</td>
<td>the specified terminal requested a reconfiguration of the region</td>
</tr>
<tr>
<td>C079</td>
<td>region reconfiguration aborted</td>
</tr>
<tr>
<td>C080</td>
<td>user shared memory allocated successfully</td>
</tr>
<tr>
<td>C081</td>
<td>user shared memory released</td>
</tr>
<tr>
<td>C082</td>
<td>the current XVSAM recovery log file is shown</td>
</tr>
<tr>
<td>C083</td>
<td>the local SYSID is shown</td>
</tr>
<tr>
<td>C084</td>
<td>specified program cannot be loaded due to size error</td>
</tr>
<tr>
<td>C085</td>
<td>the current licensing options allow the enterprise function set on the region</td>
</tr>
<tr>
<td>C086</td>
<td>the region is requesting the specified amount of shared memory</td>
</tr>
<tr>
<td>C087</td>
<td>generic auto-explanatory message</td>
</tr>
<tr>
<td>C088</td>
<td>specified device is not available: connection refused</td>
</tr>
<tr>
<td>C089</td>
<td>specified service is enabled</td>
</tr>
<tr>
<td>C090</td>
<td>specified service is enabled</td>
</tr>
<tr>
<td>C091</td>
<td>auto-explanatory messages about exit programs</td>
</tr>
<tr>
<td>C092</td>
<td>auto-explanatory messages about LDAP</td>
</tr>
<tr>
<td>C093</td>
<td>auto-explanatory messages about SSL</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>CNFE</td>
<td>auto-explanatory configuration error messages</td>
</tr>
<tr>
<td>CNFW</td>
<td>auto-explanatory configuration warning messages</td>
</tr>
<tr>
<td>C999</td>
<td>auto-explanatory messages</td>
</tr>
<tr>
<td>CKTI</td>
<td>auto-explanatory messages from CKTI</td>
</tr>
<tr>
<td>USER</td>
<td>user message</td>
</tr>
<tr>
<td>XADM</td>
<td>auto-explanatory messages from XADM task (obsolete)</td>
</tr>
<tr>
<td>XTCP</td>
<td>auto-explanatory messages from TCP/IP services</td>
</tr>
<tr>
<td>XSNA</td>
<td>auto-explanatory messages from SNA services</td>
</tr>
<tr>
<td>CSRR</td>
<td>auto-explanatory messages from the transaction routing system</td>
</tr>
<tr>
<td>MIRC</td>
<td>auto-explanatory messages from MIRC</td>
</tr>
<tr>
<td>MIRS</td>
<td>auto-explanatory messages from MIRS</td>
</tr>
<tr>
<td>H2R0</td>
<td>H2R PSB loading started</td>
</tr>
<tr>
<td>H2R1</td>
<td>H2R PSB loading completed, with specified memory usage</td>
</tr>
<tr>
<td>H2R2</td>
<td>H2R detected the loading failure for the specified PSB</td>
</tr>
<tr>
<td>IMSM</td>
<td>auto-explanatory messages from XIMS</td>
</tr>
<tr>
<td>XTND</td>
<td>auto-explanatory messages from XTND</td>
</tr>
</tbody>
</table>
3.1.10. Accounting

XCICS TS furnishes statistical information, to measure the transaction resource usage. In the logs directory, XCICS instance stores a text file for each terminal connected (named `rep_<EIBTRMID>`), containing the track of the transactions performed, including:

- elapsed time
- cpu time
- services utilization

Data stored in these files may be easily analyzed by user programs, or by means of the `xreport` utility.

The `xreport` utility can be started at any time from the command line, and it generates a printable report in the logs directory (named `REPORT.<timestamp>`) and a CSV file (named `REPDBD.<timestamp>`).
3.1.10. Tracing

XCICS is can trace:

- core (main process)
- tasks

Core tracing is useful to check configuration issues or for debugging purpose. Two levels of core tracing are available:

- none: no log is produced
- full: log is produced

Core tracing can be configured at startup time using the set core_trace directive in xcics.conf, or at run-time using xcicsadm utility or CETR transaction.

Task tracing can be used to keep track of all the activities performed by a single terminal in terminal or transaction and, according to the trace level, this trace can be used to perform program debugging.

By means of these logging features, all the call to XCICS performed by user transactions as well as the XCICS internals can be logged on a different file for each terminal. These files are written in the XCICS logs directory, and their name is `deb<EIBTRMID>.<suffix>`. The EIBTRMID part of the name refers to the name of terminal which generated the log; the suffix indicates the connection time for interactive terminals, and process number for background task.

Three different levels are available:

- none: no log is produced
- user: only EXEC CICS commands and parameters are tracked
- full: EXEC CICS commands as well as XCICS internals are tracked.

Terminal logging may be configured at start up time, for the whole application, directly from the XCICS configuration file, using the set core_trace directive, or at run-time using xcicsadm utility or CETR transaction.

Tracing may cause performance lacks, therefore it would be a good practice to start production regions with no tracing active, and directly activate them only if needed.
3.1.10. Journaling

XCICS Transaction Server provides a journaling system similar to the one featured by IBM CICS™.

XCICS Journals are kept into stream files, which organization reflects the one described in the "IBM CICS Customization Guide", but with some differences due to the different operating system.

Journal files

Each journal is contained in a single stream file, containing a record for each journaling call.

The record is composed by two different parts:

- Record Header
- Caller Data

Record Header

The record header contains information that describes some of the attributes of the record, and it has always the same structure.

This is the format of the record header:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLRH_RECORD_LENGTH</td>
<td>4-byte record length</td>
</tr>
<tr>
<td>GLRH_HEADER_LENGTH</td>
<td>4-byte header length</td>
</tr>
<tr>
<td>GLRH_REC_DATA_LEN</td>
<td>4-byte record data length</td>
</tr>
<tr>
<td>GLRH_GMT</td>
<td>8-byte time (GMT)</td>
</tr>
<tr>
<td>GLRH_LOCAL</td>
<td>8-byte time (local)</td>
</tr>
<tr>
<td>GLRH_TRAN_ID</td>
<td>4-byte transaction identifier</td>
</tr>
<tr>
<td>GLRH_TASK_ID</td>
<td>4-byte task identifier</td>
</tr>
<tr>
<td>GLRH_TERM_ID</td>
<td>4-byte terminal identifier</td>
</tr>
<tr>
<td>GLRH_REC_TYPE</td>
<td>2-byte record type. Either:</td>
</tr>
<tr>
<td></td>
<td>`X' 0001'  Start of run record</td>
</tr>
<tr>
<td></td>
<td>`X' 0002'  Any other record</td>
</tr>
<tr>
<td>GLRH_REC_COMPID</td>
<td>2-byte domain identifier</td>
</tr>
<tr>
<td>GLRH_REC_JOURNAL</td>
<td>8-byte journal name</td>
</tr>
</tbody>
</table>

Start of task indicator 1-byte which may contain:

- `X' 8f'  Equivalent to JCSPSOTK (start of task)
- `X' 4f'  Equivalent to JCSPLSTK (start of UOW)

Caller Data

The caller data contains the information being tracked, and it differs depending on the XCICS component issuing the record, and on the function being journaled. Two record types are present:

- Start-of-run records
- General Caller Data

Format of Start-of-run records

When XCICS opens a journaling log, it writes a start-of-run record to it as the first record for this run of XCICS. This record comprises a record header (with the same format as that for any general log journal record) followed by a start-of-run body.

This is the format of Start-of-run record:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOR_CICS_RELEASE</td>
<td>4-byte CICS release</td>
</tr>
<tr>
<td>SOR_SPECIFIC_APPLID</td>
<td>8-byte CICS applid</td>
</tr>
<tr>
<td>SOR_CICS_USERNAME</td>
<td>8-byte CICS username</td>
</tr>
</tbody>
</table>

Format of caller data
Caller data follows the record header.

The format of the caller data part of a general log journal record differs according to the XCICS component writing the record, and the function being journaled.

Journal records can be written by any of the following XCICS components:

- journal control (in the case of a request issued by a user)
- file control (actually not yet supported)
- terminal control

The field GLRH_REC_COMPID in the record header tells you which component has written the record: UJ, FC, or TC respectively.

This is the structure of a caller data record:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_UH_LENGTH</td>
<td>4-byte header length.</td>
</tr>
<tr>
<td>CL_UH_JOURNAL_TYPE</td>
<td>2-byte journal type.</td>
</tr>
<tr>
<td>Reserved</td>
<td>2-byte reserved field.</td>
</tr>
<tr>
<td>CL_UH_PREFIX_LENGTH</td>
<td>4-byte prefix length.</td>
</tr>
<tr>
<td>Prefix</td>
<td>Variable-length prefix area.</td>
</tr>
<tr>
<td>User data</td>
<td>Variable-length user data.</td>
</tr>
</tbody>
</table>

3.1.10.4.1 Command line utilities

An XCICS journal log can be easily dumped using the `jrnlfmt` utility, which provides an user friendly interface to view the content of journal log file.

When issued against a log file, it produces on stdout an human-readable visualization of the file.

**Syntax**

```
jrnlfmt [options] <logfile>
```

**Options**

- `-l` long format
- `-x` hexadecimal display of caller data
3.1.11Customization Guide

Some aspects of the region behaviour may be customized using user written programs, also known as "exit programs". Exit programs may be used to control:

- Online Dynamic Conversion System
- SNT (Signon Table) loading
- Users authentication
- AID management

Exit programs may be coded in every language, even if C is the best language for this kind of programs.
3.1.11. Exit programs

Some aspects of the configuration as well as some run-time behaviour of XCICS/TS may be influenced and/or driven by user exit programs.

XCICS/TS supports the following type of exit programs:

- odcs_ts
- snt_loader
- user_signon
- user_signoff
- user_chpasswd
- dfhaid

Exit programs are normally coded by the user in C language routine, compiled and linked in a shared library and declared at configuration time.

General coding rules

The exit programs are normally coded in C language and their entry point must respect the function prototype described in program specific section.

The source program must include the file "cics.h".

Local routines should be declared "static".

Example

```c
#include "cics.h"
int my_exit_program();
static int my_own_routine();
int my_exit_program() {
    my_own_routine();
    return 0;
}
static int my_own_routine() {
    return 1;
}
```

Compiling & Linking

The exit program sources must be compiled, producing a link-editable PIC object (.o), and then linked into a shared library.

Both compiling and linking require a set of platform and installation dependant options, that must fit the XCICS building options. To facilitate the programmer's work, an include file for the make system is available at $XFRAMEHOME/include/xport.make. Including this file in the user Makefile the correct set of options and command is ready to be used.

Example

```make
# makefile
#
COMPILE-S(C) -I $(XFRAMEHOME)/include \
    -I$(XFRAMEHOME)/xcics/cpy \
    -I$(XFRAMEHOME)/xsdf/cpy
```
3.1.11.1.1 odcs_ts routine

The odcs_ts routine is invoked by XCICS/TS Online Data Conversion System (ODCS) to convert the temporary storage records, whenever a TS queue is accessed on or by remote EBCDIC systems.

Normally the ODCS routines have per-resource definition, but temporary storage queues, which require no configuration, represent and exception: because of that user need to define one converter routine for all TS queue: the odcs_ts exit program.

In this case the routine must be smart enough to recognize by itself (using the information provided by XCICS) the data being converted.

The odcs_ts routine must conform to the rules defined by the XCICS/TS Online Data Conversion System (ODCS).

I.e.

```c
int my_ts_conv(struct xcics_conversion_info *ci) {
    if (strcmp(ci->target, "QUEUE01X")==0) {
        if (ci->direction==CONVERT_ASCII_TO_EBCDIC) {
            CICSConv_A2E_CHAN(ci->area, ci->length);
        } else {
            CICSConv_E2A_CHAN(ci->area, ci->length);
        }
        return 0;
    }
    if (strcmp(ci->target, "QUEUE00000")==0) {
        if (ci->direction==CONVERT_ASCII_TO_EBCDIC) {
            CICSConv_A2E_ZONOEDX(ci->area, 8);
            CICSConv_A2E_CHAN(ci->area, ci->length-8);
        } else {
            CICSConv_E2A_ZONOEDX(ci->area, ci->length-8);
            CICSConv_E2A_CHAN(ci->area, ci->length-8);
        }
        return 0;
    }
    return 0;
}
```

For further information on ODCS routines please refer to the "On-line Data Conversion" section.

3.1.11.1.2 snt_loader routine

This routine is invoked by XCICS/TS at startup and reconfiguration time to let the user add programmatically entries in the SNT (users).

For each user entry it wants to add to the SNT, the routine must call the function `fcx_add_snt_entry` passing the pointer
to an area type struct xcics_user_info containing the information of the user being created.

**Prototype**

```c
int function_name(BOOL startup);
```

**Passed parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startup</td>
<td>BOOL</td>
<td>TRUE when invoked at XCICS/TS startup.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE when invoked at XCICS/TS reconfiguration.</td>
</tr>
</tbody>
</table>

**Return codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>!=0</td>
<td>fault detected</td>
</tr>
</tbody>
</table>

**Sample declaration**

```c
#define exit_program type=snt_loader, name=my_define_snt, library=libmysnt.so;
```

**Sample code**

```c
#include "cics.h"
int my_define_snt() {
    struct xcics_user_info ui;
    memset(&ui, 0x00, sizeof(ui));
    ui.userid="BATMAN";
    ui.userclass="OPERATOR";
    ui.username="Bruce Wayne";
    fCX_add_snt_entry(&ui);
    memset(&ui, 0x00, sizeof(ui));

    ui.userid="BATMAN";
    ui.userclass="OPERATOR";
    ui.username="Bruce Wayne";
    fCX_add_snt_entry(&ui);
    return 0;
}
```

### 3.11.11.3 user_signon routine

When XCICS/TS auth_mode is set to "user", an user_signon routine must be declared.

This routine is invoked by XCICS/TS at operator SIGNON time (CESN and EXEC CICS SIGNON) to verify if an operator is authorized to use the requested USERID. The USERID must be declared in SNT.

Once XCICS/TS verified the the requested USERID has been defined, it calls the routine passing USERID and
PASSWORD entered by the operator and the pointer to the SNT entry of the requested user. The routine must return a valid return code to communicate to XCICS/TS whether the user is authorized or not.

**Prototype**

```c
int function_name(char *userid, char *password, tCXsnt *snt_entry);
```

**Passed parameters**

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userid</td>
<td>char *</td>
<td>a NULL terminated string, containing the USERID</td>
</tr>
<tr>
<td>password</td>
<td>char *</td>
<td>a NULL terminated string, containing the password</td>
</tr>
<tr>
<td>snt_entry</td>
<td>tCXsnt *</td>
<td>a pointer to the SNT entry of the user</td>
</tr>
</tbody>
</table>

**Return codes**

- **CXE_OK** user is authenticated
- **CXE_NOTAUTH** user is not authenticated

**Sample declaration**

```c
#define exit_program type=user_signon, name=my_signon, library=libmysnt.so;
```

**Sample code**

```c
#include "cics.h"
int my_signon(char *userid, char *password, tCXsnt *snt_entry) {
    if (strcmp(userid, "UI98736")==0 && strcmp(password, "PSW00011")==0) {
        return CXE_OK;
    }
    if (strcmp(userid, "UI98556")==0 && strcmp(password, "PSW00011")==0) {
        return CXE_OK;
    }
    return CXE_NOTAUTH;
}
```

### 3.1.11.1.4 user_signoff routine

When XCICS/TS auth_mode is set to "user", user may define an user_signoff routine to handle the operator SIGNOFF (CESF and EXEC CICS SIGNOFF). At SIGNOFF time, XCICS/TS calls the routine passing the USERID and the pointer to the SNT entry of the currently signed-on user.

The routine may handle this information and return a return code to communicate to XCICS/TS whether the operation succeeded or not.

If the return code is not equal to CXE_OK (0), the condition INVREQ is raised.
Prototype

```c
int function_name(char *userid, tCXsnt *snt_entry);
```

Passed parameters

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userid</td>
<td>char *</td>
<td>a NULL terminated string, containing the USERID</td>
</tr>
<tr>
<td>snt_entry</td>
<td>tCXsnt *</td>
<td>a pointer to the SNT entry of the user</td>
</tr>
</tbody>
</table>

Return codes

<table>
<thead>
<tr>
<th>code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>!=0</td>
<td>fault (INVREQ is raised)</td>
</tr>
</tbody>
</table>

Sample declaration

```c
#define exit_program type=user_signoff, name=my_signoff, library=libmysnt.so;
```

Sample code

```c
#include "cics.h"
int my_signoff(char *userid, tCXsnt *snt_entry) {
if (strcmp(userid, "UI98736")==0) {
    /** I don't want UI98736 to signoff */
    return 1;
}
return 0;
}
```

### 3.1.11.1.5 user_chpasswd routine

When XCICS/TS auth mode is set to "user", user may define an user_chpasswd routine to handle the operator password change SIGNOFF (CESN and EXEC CICS SIGNON NEWPASSWORD). If this routine is not declared operators will not be able to change their password.

After a successful SIGNON, if the a new password is requested, XCICS/TS calls the routine passing the USERID, the old and new PASSWORDs entered by the operator and the pointer to the SNT entry of the user.

The routine may handle this information and return a return code to communicate to XCICS/TS whether the operation succeeded or not.

If the return code is not equal to CXE_OK (0), the condition INVREQ is raised.

Prototype

```c
int function_name(char *userid, char *old_password, char *new_password, tCXsnt *snt_entry);
```
Passed parameters

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userid</td>
<td>char *</td>
<td>a NULL terminated string, containing the USERID</td>
</tr>
<tr>
<td>old_password</td>
<td>char *</td>
<td>a NULL terminated string, containing the old password</td>
</tr>
<tr>
<td>new_password</td>
<td>char *</td>
<td>a NULL terminated string, containing the new password</td>
</tr>
<tr>
<td>snt_entry</td>
<td>tCXsnt *</td>
<td>a pointer to the SNT entry of the user</td>
</tr>
</tbody>
</table>

Return codes

<table>
<thead>
<tr>
<th>code</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>!=0</td>
<td>fault (INVREQ is raised)</td>
</tr>
</tbody>
</table>

Sample declaration

```c
#define exit_program type=user_chpasswd, name=dfhep_chpasswd, library=libmysnt.so;
```

Sample code

```c
#include "cics.h"
int dfhep_chpasswd(char *userid, char *old_password, char *new_password, tCXsnt *snt_entry) {
    if (strcmp(userid, "UI98736") == 0) {
        /* I don't want UI98736 to change his password*/
        return 1;
    }
    if (my_store_password(userid, new_password)) {
        return 1;
    }
    return 0;
}
```

3.1.11.1.6 dfhaid routine

If declared, XCICS/TS invokes the dfhaid routine after terminal input is received, and before to (re)start a transaction. This routine may instruct XCICS/TS to start a specific transaction code, instead of the one eventually scheduled by the last RETURN TRANSID.

The routine receives the address of the TCTTETC area, which contains the identifier of the next transaction (TRANSID) to start and the AID entered by the user.

A return code not equal zero will communicate to XCICS/TS that the routine altered the TCTTETC.

Prototype

```c
int function_name(unsigned char *tcttetc, unsigned char key);
```

Passed parameters

<table>
<thead>
<tr>
<th>name</th>
<th>type</th>
<th>description</th>
</tr>
</thead>
</table>
unsigned char * tcttetc
address of the TCTTETC area

key char
AID received

Return codes

0 XCICS will handle the AID and transaction code
!=0 the routine altered the next transaction code

Sample declaration

```c
#define exit_program type=dfhaid, name=my_dfhaid_handler, library=libmysnt.so;
```

Sample code

```c
#include "cics.h"
#include "dfhaid.h"

int my_dfhaid_handler(unsigned char *tcttetc, unsigned char key) {
    if (memcmp(tcttetc, "MEN", 4) == 0) {
        return 0;
    }
    switch (key) {
        case DFHPA1:
            memcpy(tcttetc, "HLP1", 4);
            return 1;
        case DFHPA2:
            memcpy(tcttetc, "HLP2", 4);
            return 1;
        case DFHPF9:
            memcpy(tcttetc, "PRTX", 4);
            return 1;
    }
    return 0;
}
```
3.1.11. On-line data conversion

In some circumstances, data conversion between ASCII and EBCDIC is required. This happens when XCICS Transaction Server, running on a UNIX/Linux system (ASCII), communicates and exchanges data with a mainframe system (EBCDIC).

XCICS provides a transparent conversion system for particular features such:

- DPL - Distributed Program Link
- Function Shipping

User may request XCICS, at configuration time, to perform ASCII-EBCDIC translation for a specific resource when invoked/accessed on a server region running on a mainframe.

To do that, user have to:

- specify the correct OS encoding in the CCT definition of the remote system
- specify the conversion routine for the resource accessed by or on IBM CICS
- provide the conversion routines according to the XCICS Online Data Conversion specs

The dynamic conversion system may be specified and invoked on resource such:

- programs, invoked via DPL, to convert the commarea
- transactions, started via function shipping, to convert the commarea
- files, accessed via RFA, to convert the record and the key
- transient data, accessed via function shipping to convert the data record
- temporary storage, accessed via function shipping to convert the data record

Whenever a dynamic conversion is required, the name of the converter routine must be specified in the resource definition.

Temporary storage queues, which have no definition, are the only exception: user can define one only converter routine for TS queue, and specify it the general settings of XCICS: in this case the routine must be smart enough to recognize by itself (using the information provided by XCICS) the data being converted.

In the following example, a system named CICA is defined and the os=EBCDIC is specified, the program REMOTPGM is specifying CICA as server region name and a converter function.

With this definition, every time the program REMOTPGM is linked to, XCICS activates the DFHIMR transaction on the remote region, and performs a DPL. Before to send the commarea to the server region and before to return to the program the commarea returned by the server program, XCICS will invoke the "remotca" routine from the "libcaconv.so" library to convert the data from ASCII to and from EBCDIC.

I.e.

```
define connection sysid=CICA,
    type=sna, mode=LU62APPB, netname=CICSIBM0,
    os=ebcdic, acquire;
define program name=REMOTPGM,
    language=cobol,
    sysid=CICA,
    converter_function=remotca, converter_lib=libcaconv.so;
```

Conversion routines

Routines invoke by the XCICS online data conversion system must be contained in a shared library file, and must conform to the this prototype:

```
int function_name(struct xcics_conversion_info *p);
```

The function name is chosen by the user according to its standards/preferences. In order to avoid interferences with
other routines, its strongly suggested to choose a "unique" identifier to prefix the user-written functions.

The parameter supplied to the function is a pointer to a data structured according to the definitions supplied in "cicsconv.h".

If the routine returns 0 (zero) to the caller XCICS uses the data converted by the routine. Any other value is treated as an error, so data is handles AS-IS and no conversion occurs.

By means of "cicsconv.h" header file some macros are available to make coding easier.

### Information supplied to the routines

By means of the C structure **xcics_conversion_info** the following information are provided to the converter function:

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>C Data Type</th>
<th>Possible Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| direction      | int         | CONVERT_EBCDIC_TO_ASCII  
|                |             | CONVERT_ASCII_TO_EBCDIC | conversion type (ASCII to EBCDIC or viceversa) |
| area           | unsigned char * |                 | pointer to the area to be converted |
| length         | size_t      |                 | length or the area to be converted |
| remote_sysid   | char *      |                 | null terminated string containg the sysid of the remote system |
| operation_code | int         | DFHMIR_LINK_PROGRAM  
|                |             | DFHMIR_READ_DIRECT   
|                |             | DFHMIR_WRITE         
|                |             | DFHMIR_DELETE        
|                |             | DFHMIR_REWRITE       
|                |             | DFHMIR_UNLOCK        
|                |             | DFHMIR_START_BROWSE   
|                |             | DFHMIR_END_BROWSE     
|                |             | DFHMIR_RESET_BROWSE   
|                |             | DFHMIR_READ_NEXT      
|                |             | DFHMIR_READ_PREVIOUS  
|                |             | DFHMIR_START          
|                |             | DFHMIR_READQ_TS       
|                |             | DFHMIR_WRITEQ_TS      
|                |             | DFHMIR_DELETEQ_TS     
|                |             | DFHMIR_READQ_TD       
|                |             | DFHMIR_WRITEQ_TD      
|                |             | DFHMIR_DELETEQ_TD     | value describing the operation being performed |
| target         | char *      |                 | null terminated string identifying the name of the resource (i.e. the program name in a DPL, the TS name in a TS function shipping op.) |
| type           | int         | DFHMIR GENERIC_ARAE  
|                |             | DFHMIR COMMAREA      
|                |             | DFHMIR RECORD        
|                |             | DFHMIR RIDFLD        | a value identifying what being converted |
Automatic routine generation

Conversion routines may be automatically generated using the XCONV xmlconverter utility, by means of the -area option.
In the following example, the structure of the COMMAREA of REMOTPGM, described in the COBOL copybook REMOTCA.cpy is automatically translated into an online conversion routine.

First of all, the copybook is analyzed and its structure is converted into an XML description of the structure itself:

```
# cpy2xml -o REMOTCA xml REMOTCA.cpy
```

Then, the XML is translated into a C conversion routine, suitable for the online system:

```
# xmlconverter -area -o remotca.c REMOTCA.xml
```

and, at the end, the routine is compiled and linked into a shared library (the commands below may vary according to the different operating systems):

```
# cc -c remotca.c
# ld -G -o libcaconv.so remotca.o
```

Multiple record structures

When a COMMAREA or a record structure contains multiple definitions (redefines), dynamic conversion system can still be used, but the programmer must also code the recognition logic.

So the programmer have to:

1. identify the different overlapping record structures
2. properly convert them to XML using cpy2xml
3. generate the conversion routine using xmlconverter
4. edit the generated c code, adding the routine entry point (the one that will be called by XCICS), which should be able to distinguish between different record structure and call the proper conversion routine (generated by xmlconverter)
5. compile and link the routine

I.e.

```c
int convert_multi_ca(struct xcics_conversion_info *p) {
    unsigned char *code0001;
    BYTE function_code[4];
    register int i;

    /*
     * depending on the conversion direction
     * the record code is translated
     */
    memcpy(function_code, p->area, 4);
    if (p->direction==CONVERT_EBCDIC_TO_ASCII) {
        CICSCONV_E2A_CHAR (function_code, 4);
    }
}
```
Conversion macros

The "cics.h" includes a number of conversion macros which may be used to translate the data. These macros translate data according to the codepage and DBCS table configured in the region.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICSCONV_E2A_CHAR(pointer, lenght)</td>
<td>convert from EBCDIC to ASCII the area addressed by pointer for the specified length as character field (PIC X)</td>
</tr>
<tr>
<td>CICSCONV_A2E_CHAR(pointer, lenght)</td>
<td>convert from ASCII to EBCDIC the area addressed by pointer for the specified length as character field (PIC X)</td>
</tr>
<tr>
<td>CICSCONV_E2A_ZONED(pointer, lenght)</td>
<td>convert from EBCDIC to ASCII the area addressed by pointer for the specified length as zoned field (PIC 9 or S9)</td>
</tr>
<tr>
<td>CICSCONV_A2E_ZONED(pointer, lenght)</td>
<td>convert from ASCII to EBCDIC the area addressed by pointer for the specified length as zoned field (PIC 9 or S9)</td>
</tr>
<tr>
<td>CICSCONV_E2A_DBCS(pointer, lenght)</td>
<td>convert from EBCDIC to ASCII the area addressed by pointer for the specified length as DBCS field (PIC G)</td>
</tr>
<tr>
<td>CICSCONV_A2E_ZONED(pointer, lenght)</td>
<td>convert from ASCII to EBCDIC the area addressed by pointer for the specified length as DBCS field (PIC G)</td>
</tr>
<tr>
<td>CICSCONV_LOG(string)</td>
<td>logs the NULL terminated string into the terminal log file</td>
</tr>
</tbody>
</table>
3.1.12 MQ Trigger Monitors

XCICS/TS provides an IBM WebSphere MQ CICS Adapter like feature: the MQ Trigger Monitor services. These services act like the mainframe task CKTI: they listen on a MQ initiation queue and activate an user task whenever a message is catch on the initiation queue.

The mechanism

The queue manager (QueueMgr) has three queues defined: CICS01.INITQ, CICS01.IN and CICS01.OUT. The CICS01.INITQ is the initiation queue and it is used by the MQ triggering mechanism (triggering enables the activation of programs when messages arrive on queues). The initiation queue receives trigger messages (created by the queue manager) indicating the event of the arrival of a message on a queue (in this case CICS01.IN). The trigger message has the name of the queue where the message has arrived and the name of the program that needs to be started to process the message.

An MQ object called PROCESS must be defined when triggering is used. The process has the information about of the type and name of the application that is started by the trigger monitor. The name of the process is configured as an attribute of the application queue (CICS01.IN). The process object must exist and make a reference to the user transaction name to start.

The transaction CKTI (part of the XCICS/TS trigger monitor system) is normally started during CICS startup (or manually using xcsadm) to monitor the CICS01.INITQ queue. The queue CICS01.IN is configured to generate a trigger message when the first message arrives (trigger first) on the queue and to request the start of a CICS transaction to process messages that arrive on the queue.

CKTI gets trigger messages from the CICS01.INITQ initiation queue and the message has the name of the CICS transaction that is started to process the message. CKTI starts the user transaction and passes a copy of the trigger message. The user transaction receives the trigger message, it finds the name of the queue that has messages waiting to be processed, and it gets the messages and processes them. As result of the process a response is created and put on the reply queue CICS01.OUT.

IMPORTANT: do not define any trigger monitor for the queues that are part of the XCICS/ MQ communication. The CKTI transaction is their trigger monitor.

Example

Queue & Process definition

```c
define qlocal(CICS01.INITQ) REPLACE DESC('Initialization queue for CICS01')
define qlocal(CICS01.IN) REPLACE DESC('Input queue for CICS01') +
    initq(CICS01.INITQ) process(CICS01.PROCESS) trigger trgtyp(TRANS) every
define process(CICS01.PROCESS) REPLACE DESC('User transaction') +
    applid(CICS) applicid('QCTL') userdatadesc('Data from process')
define qlocal(CICS01.OUT) REPLACE DESC('Output queue for CICS01')
```

XCICS configuration

```c
set use_mpseries=yes;
# server XA string
set mpseries_xa_string="qname=venus.queue.manager";
# client XA string
# set mpseries_xa_string="channel=CHANNEL.FC, trotype=tcp, connname=hpux02, qname=venus.queue.manager"
set mpseries_queue_manager=venus.queue.manager;
add mptrgm queue= CICS01.INITQ

define program name=MYPGM1$
define transaction name=QCTL, program=MYPROGRAM, twa=512, protection=1;
3.1.12. Programming considerations

When correctly setup, programs running under XCICS/TS will be able to call WSMQ API CALLs (i.e. MQPUT, MQGET).

MQCONN and MQDISC should not be called: XCICS/TS takes care about the connection to the WSMQ queue manager and provides a default connection.

Like IBM CICS, each XCICS/TS region may have only one connection to WSMQ. This connection is part of the LUW (Logical Unit of Work) and therefore it is under the transactional control of XCICS/TS. EXEC CICS SYNCPOINT or SYNCPOINT ROLLBACK must be used in place of MQBEGIN, MQCMIT and MQBACK.

WARNING: WSMQ API calls have a different behaviour on mainframe and UNIX/Linux system. On mainframe systems, WSMQ API calls are put by default within the LUW, while on UNIX/Linux systems, they are not put within the LUW by default.

Because of this difference, programs calling WSMQ API should specify explicitly MO_SYNCPOINT or MO_NO_SYNCPOINT in the WSMQ API calls.

I.e.

mainframe code:

```
CALL 'MQPUT' USING HCONN, Q-HANDLE, MESSAGE-DECRIPTOR, PMOPTIONS, BUFFER-LENGTH, IO-BUFFER, COMPLETION-CODE, REASON.
```

UNIX/Linux code:

```
MOVE MQPMO-SYNCPOINT TO MQPMO-OPTIONS OF PMOPTIONS
CALL 'MQPUT' USING HCONN, Q-HANDLE, MESSAGE-DECRIPTOR, PMOPTIONS, BUFFER-LENGTH, IO-BUFFER, COMPLETION-CODE, REASON.
```
3.2 XCICS Programming Guide

This book gives guidance about the development of application programs that use the CICS EXEC application programming interface to access CICS services and resources.

It also gives guidance information on debugging such CICS applications.
3.2.1 Programming guide

An application is a collection of related programs that together perform a business operation. XCICS applications execute under XCICS control, using XCICS services and interfaces to access programs and files.

XCICS is a transaction processing subsystem. That means that it provides services for you to run applications online, by request, at the same time as many other users are submitting requests to run the same applications, using the same files and programs. XCICS manages the sharing of resources; integrity of data and prioritization of execution, with fast response.

XCICS applications are traditionally run by submitting a transaction request. Execution of the transaction consists of running one or more application programs that implement the required function.

You should note that the term transaction is now used extensively in the IT industry to describe a unit of recovery or what XCICS calls a unit of work. This is typically a complete operation that is recoverable; it can be committed or backed out as an entirety as a result of programmed command or system failure. In many cases the scope of a XCICS transaction is also a single unit of work, but you should be aware of the difference in meaning when reading XCICS documentation.

CICS programs, transactions and tasks

To develop and run XCICS applications, you need to understand the relationship between XCICS programs, transactions and tasks. These terms are used throughout XCICS documentation and appear in many commands:

Transaction

A transaction is a piece of processing initiated by a single request. This is usually from an end-user at a terminal, but may also be made from a Web Service, from a remote workstation program, from an application in another XCI/CICS system or triggered automatically at a predefined time.

A single transaction consists of one or more application programs that, when run, carry out the processing needed.

However, the term transaction is used in XCICS to mean both a single event and all other transactions of the same type. You describe each transaction type to CICS with a TRANSACTION resource definition. This definition gives the transaction type a name (the transaction identifier, or TRANSID) and tells XCICS several things about the work to be done; such as what program to invoke first, and what kind of authentication is required throughout the execution of the transaction.

Task

You will also see the word task used extensively in XCICS documentation. This word also has a specific meaning in XCICS. When XCICS receives a request to run a transaction, it starts a new task that is associated with this one instance of the execution of the transaction. type. That is, one execution of a transaction, with a particular set of data, usually on behalf of a particular user at a particular terminal. When the transaction completes, the task is terminated.
3.2.1.1 General rules

You write an XCICS program in much the same way as you write any other program. You can use COBOL, C, Java, or PL/I to write XCICS application programs. Most of the processing logic is expressed in standard language statements, but you use XCICS commands, or the Java class libraries to request CICS services.

This book describes the use of the CICS compatible command level programming interface, 'EXEC CICS', that can be used in COBOL, C or PL/I programs. These commands are defined in detail in the XCICS Command Reference.

Also Java interface APIs are described in the Java Programming chapter. User can write XCICS programs in much the same way as he writes any other program. He can use COBOL, C, Java and PL/I to write XCICS application programs.

XCICS is source level compatible with IBM CICS (tm), so, in almost all cases, user can directly recompile its mainframe sources and let it run in the XFRAME environment.

EXEC CICS blocks

For COBOL, C and PL/I, XCICS commands must be specified in the IBM-like EXEC CICS instruction block.

In COBOL the EXEC block is terminated by END-EXEC.

In PL/I and C language, the EXEC CICS block is terminated by ";".

I.E.

COBOL:
MOVE A TO B
EXEC CICS HANDLE CONDITION
MAPFAIL (MAPF-01)
END-EXEC

C language:
memcpy(b, t, 20);
EXEC CICS SEND TEXT FROM(t) LENGTH(20);

Exceptional Conditions Handling

CICS "EXCEPTIONAL CONDITIONS" or "CONDITIONs", are defined to catch CICS errors.

Conditions may occur at any time during execution of a command and, unless it is specified otherwise in application programs.

If no handling has been set, a default action for each condition is taken automatically by XCICS. Usually, these default actions terminate tasks abnormally.

There are three possible states in which an application program can be, in respect with a particular condition detected during the attempted execution of a particular command, as follows:

HANDLE CONDITION active

In this case, control goes to the label in the program defined earlier by a HANDLE CONDITION command.

This state occurs later the execution of a HANDLE CONDITION.

Take no action

The control returns to the next instruction following the command which execution failed.

This state occurs after the execution of an IGNORE CONDITION. If there is no current HANDLE CONDITION command that includes a label.

This state also occurs during execution of a command including one of the options NOHANDLE or RESP.
**Default action**

For most conditions this is to terminate the task abnormally.

This state occurs if a HANDLE CONDITION has never been executed. This state also occurs after executing a HANDLE CONDITION without label or PUSH HANDLE.

**Alternative to the HANDLE Command.**

The NOHANDLE, RESP and RESP2 options (described below) are supplied as an alternative to the HANDLE command just described.

These commands too are fully supported by XCICS.

**NOHANDLE Option**

You can use the NOHANDLE option with any command to specify that you want no action to be taken for any condition or AID resulting from the execution of that command.

**RESP Option**

You can use the RESP option with any command to test the response to the execution of the command.

```
RESP (xxx)
```

“xxx” is a user-defined fullword binary data area. At the return from the command, it contains a value corresponding to the condition that may have been raised, or to a normal return, that is xxx = DFHRESP(NORMAL)

**Handle Exceptional Condition (HANDLE CONDITION)**

You use this command to specify the label to which control is to be passed if a condition occurs.

You must include the name of the condition and, optionally, a label to which control is to be passed if the condition occurs.

```
HANDLE CONDITION <condition> [(label)]
```

**Ignore Exceptional Conditions (IGNORE CONDITION)**

You use this command to specify that no action is to be taken if the condition occurs (that is, control to the instruction following to the command that has failed).

```
IGNORE CONDITION <condition>
```

**Supported conditions**

The following conditions can be referred in the HANDLE CONDITION and IGNORE CONDITION statements:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Condition</th>
<th>Condition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIDERR</td>
<td>DSSTAT</td>
<td>DUPKEY</td>
<td>DUPREC</td>
</tr>
<tr>
<td>ENDDATA</td>
<td>ENDFILE</td>
<td>ENDINPT</td>
<td>ENQBUSY</td>
</tr>
<tr>
<td>ENVDEFERR</td>
<td>EOC</td>
<td>EODS</td>
<td>EOF</td>
</tr>
<tr>
<td>ERROR</td>
<td>EXPIRED</td>
<td>FUNCERR</td>
<td>ILLOGIC</td>
</tr>
<tr>
<td>INBFMH</td>
<td>INVERRTERM</td>
<td>INVLDG</td>
<td>INVMPNZ</td>
</tr>
<tr>
<td>INVPARTN</td>
<td>INVPARTNSET</td>
<td>INVREQ</td>
<td>INVTSREQ</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>IOERR</td>
<td>ISCINVREQ</td>
<td>ITEMERR</td>
<td>JIDERR</td>
</tr>
<tr>
<td>LENGERR</td>
<td>MAPFAIL</td>
<td>NOJBUFSP</td>
<td>NONVAL</td>
</tr>
<tr>
<td>NOPASSBKRD</td>
<td>NOPASSBKWR</td>
<td>NOSPACE</td>
<td>NOSTART</td>
</tr>
<tr>
<td>NOSTG</td>
<td>NOTALLOC</td>
<td>NOTFND</td>
<td>NOTOPEN</td>
</tr>
<tr>
<td>OVERFLOW</td>
<td>PARTNFAIL</td>
<td>PGMIDERR</td>
<td>QBUSY</td>
</tr>
<tr>
<td>QIDERR</td>
<td>QZERO</td>
<td>RDATT</td>
<td>RETPAGE</td>
</tr>
<tr>
<td>RTEFAIL</td>
<td>RTESOME</td>
<td>SEGIDERR</td>
<td>SELNERR</td>
</tr>
<tr>
<td>SESSBUSY</td>
<td>SESSIONERR</td>
<td>SIGNAL</td>
<td>SYSBUSY</td>
</tr>
<tr>
<td>SYSIDERR</td>
<td>TERMIDERR</td>
<td>TRANSIDERR</td>
<td>TSIOERR</td>
</tr>
<tr>
<td>UNEXPIN</td>
<td>WRBRK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Suspend Condition Handling (PUSH and POP)**

The commands PUSH and POP enable the suspension of all current HANDLE CONDITION, HANDLE AID, and HANDLE ABEND commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.

```
PUSH HANDLE
POP HANDLE
```
3.2.1.2 Programming in COBOL

XCICS supports COBOL programs that have been compiled using either:

- MicroFocus Server Express 2.0 or higher
- AcuCOBOL 7

With COBOL you must use only EXEC CICS commands to invoke operating system services. Some COBOL statements must not be used. Simplified based addressing using cell pointer variables and the ADDRESS special register.

Programming restrictions

This section describes COBOL language elements that you cannot use under CICS, or whose use is restricted or can cause problems under XCICS.

In general, neither the XCICS translator nor the COBOL compiler detects the use of COBOL words affected by the following restrictions. The use of a restricted word in a CICS environment may cause a failure at execution time.

The following restrictions apply to a COBOL program that is to be used as an XCICS application program. (See the appropriate COBOL programming guide for more information about these functions.)

- Statements that produce variable-length areas, such as OCCURS DEPENDING ON, should be used with caution within the WORKING-STORAGE SECTION.
- If you have any CICS applications written in COBOL, you may need to review the COBOL runtime options in use at your installation. In particular, if your applications are not coded to ensure that the WORKING-STORAGE SECTION is properly initialized (for example, cleared with binary zeros before sending maps), you should use the correct compiler option to prevent this (i.e. DEFAULTBYTE=0 in MFCOBOL).
- You cannot use entry points in COBOL in XCICS.
- Do not use EXEC CICS commands in a Declaratives Section.
- Do not use a ACCEPT FROM CONSOLE statement in a CICS program.
- Do not use DISPLAY . . . UPON CONSOLE and DISPLAY . . . UPON SYSPUNCH. DISPLAY to the system logical output device (SYSOUT, SYSLIST, SYSLST) is supported.
- Do not use STOP RUN.

Using based addressing with COBOL

XCICS application programs need to access data dynamically when the data is in a XCICS internal area, and only the address is passed to the program. Examples are:

- XCICS areas such as the CWA, TWA, and TCTTE user area (TCTUA), accessed using the ADDRESS command
- Input data, obtained by EXEC CICS commands such as READ and RECEIVE with the SET option

COBOL provides a simple method of obtaining addressability to the data areas defined in the LINKAGE SECTION using pointer variables and the ADDRESS special register.

The ADDRESS special register holds the address of a record defined in the LINKAGE SECTION with level 01 or 77. This register can be used in the SET option of any command in ADDRESS mode. These commands include GETMAIN, LOAD, READ, and READQ.

Calling subprograms from COBOL

In an XCICS system, when control is transferred from the active program to an external program, but the transferring program remains active and control can be returned to it, the program to which control is transferred is called a subprogram.
There are three ways of transferring control to a subprogram:

**EXEC CICS LINK**

The calling program contains a command in one of these forms:

- EXEC CICS LINK PROGRAM('subpgname')
- EXEC CICS LINK PROGRAM(name)

In the first form, the called subprogram is explicitly named as a nonnumeric literal within quotation marks. In the second form, name refers to the COBOL data area with length equal to that required for the name of the subprogram.

**Static COBOL call**

The calling program contains a COBOL statement of the form:

```
CALL 'subpgname'
```

**Dynamic COBOL call**

The calling program contains a COBOL statement of the form:

```
CALL identifier
```

The identifier is the name of a COBOL data area that must contain the name of the called subprogram.

COBOL programs can call any language programs statically or dynamically. LINK or XCTL are not required for inter-language communication, unless you wish to use CICS functions such as COMMAREA.

Each LINK command creates a new logical level, the called program being at a level one lower than the level of the calling program (XCICS is taken to be at level 0). Figure 3 in topic 1.4.4.2 shows logical levels and the effect of RETURN commands and CALL statements in linked and called programs.

**Rules for calling subprograms**

The following rules describe the requirements and behavior of called or linked subprograms.

**Translation**

**LINK**

The linked subprogram must be translated if it, or any subprogram invoked from it, invokes XCICS commands.

**Static and Dynamic COBOL CALL**

The called subprogram must be translated if it contains CICS commands or references to the EXEC interface block (DFHEIBLK) or to the CICS communication area (DFHCOMMAREA).

**Compilation**

All programs must be compiled.
XCICS configuration entries

**LINK**
The linked subprogram must be defined in PPT. If the linked subprogram is unknown or unavailable, even though autoinstall is active, the LINK fails with the PGMIDERR condition.

**Static & Dynamic COBOL CALL**
The calling program must not be defined in PPT.

Return from subprogram

**LINK**
The linked subprogram must return using either RETURN or a native language return command such as the COBOL statement GOBACK.

**Static and Dynamic COBOL CALL**
The called subprogram must return using a EXIT PROGRAM.
The use of RETURN in the called subprogram leads to unpredictable results.

Language of subprogram

**LINK**
Any language supported by XCICS.

**Static and Dynamic COBOL CALL**
Any language.

Passing parameters to subprogram

Data can be passed by any of the standard CICS methods (COMMAREA, TWA, TCTUA, TS queues) if the called or linked subprogram is processed by the CICS translator.

**LINK**
If the COMMAREA is used, its address must be passed in the LINK command. If the linked subprogram uses 24-bit addressing, and the COMMAREA is above the 16MB line, CICS copies it to below the 16MB line, and recopies it on return.

**Static COBOL CALL**
The CALL statement may pass DFHEIBLK and DFHCOMMAREA as the first two parameters, if the called program is to issue EXEC CICS requests, or the called program can issue EXEC CICS ADDRESS commands. The COMMAREA is optional but if other parameters are passed, a dummy COMMAREA must also be passed.

**Dynamic COBOL CALL**
The CALL statement may pass DFHEIBLK and DFHCOMMAREA as the first two parameters, if the called program is to issue EXEC CICS requests, or the called program can issue EXEC CICS ADDRESS commands. The COMMAREA is...
optional but if other parameters are passed, a dummy COMMAREA must also be passed.
3.2.1.3 Programming in C

XCICS supports C programs that have been compiled using either:

- Vendor specific ANSI/C Compiler
- GNU GCC

All the EXEC CICS commands available in COBOL and PL/I applications are also supported in C applications, with the exception of those commands related to nonstructured exception handling.

Also note the following programming considerations:

### Exception handling

#### Nonstructured exception handling

The EXEC CICS commands related to nonstructured exception handling:

- HANDLE ABEND LABEL(label)
- HANDLE AID
- HANDLE CONDITION
- IGNORE CONDITION
- PUSH HANDLE
- POP HANDLE

are not supported in C applications. Use of these commands is diagnosed by the translator.

#### Condition handling

In a C application, every EXEC CICS command is treated as if it had the NOHANDLE or RESP option specified. This means that the set of "system action" transaction abends that result from a condition occurring but not being handled, is not possible in a C or C++ application. Control always flows to the next instruction, and it is up to the application to test for a normal response.

#### ABEND handling

HANDLE ABEND PROGRAM commands are allowed, but you cannot use PUSH HANDLE or POP HANDLE.

### COMMAREA

The address of the communication area is passed as an argument to a C main function. Of course, C functions can still use ADDRESS COMMAREA to obtain the address of the communications area.

### EIB

The address of the EIB is passed as an argument to a C main function. Of course, C functions can still use ADDRESS EIB to obtain the address of the EIB.

### OVERFLOW conditions

If you want any OVERFLOW condition to be indicated in the RESP field on return from a SEND MAP command with the ACCUM option, you should specify the NOFLUSH option.
**Return value**

If you terminate a C program with an exit() function or the return statement, instead of a CICS RETURN command, the value passed through the exit() function is saved in the EIBRESP2 field of the EIB on return from the program.

**Data declarations**

The following data declarations are provided by CICS for C:

- Execution interface block definitions (EIB)
- BMS screen attributes definitions: C versions of the DFHBMSCA, DFHMSRCA, and DFHAID files are supplied by CICS, and may be included by the application programmer when using BMS.

The EIB declarations are enclosed in #ifndef and #endif lines, and are included in all translated files.

**Restrictions**

The following lists describe some of the restrictions that exist with C programs XCICS:

- C and C++ languages do not support packed decimal data. The application has access to packed decimal data using the character string data type. No C or C++ standard library functions are available to perform arithmetic on this data, but you may write your own.
- You can easily use HOURS, MINUTES, and SECONDS options. You may define expiration times using TIME or INTERVAL options if you provide functions to handle them in your application.
- You can enter all CICS keywords in mixed case.
- If you do not specify the LENGTH option on commands that support LENGTH (for example, READ, READNEXT, READPREV, and WRITE commands), the translator does not supply a default value. In effect, NOLENGTH is implicit for C programs.
- The string handling functions in the C standard library use a null character as an end-of-string marker. Because XCICS uses a different marker, you must take care when using C or C++ functions, for example strcmp, to operate on XCICS data areas.
- Two arguments, argc and argv, are normally passed to a C or C++ main function. argc denotes how many variables have been passed; argv is an array of zero-terminated variable strings. In XCICS, the value of argc is 4 and the argv array contains:
  - argv[0] contains the address of a zero-terminated string containing a copy of EIBTRNID
  - argv[1] contains the address of the EIB block
  - argv[2] contains the address of the COMMAREA if any or NULL
  - argv[3] contains the EIBCALEN
  - argv[4] contains NULL
- Where CICS expects a fixed-length character string such as a program name, map name, or queue name, you must pad the literal with blanks up to the required length if it is shorter than expected.
- BMS maps structure must be packed, that means aligned to 1 byte boundary. The xbms2mod utility automatically produces OS dependant packed C structs.

**Passing arguments in C or C++**

Arguments in C language are copied to the program stack at run time, where they
are read by the function. These arguments can either be values in their own right, or they can be pointers to areas of memory that contain the data being passed. Passing a pointer is also known as passing a value by reference.

Other languages, such as COBOL and PL/I, usually pass their arguments by reference, which means that the compiler passes a list of addresses pointing to the arguments to be passed. This is the call interface supported by CICS. To pass an argument by reference, you prefix the variable name with &, unless it is already a pointer, as in the case when an array is being passed.

As part of the build process, the compiler may convert arguments from one data type to another. For example, an argument of type char may be converted to type short or type long.

When you send values from a C program to XCICS, the translator takes the necessary action to generate code that results in an argument list of the correct format being passed to XCICS. The translator does not always have enough information to enable it to do this, but in general, if the argument is a single-character or halfword variable, the translator makes a precall assignment to a variable of the correct data type and passes the address of this temporary variable in the call.

When you receive data from XCICS, the translator prefixes the receiving variable name with &, which causes the C compiler to pass it values by reference rather than by value (with the exception of a character string name, which is left unchanged). Without the addition of &, the compiler would copy the receiving variable and then pass the address of the copy to CICS. Any promotion occurring during this copying could result in data returned by CICS being lost.

This table shows the rules that apply when passing values as arguments in EXEC CICS commands:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Usage</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character literal</td>
<td>Data value (sender)</td>
<td>The user must specify the character literal directly. The translator takes care of any required indirection.</td>
</tr>
<tr>
<td>Character variable (char)</td>
<td>Data-area (receiver)</td>
<td>The user must specify a pointer to the variable, possibly by prefixing the variable name with &amp;.</td>
</tr>
<tr>
<td>Character variable (char)</td>
<td>Data value (sender)</td>
<td>The user must specify the character variable directly. The translator takes care of any required indirection.</td>
</tr>
<tr>
<td>Character string literal</td>
<td>Name (sender)</td>
<td>The user can either code the string directly as a literal string or use a pointer which points to the first character of the string.</td>
</tr>
<tr>
<td>Character string variable</td>
<td>Data-area (receiver) Name (render)</td>
<td>Whether receiving or sending, the argument should be the name of the character array containing the string—the address of the first element of the array.</td>
</tr>
<tr>
<td>Integer variable (short, long, or int)</td>
<td>Data-area (receiver)</td>
<td>The user must specify a pointer to the variable, possibly by prefixing the variable name with &amp;.</td>
</tr>
<tr>
<td>Integer variable (short, long, or int)</td>
<td>Data value (sender)</td>
<td>The user must specify the name of the variable. The translator looks after any indirection that is required.</td>
</tr>
</tbody>
</table>
Integer constant (short, long, or int)  Data value (sender)  The user must specify the name of the variable. The translator looks after any indirection that is required.

Structure or union  Data-area (sender)  Data-area (receiver)  The user must code the address of the start of the structure or union, possibly by prefixing its name with &.

Array (of anything)  Data-area (sender)  Data-area (receiver)  The translator does nothing. You must code the address of the first member of the array, which the compiler interprets as the address of the first member.

Pointer (to anything)  Data-area (sender)  Ptr-ref (receiver)  Whether receiving or sending, the argument should be the name of the variable that denotes the address of interest. The translator takes care of the extra level of indirection that is necessary to allow CICS to update the pointer.

Note: Receiver is where data is being received from CICS; Sender is where data is being passed to CICS.

Accessing the EIB

The address of the exec interface block (EIB) is passed as an argument to a C main function. C and C++ functions can use the ADDRESS EIB command too, to obtain the address of the EIB.

Addressability is achieved by using the command:

```
EXEC CICS ADDRESS EIB(dfheiptr);
```

Within a C application program, fields in the EIB are referred to in lower case and fully qualified as, for example, "dfheiptr->eibtmid".

The following mapping of data types is used:

- Halfword binary integers are defined as "short int"
- Fullword binary integers are defined as "long int"
- Single-character fields are defined as "unsigned char"
- Character strings are defined as "unsigned char" arrays

Compiler considerations

On some C compilers (i.e. IBM C for AIX 5.0), the 1 byte structure alignment, is not correctly handle. This causes errors when accessing BMS maps structures.

The GNU GCC compiler (3.x or higher) actually works correctly on all the platforms where XFRAME is available, therefore it is the suggestedes compiler for C language XCICS application development.
3.2.1.4 Programming in PL/I

XCICS supports PL/I programs that have been compiled using Liant OpenPL/I.

If OPTIONS(MAIN) is specified in an application program, that program can be the first program of a transaction, or control can be passed to it by means of a LINK or XCTL command.

In application programs where OPTIONS(MAIN) is not specified, it cannot be the first program in a transaction, nor can it have control passed to it by an LINK or XCTL command, but it can be link-edited to a main program.

**PL/I programming restrictions**

The following restrictions apply to a PL/I program that is to be used as a CICS application program.

You cannot use the multitasking built-in functions.

You cannot use the multitasking options.

You should not use the PL/I statements:

- EXIT
- STOP

You are provided with EXEC CICS commands for the storage and retrieval of data, and for communication with terminals. (However, you can use CLOSE, PUT, and OPEN, for SYSPRINT.)

Do not define variables or structures with variable names that are the same as variable names generated by the translator. These begin with DFH. Care must be taken with the LIKE keyword to avoid implicitly generating such variable names.

All PROCEDURE statements must be in upper case, with the exception of the PROCEDURE name, which may be in lower case.

If a CICS command uses the SUBSTR built-in function in defining a data value, it should include a LENGTH option to specify the data length, unless the translator option NOLENGTH is specified.
3.2.1.5 Programming in Java

XCICS supports Java programs, compiled by using JDK 1.4 or higher.

**Java programming considerations**

Java classes accessing XCICS services, must use the XCICS XJE APIs, by means of the classes in the package com. hite.xcics.

Each Java class acting as an XCICS program must extends the abstract class com.hite.xcics.AbstractProgram, and it must implement a method run(), which is the XCICS entry point. The run() method must be public and must throw ProgramTerminated exception.

i.e.

```java
public void run() throws ProgramTerminated {
  ...
}
```

**Accessing XCICS services**

XCICS services are invoked by means of an instance of com.hite.XCicsAPI named “cics”, which is a member of AbstractProgram.

The methods of XCicsAPI are described in [XCICS Java APIs](#).

**XCicsException exceptions**

All services accessed by means of object “cics”, throw exceptions extending com.hite.xcics.XCicsException.

**ProgramTerminated exception**

The com.hite.xcics.ProgramTerminated exception must never be caught.

**Accessing the EIB**

The EIB may be accessed by means of the com.hite.xcics.Dfheiblk class. The current instance of this class is retrieved using the getDfheiblk() method in XCicsAPI.

**Host Data Mapping package**

The com.hite.hdm package contains a set of classes useful to access mainframe-style data types such as packed, comp or fixed length character areas (PIC X). Very often it is used to pass data to XCICS services, and it may also be use to handle mainframe-like user fields.

**A sample program**

```java
import com.hite.xcics.*;
import com.hite.hdm.*;

public class JavaPgm extends AbstractProgram {
  public void run() throws ProgramTerminated {
    Dfheiblk dfheiblk = cics.getDfheiblk();
  }
}
```
try {
    cics.sendText("Hello world");
    cics.link("PROGRAM");
} catch (PgmiderrException pe) {
    cics.goback();
} catch (XCicsException e) {
    cics.log(e);
}
3.2.1.6 Compiling programs

XFRAME suite provides some scripts to make programmers work easier. This chapter briefly describes the most important tools for XCICS program development. For more detailed information about these tools, please refer to the Programmer's Tools book.

**Compiling COBOL programs**

The COBOL programs can be easily compiled by means of the `xcob` compiler script, which provides a set of facility to speed up program development and compiling.

Normally, once correctly configured, an on-line program can be compiled simply issuing:

```
xcob [options] <PROGRAM>
```

I.e.

```
# xcob -sua PRGM0010.pre
```

**Compiling PL/I programs**

PL/I programs can be compiled by means of the `xpli` compiler script, which provides an easy interface to XCICS translator and Liant PL/I compiler.

Once configured, a PL/I program may be compiled issuing:

```
xpli [options] <PROGRAM>
```

**Compiling C programs**

There is no specific script for C compiling, because there are to many C compilers available. Therefore, to compile C programs you have to follow these steps:

1. translate the C source with `xpre`
2. compile the translated source with your preferred C compiler
3. link the obtained object in a shared library

**Translate CICS commands**

First of all, CICS commands must be translated into C commands with `xpre`. It must be invoked with the option `--language=c`, to activate C language parser.

**Compile C source**

Then translated C source have to be compiled into PIC (Position Independent Code), with the C compiler (cc or gcc). The C compiler must be invoked adding the XFRAME headers path to the include search list and properly defining the `HWFLAG` and `OSFLAG` symbols, according to the XFRAME configuration.

**Create the shared library**

The code object obtained from the compiling, must be included into a shared library.

XFRAME provides an include for makefiles to define configuration dependant symbols and commands for program compiling: `xport.make`. Programmers simply have to include this file in their makefiles.

I.e.

```
```
Compiling Java programs

Java programs may be compiled directly using JDK compiler, the `javac` command. Simply add to the following jars to `classpath`:

- `$XFRAMEHOME/lib/hdm.jar`
- `$XFRAMEHOME/lib/xcics.jar`

i.e.

```
javac -classpath $XFRAMEHOME/lib/hdm.jar:$XFRAMEHOME/lib/xcics.jar:. JavaCicsClass.java
```

Of course, Java programmers may use `ant`
<path refid="xframe.classpath"/>
</classpath>
</javac>
</target>
</project>
3.2.1.7 Application design

This section introduces some basic concepts to help you design XCICS applications.

The programming models implemented in XCICS are inherited from IBM XCICS, and exhibit many of the characteristics of conversational, terminal-oriented applications. There are basically three styles of programming model:

- Terminal-initiated, that is, the conversational model
- Distributed program link (DPL), or the RPC model
- START, that is, the queuing model.

Once initiated, the applications typically use these and other methods of continuing and distributing themselves, for example, with pseudoconversations, RETURN IMMEDIATE or DTP. The main difference between these models is in the way that they maintain state (for example, security), and hence state becomes an integral part of the application design. This presents the biggest problem when you attempt to convert to another application model.

A pseudoconversational model is mostly associated with terminal-initiated transactions and was developed as an efficient implementation of the conversational model. With increased use of 1-in and 1-out protocols such as HTTP, it is becoming necessary to add the pseudoconversational characteristic to the DPL or RPC model.

In a conversational transaction, the length of time spent in processing each of a user’s responses is extremely short when compared to the amount of time waiting for the input. A conversational transaction is one that involves more than one input from the terminal, so that the transaction and the user enter into a conversation. A nonconversational transaction has only one input (the one that causes the transaction to be invoked). It processes that input, responds to the terminal and terminates.

Processor speeds, even allowing for accessing data sets, are considerably faster than terminal transmission times, which are considerably faster than user response times. This is especially true if users have to think about the entry or have to enter many characters of input. Consequently, conversational transactions tie up storage and other resources for much longer than nonconversational transactions.

A pseudoconversational transaction sequence contains a series of nonconversational transactions that look to the user like a single conversational transaction involving several screens of input. Each transaction in the sequence handles one input, sends back the response, and terminates.

Before a pseudoconversational transaction terminates, it can pass data forward to be used by the next transaction initiated from the same terminal, whenever that transaction arrives. A pseudoconversational transaction can specify what the next transaction is to be, using the TRANSID option of the RETURN command. However, you should be aware that if another transaction is started for that device, it may interrupt the pseudoconversational chain you have designed, unless you specify the IMMEDIATE option on the RETURN command. In this case, the transaction specified by the TRANSID command is attached regardless of any other transactions queued for this terminal.

No transaction exists for the terminal from the time a response is written until the user sends the next input and XCICS starts the next transaction to respond to it. Information that would normally be stored in the program between inputs is passed from one transaction in the sequence to the next using the COMMAREA or one of the other facilities that XCICS provides for this purpose.

In the XCICS architecture, each transaction occupies a transaction server process (engine) for its entire duration, releasing it only at its end.

That means that conversational transaction will keep occupied a server process, even if they are not working, and causing other transaction to use other processes and obliging to configure the region with an higher number of server processes. A pseudoconversational program will use one of server process only for its pure execution time, releasing the process as well as all other resources when completed.

To summarize, although conversational tasks may be easier to write, they have serious disadvantages both in performance and in their effect on the overall operability of the XCICS systems containing them. Processors are now larger, with more real storage and more power than in the past, and this makes conversational tasks less painful in small amounts; but if you use conversational applications, you may rapidly run into virtual storage constraint.

XCICS ensures that changes to recoverable resources (such as data sets, transient data, and temporary storage) made by a unit of work (UOW) are made completely or not at all. A UOW is equivalent to a transaction, unless that transaction issues SYNCPOINT commands, in which case a UOW lasts between syncpoints.

When a transaction makes a change to a recoverable resource, XCICS makes that resource unavailable to any other transaction that wants to change it until the original transaction has completed. In the case of a conversational transaction, the resources in question may be unavailable to other terminals for relatively long periods.

If you use pseudoconversational transactions, however, the resources are only very briefly unavailable (that is, during the
short component transactions). However, unless all recoverable resources can be updated in just one of these transactions, recovery is impossible because UOWs cannot extend across transactions. So, if you cannot isolate updates to recoverable resources in this way, you must use conversational transactions.

**Separating business and presentation logic**

In general, it is good practice to split applications into a part containing the business code that is reusable, and a part responsible for presentation to the client. This technique enables you to improve performance by optimizing the parts separately, and allows you to reuse your business logic with different forms of presentation.

When separating the business and presentation logic, you need to consider the following:

- Avoid affinities between the two parts of the application.
- Be aware of the DPL-restricted API
- Be aware of hidden presentation dependencies, such as EIBTRMID usage.

The following sample illustrates a simple XCICS application that accepts data from an end user, updates a record in a file, and sends a response back to the end user. The transaction that runs this program is the second in a pseudoconversation. The first transaction has sent a BMS map to the end user's terminal, and the second transaction reads the data with the EXEC CICS RECEIVE MAP command, updates the record in the file, and sends the response with the EXEC CICS SEND MAP command.

The EXEC CICS RECEIVE and EXEC CICS SEND MAP commands are part of the transaction's presentation logic, while the EXEC CICS READ UPDATE and EXEC CICS REWRITE commands are part of the business logic.

```cics
.. EXEC CICS RECEIVE MAP
.. EXEC CICS READ UPDATE
.. EXEC CICS REWRITE
.. EXEC CICS SEND MAP
```

A sound principle of modular programming in XCICS application design is to separate the presentation logic from the business logic, and to use a communication area and the EXEC CICS LINK command to make them into a single transaction, as in the following sample.

The presentation logic is implemented in the caller program:

```cics
.. EXEC CICS RECEIVE MAP
.. EXEC CICS LINK .
.. EXEC CICS SEND MAP
```

The business logic implemented in the called program:

```cics
EXEC CICS ADDRESS COMMAREA
.. EXEC CICS READ UPDATE
.. EXEC CICS REWRITE
.. EXEC CICS RETURN
```

Once the business logic of a transaction has been isolated from the presentation logic and given a communication area interface, it is available for reuse with different presentation methods. For example, you could use Distributed Program
Link (DPL) to implement a two-tier model, or XCICS Web support with the XCICS business logic interface, where the presentation logic is HTTP-based.
3.2.1.8 Debug

3.2.1.8.1 COBOL programs

On-line COBOL programs can be debugged using the Animator feature offered by Microfocus Server Express.

**Debugging with MicroFocus Server Express**

**Cross-session method**

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

**Debug of programs running on current terminal**

The following steps must be followed to start the debug of a the current terminal:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON, DISP=<display address>
```

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the Microfocus animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

**Debug of a specific transaction code**

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON, DISP=<display address>, TRAN=<transid>
```

Whenever the specified transid is started in the XCICS region, the Microfocus animator will start an XTERM session on the target display.

**Debug of programs running on a specific terminal**

Whenever the programmer wants to debug programs running on background terminal (ie. on a printer terminal), the TERM parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
• start the transaction CEDB with these parameters

CEDB ON, DISP=<display address>, TERM=<termid>

Whenever a transaction is started on the requested terminal identifier, the Microfocus animator will start an XTERM session on the target display.

**Unsolicited Dynamic Attachment Method**

This method allows the normal execution of an XCICS program, and only when the user wants to start to debug, attaches the debugger to the running process.

The following steps must be followed to start the animation of a TP program:

• connect XCICS using the terminal emulator
• enter the transaction:

CEDB ON

• wait for terminal to switch in debugging status
• take note of the PID shown on the screen (the PID is always shown in the X4J status bar)
• open a telnet session and enter:

anim <PID>

Note: when connecting XCICS with a TN3270 emulator (i.e. IBM Personal Communication), obviously no PID number is shown anywhere. In order to easily get the debug status of the terminal and the PID of the working process, simply enter on the screen

CEDB

**Deprecated procedure**

The following steps must be followed to start the animation of a TP program:

• connect XCICS using X4J
• start the transaction XANM
• disconnect XCICS
• reconnect XCICS
• send ENTER to XCICS and read the PID number on the status line of X4J

Note: when connecting XCICS with a TN3270 emulator (i.e. IBM Personal Communication), obviously no PID number is shown anywhere. In order to easily get the PID of the working engine, simply enter on the screen

XANM PID

before to start the transaction.
Debugging with ACUCOBOL Extend

XTerm method

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

Debug of programs running on current terminal

The following steps must be followed to start the debug of a the current terminal:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

CEDB ON,DISPLAY=<display address>

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the ACUCOBOL debugger animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

Debug of a specific transaction code

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

CEDB ON,DISPLAY=<display address>,TRAN=<transid>

Whenever the specified transid is started in the XCICS region, the ACUCOBOL debugger will start an XTERM session on the target display.

Debug of programs running on a specific terminal

Whenever the programmer wants to debug programs running on background terminal (ie. on a printer terminal), the TERM parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

CEDB ON,DISPLAY=<display address>,TERM=<termid>
Whenever a transaction is started on the requested terminal identifier, the ACUCOBOL debugger will start an XTERM session on the target display.

### Terminal method

If an X/Server is not available, ACUCOBOL may connect to a telnet session too. The procedures are the same as described above, except for the presence of the parameter TTY in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a unix session (telnet, ssh or serial) and enter:
  
  ```sh
runcbl --wait
  ```

- start the transaction CEDB with these parameters
  
  ```sh
CEDB ON,TTY=\langle device>\n  ```

The device string is the one shown in field “Terminal” by the runcbl command.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.

### Example

On the UNIX terminal:

```bash
# runcbl --wait
Named pipes created
Terminal: pts/14
Waiting for application runtime to open pipe
```

On the XCICS terminal:

```sh
CEDB ON,TTY=pts/14,TRAN=ACCT
```

### Thin client method

If XCICS has been configured with the option “allow_acu_thin=yes”, programmers have to debug on the ACUCOBOL thin client. To do that, the ACU thin client must be installed on the programmer workstation.

The procedures are the same as described above, except for the presence of the parameter THIN in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a DOS shell on the local PC, and run
  
  ```sh
  acuthin --wait --port <port number> --restart
  ```

- start the transaction CEDB with these parameters
  
  ```sh
CEDB ON,THIN=<thin client address>
  ```

The thin client address is the address of the workstation, followed by “:” and the port number.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.
**Example**

On the workstation at IP 192.168.1.32, open the DOS shell and type:

```
C:\Acucorp\Acubdi700\Acu gin\bin> acuthin --wait --port 8000 --restart
```

On the XCICS terminal:

```
CEDB ON,THIN=192.168.1.32:8000,TRAN=ACCT
```

**3.2.1.8.2 C language programs**

C language programs may be debugged using every symbolic debugger, which is able to remotely attach a process. In any case XCICS process to attach must be created with this procedure:

- connect XCICS using X4J
- enter CEDB ON
- wait for debug to be activated
- use the PID displayed to connect with your debugger

**Using GDB**

To debug interactively C programs with GDB (GNU Debugger), this procedure should be followed:

1. compile programs with debug symbols (-g option)
2. define an empty entry point (function) to be compiled & linked into a shared library to preload with "load library" command in xcics.conf. I.e.

```c
int gdb_entry_point() {
    return 0;
}
```
3. put a call to this function in the module to debug
4. startup XCICS
5. connect XCICS and issue CEDB ON and wait for its messages
6. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
7. from command-line, change PWD to source directory and issue:

```
gdb $XFRAMEHOME/bin/xcicsd <PID>
```
8. set a breakpoint on the dummy entry point and continue execution. I.e

```
gdb> break gdb_entry_point
```
9. continue using the transaction
10. when the program execution reaches the dummy entry point, debugger will stop.
11. at this point GDB have to reload symbols from user library using the sharedlibrary command. I.e.

   gdb> sharedlibrary

12. GDB is now ready to debug the application.
Remember that user programs is always reloaded from the shared library every time the transaction restart.
Please refer to GDB documentation for further information about it.

**Using DBX on AIX**

To debug interactively C programs with DBX, this procedure should be followed:

1. compile programs with debug symbols (-g option)
2. startup XCICS
3. connect XCICS and issue CEDB ON
4. for the CEDB messages on terminal.
5. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
6. from command-line, change PWD to source directory and issue:

   ```
   dbx -a <PID>
   ```

7. set a breakpoint on the module or function to debug and continue execution. I.e

   ```
   (dbx) stop in MYPROG
   (dbx) c
   ```

8. continue using the transaction
9. when the program execution reaches the dummy entry point, debugger will stop.
10. now it is possible to use dbx commands to debug the program.
Remember that user programs is always reloaded from the shared library every time the transaction restart.
Please refer to DBX documentation for further information about it.

**3.2.1.8.3 PL/I programs**

PL/I on-line programs can be debugged using Liant Codewatch(tm).
The following steps must be followed to start the interactive debug of an XCICS program:

- connect XCICS using X4J
- issue "CEDB ON"
- wait for CEDB to complete
- issue "XANM PLISTOP=<MODULE>" to defined to PL/I entry to break on
- read the PID number on the status line of X4J

Once PID is obtained, from a command line session, issue the command.
This script is facility provided with XFRAME, which starts codewatch and instructs its to attach the specified process, to load defined module and start debugging.

Please refer to your Liant manual for further information about Codewatch(tm).

3.2.1.8.4 BMS support

BMS (Basic Mapping Support) is an interface between XCICS and its application programs. It formats input and output display data in response to BMS commands in the application programs. An XCICS program does not use program language commands to perform input and output, but it normally uses BMS commands to communicate with the terminal.

The advantage of BMS is that it simplifies programming as the code is kept largely independent of any changes in the network or of the used terminals. Moreover it handles all terminal devices as standard IBM 3270 terminals, granting backward compatibility with those applications handling terminals with device-dependent streams.

XCICS/TS is completely compatible with IBM FULL-BMS management.

Also BMS maps definition is source-compatible with the mainframe one: in fact maps are defined using the DFHMSD, DFHMDI, DFHMDF macros system, like for IBM CICS.

Maps source definition coming from the mainframe, as well as newly created ones, may be compiled using "xbms2mod" utility.

For further information about BMS handling and map definition please refer to the "IBM CICS Application Programming Guide".
3.2.2 Commands reference

This section contains general information which applies to all the CICS or XCICS API commands.

**CICS API command format**

The general format of a CICS command is EXEC CICS followed by the name of the required command, and possibly by one or more options, as follows:

```
EXEC CICS <COMMAND> <OPTION>(<ARG>)
```

where:

**COMMAND**

describes the operation required (for example, READ).

**OPTION**

describes any of the many optional facilities available with each function. Some options are followed by an argument in parentheses. You can write options (including those that require arguments) in any order.

**ARG**

is a value such as “data-value” or “name”. A “data-value” can be a constant, this means that an argument that sends data to CICS is generally a “data-value”. An argument that receives data from CICS must be a “data-area”. Some arguments described as “data-area” can both send and receive data. In these cases, you must ensure that the “data-area” is not in protected storage.
3.2.2.1 ALLOCATE

Acquire a session to a remote APPC logical unit for use by an APPC mapped conversation.

ALLOCATE  SYSID(systemname)
[  PROFILE(name) ]
[  PARTNER(name) ]
[  STATE(cvda) ]

Options

PARTNER

specifies the name (8 characters) of a set of definitions that include the names of a remote LU (NETNAME) and a communication profile to be used on the allocated session. You can use this option as an alternative to specifying SYSID and PROFILE explicitly.

PROFILE

specifies the name (1-8 characters) of a set of session-processing options that are to be used during the execution of mapped commands for the session specified in the SYSID option. If you specify SYSID and omit PROFILE, a default profile (DFHCICSA) is selected.

STATE

gets the state of the current conversation. The cvda value returned by CICS is ALLOCATED.

SYSID

specifies the name (1-4 characters) by which the remote APPC LU is known to this CICS. This option requests that one of the sessions to the named system is to be allocated.
### 3.2.2.2 ASSIGN

Requests values from outside the application program’s local environment.

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWALENGTH</td>
<td>returns an halfword binary field indicating the length of CWA. If no CWA exists, a zero length is returned.</td>
</tr>
<tr>
<td>TCTTUALENG</td>
<td>returns an halfword binary field indicating the length of TCTUA.</td>
</tr>
<tr>
<td>TWALENG</td>
<td>returns an halfword binary field indicating the length of TWA.</td>
</tr>
<tr>
<td>SCRHT</td>
<td>returns an halfword binary field indicating the screen height in characters for the current terminal.</td>
</tr>
<tr>
<td>SCRWT</td>
<td>returns an halfword binary field indicating the screen width in characters for the current terminal.</td>
</tr>
<tr>
<td>PAGENUM</td>
<td>returns an halfword binary field indicating the number of the page, where overflow condition occurred.</td>
</tr>
</tbody>
</table>
FACILITY
returns a 4-bytes identifier of the principal facility that initiated the task.

OPCLASS
returns, in a 24-bit string, the class of the operator.

OPID
returns the 3-bytes operator identifier

OPSECURITY
returns the opkey of the operator.

TERMCODE
returns a 2-bytes code giving the type and model of the terminal. Supported only for compatibility reasons, returns:
- 0xC000 for APPC sessions
- 0x91F2 for standard terminals

USERID
returns the 8-bytes identifier of the user currently signed-on.

NETNAME
returns the 8-bytes identifier of the terminal device.

QNAME
returns the 4-bytes identifier of the transient data queue that initiated the task by trigger level.

MAPHEIGHT
returns a halfword binary indicating the height of the most recently positioned map.

MAPWIDTH
returns a halfword binary indicating the width of the most recently positioned map.

MAPLINE
returns a halfword binary indicating number of the line on the display containing the origin of the most recently positioned map.

STARTCODE
returns a 2-bytes identifier indicating how the task was started. It can have the following values:
- D - a distributed program link (DPL), without SYNCONRETURN option.
- DS - a distributed program link (DPL), with SYNCONRETURN option.
- QD - Transient data trigger level
- S - START without data
- SD - START with data
- TD - terminal input

**ABCODE**

returns a 4-bytes identifier of the current abend code.

**ABPROGRAM**

returns an 8-character name of the failing program for the latest abend.

If the abend originally occurred in a DPL server program running in a remote system, ABPROGRAM returns the DPL server program name.

This field is set to binary zeros if it is not possible to determine the failing program at the time of the abend.

**APPLID**

returns the 8-bytes application name of the XCICS region owning the transaction

**SYSID**

returns the 4-bytes identifier of the XCICS region owning the transaction

**PROGRAM**

returns the 8-byte identifier of the currently running program.

**PRINSYSID**

returns the 4-bytes identifier, as defined in "define connection", of the remote region that initiated the APPC session.

**INVOKINGPROG**

returns the 8-byte identifier of the program that used LINK or XCTL to transfer the control to the current program.

If the command is issued at highest level, 8 blanks are returned.
3.2.2.3 ASKTIME

This command is used to update the date and time-of-day-clock and the fields EIBDATE and EIBTIME. These two fields contain initially the date and time when the task started and then are updates after execution of every EXEC CICS command.

\[
\text{ASKTIME ABSTIME(time)}
\]

**Options**

**ABSTIME(TIME)**

specifies the data area for the time, in packed decimal, since 00:00 on 1 January 1900 (in milliseconds rounded to the nearest hundredth of a second). You can use FORMATTIME to change the data into other familiar formats.
3.2.2.4 ABEND

This command is used to request a task to be terminated abnormally.

<table>
<thead>
<tr>
<th>ABEND</th>
<th>ABCODE( abcode)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ CANCEL ]</td>
</tr>
<tr>
<td></td>
<td>[ NODUMP ]</td>
</tr>
</tbody>
</table>

**Options**

**ABCODE(ABCODE)**

specifies that main storage related to the task that is being terminated is to be dumped. The ABCODE is used as a transaction dumpcode to identify the dump. The name should have four characters and should not contain any leading or imbedded blanks. If ABCODE is not coded, the ABEND is identified by ???? but no dump is taken.

**CANCEL**

specifies that exits established by HANDLE ABEND commands are to be ignored. An ABEND CANCEL command cancels all exits at any level in the task (and terminates the task abnormally).

**NODUMP**

allows you to request an abend without causing a dump to be taken.
3.2.2.5 BIF DEEDIT

It specifies that alphabetic and special characters are removed from data field, and the remaining digits right-aligned and padded to the left with zeros as necessary.

BIF DEEDIT  FIELD(field)  
LENGTH(len)

**Options**

FIELD(FIELD)

specifies the field to be edited.

LENGTH(LEN)

specifies the field length in bytes.
3.2.2.6 CANCEL

CANCEL cancels a previously issued DELAY, POST, or START command.

```
CANCEL  REQID(reqid)
       [  TRANSID(transid)  ]
```

**Options**

**REQUID (REQUID)**

specifies a name (1-8 characters), which should be unique, to identify a command. This name is used as a temporary storage identifier. The temporary storage queue thus identified must be defined as a local queue on the CICS system where the CANCEL command is processed.

**TRANSID (TRANSID)**

specifies the symbolic identifier (1-4 characters) of a transaction to be used to determine where the CANCEL command is to be executed.
3.2.2.7 CONNECT PROCESS

CONNECT PROCESS allows an application to specify a process name and synchronization level to be passed to CICS and used when the remote partner is attached.

<table>
<thead>
<tr>
<th>CONNECT PROCESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ CONVID(name) ]</td>
</tr>
<tr>
<td>[ SESSION(name) ]</td>
</tr>
<tr>
<td>[ PROCNAME(data-area) ]</td>
</tr>
<tr>
<td>[ PROCLENGTH(data-value) ]</td>
</tr>
<tr>
<td>[ PARTNER(name) ]</td>
</tr>
<tr>
<td>[ SYNCLEVEL(data-value) ]</td>
</tr>
<tr>
<td>[ STATE(cvda) ]</td>
</tr>
</tbody>
</table>

**Description**

**CONVID(NAME)**

identifies the conversation to which the command relates. The 4-character name specifies the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB.

For compatibility with earlier releases, SESSION is accepted as a synonym for CONVID. New programs should use CONVID.

**PARTNER(NAME)**

specifies the name (8 characters) of a set of definitions that includes the name (or extended name) of a remote partner transaction (TPNAME or XTPNAME). You can use this option as an alternative to PROCNAME and PROCLENGTH.

**PROCLENGTH(DATA-VALUE)**

specifies the length (as a halfword binary value in the range 1-64) of the name specified by the PROCNAME option.

**PROCNAME(DATA-AREA)**

specifies the partner process (that is, the transaction) to be attached in the remote system.

One byte is sufficient to identify a CICS transaction. The APPC architecture allows a range of 1-64 bytes but leaves each product free to set its own maximum. CICS complies by allowing a range of 1-64 bytes. If the remote system is CICS, this option can specify the 4-byte transaction identifier or the TPNAME value given in the relevant TRANSACTION definition. Alternatively, you can examine the full identifier by coding the user exit XZCATT.

**SESSION(NAME)**

specifies the symbolic identifier (1-4 characters) of a session TCTTE. This option specifies the alternate facility to be used.

**STATE(CVDA)**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
* PENDFREE
* PENDRECEIVE
* RECEIVE
* ROLLBACK
* SEND
* SYNCFREE
* SYNCRECEIVE
* SYNCSEND

SYNCLEVEL(DATA-VALUE)

specifies the synchronization level (halfword binary value) for the current conversation. The possible values are:

* 0 None
* 1 Confirm
* 2 Syncpoint
3.2.2.8 CONVERSE

For most terminals or logical unit types a conversational mode of communication can be used. The CONVERSE command is used for this purpose. In general, the CONVERSE command can be considered as a combination of a SEND command followed immediately by a WAIT TERMINAL command and then by a RECEIVE command.

CONVERSE FROM(area)
[ FROMLENGTH(flen) ]
[ FROMFLENGTH(flen) ]
[ INTO(iarea) | SET(ptr) ]
[ TOLENGTH(flen) ]
[ TOFLENGTH(flen) ]
[ MAXLENGTH( maxl ) ]
[ MAXLENGTH( maxl ) ]
[ TRUNCATE ]
[ STATE(cvda) ]
[ ERASE ]
[ DFRESP ]
[ CTLCHAR(c) ]
[ SESSION(session) ]
[ CONVID(convid) ]
[ TRANSLATE ]

Options

CONVID(CONVID)

identifies the conversation to which the command relates.

CTLCHAR(C)

specifies a 1-byte write control character (WCC) that controls the CONVERSE command.

DFRESP

indicates that a definite response is required when the output operation has been completed.

ERASE

specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of the screen.

FROM(AREA)

specifies the data to be written to the terminal or logical unit, or sent to the partner transaction.

FROMLENGTH(FLEN)

specifies the length, as a halfword binary value, of the data.

FROMFLENGTH(FLEN)

is a fullword alternative to FROMLENGTH.

MAXLENGTH(LMAX)

specifies the maximum amount (halfword binary value) of data that CICS is to recover in response to a CONVERSE.
(default) command.

**MAXLENGTH(LMAX)**

is a fullword alternative to MAXLENGTH.

**INTO(AREA)**

specifies the receiving field for the data read from the terminal or logical unit, or the application target data area into which data is to be received from the application program connected to the other end of the current conversation.

**SET(PTR)**

specifies the pointer reference to be set to the address of the data read from the terminal. pointer reference, unless changed by other commands or statements, is valid until the next CONVERSE (default) command or the end of task.

**TOLENGTH(LEN)**

specifies the length (halfword binary value) of the data to be received.

**TOLENGTH(LEN)**

is a fullword alternative to TOLENGTH.

**NOTRUNCATE**

specifies that, when the data available exceeds the length requested, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

**SESSION(SESSION)**

specifies the symbolic identifier (1-4 characters) of a session TCTTE. This option specifies the alternate facility to be used.

**STATE(CVDA)**

gets the state of the current conversation.

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
TRANSLATE

Only for APPC: data are automatically translated to EBCDIC whenever the remote system is defined as EBCDIC in CCT. Be careful: translation assumes the data to be converted are not aware of binary fields.
3.2.2.9 CONVERTTIME

Converts an architected date and time stamp string to the ABSTIME format. CONVERTTIME analyzes three different date and time stamp formats which are commonly used on the Internet, and converts them to the ABSTIME (absolute date and time) format.

ABSTIME format gives the time, in packed decimal, since 00:00 on 1 January 1900 (in milliseconds rounded to the nearest hundredth of a second). The FORMATTIME command can be used to change this into other formats.

The architected date and time stamp string formats recognized by the CONVERTTIME command are:

RFC 1123 format: The preferred standard format for date and time stamps for the HTTP protocol, as specified in RFC 1123. An example of a date and time stamp in this format is "Tue, 01 Apr 2003 10:01:02 GMT".

RFC 850 format: An older date and time stamp format for the Internet. An example of a date and time stamp in this format is "Tuesday, 01-Apr-03 10:01:02 GMT".

ASCtime format: A date and time stamp format output from the C ASCtime function. An example of a date and time stamp in this format is "Tue Apr 1 10:01:02 2003".

<table>
<thead>
<tr>
<th>CONVERTTIME</th>
<th>DATESTRING(data-area)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ABSTIME(data-area)</td>
</tr>
</tbody>
</table>

**Options**

**DATESTRING(DATA-AREA)**

specifies a 64-character data-area to contain the architected date and time stamp string. You can supply a string in any of the formats recognized by the command, and you do not need to specify which format is used.

**ABSTIME(DATA-AREA)**

specifies a data-area to receive the converted date and time stamp in ABSTIME format. For the format of this data-area, see the description of the ASKTIME command. If the date and time stamp was not in a recognized format, no ABSTIME is returned.
3.2.2.1 DELAY

This command is used to suspend the processing of the issuing task for a specified interval of time or until a specified time of the day. It suspends any previously initiated POST command for the task.

```
DELAY INTERVAL(interval) [ TIME(time) ]
```

**Options**

**INTERVAL(HHMMSS)**

specifies, in packed decimal format, the interval of time that is to elapse from the time when the DELAY command is issued. The mm and ss are in the range 0-59. The time specified is added to the current clock time by CICS when the command is executed to calculate the expiration time.

**TIME(HHMMSS)**

specifies, in packed decimal format, the time when the task should resume processing.
### 3.2.2.1 DELETE

You use the DELETE command to delete a record from a KSDS or RRDS data set on a local or remote system.

You must identify the record to be deleted in the RIDFLD option.

You can also delete a record that has been retrieved for update, instead of rewriting it, by this command. In this case you must not specify the RIDFLD option.

```
DELETE  DATASET(dsname)
    [ RIDFLD(ridfld) ]
    [ RBA | RRN ]
    [ SYSID(sysid) ]
    [ KEYLENGTH(keylen) ]
    [ GENERIC ]
```

#### Options

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
3.2.2.1 DELETEQ TD

This command is used to delete all of the transient data associated with a particular intrapartition destination (queue).

```
DELETEQ TD QUEUE(qname)
   [ SYSID(systemname) ]
```

**Options**

**QUEUE(QNAME)**

specifies the symbolic name (1-4 alphanumeric characters) of the queue to be deleted to. The named queue must have been defined to CICS.

**SYSID(SYSTEMNAME)**

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.1 DELETEQ TS

This command is used to delete all temporary data associated with a temporary storage queue.

```
DELETEQ TS [ QUEUE(qname) | QNAME(qname) ]
[ SYSID(systemname) ]
```

**Options**

QUEUE(QNAME) | QNAME(QNAME)

specifies the symbolic name (1-8 alphanumeric characters) of the queue to be deleted. The name must be unique within the XCICS system.

SYSID(SYSTEMNAME)

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.1 DEQ

DEQ causes a resource currently enqueued on by the task to be released for use by other tasks.

\[
\begin{array}{|c|c|}
\hline
\text{DEQ} & \text{RESOURCE( area)} \\
& \text{LENGTH( len) } \\
\hline
\end{array}
\]

**Options**

**LENGTH(LEN)**

- specifies as a halfword binary value the length of the resource to be dequeued from.

**RESOURCE(AREA)**

- specifies either an area whose address represents the resource to be dequeued from, or a variable that contains the resource (an employee name, for example). In the latter case, you must use the LENGTH option.
3.2.2.1 DOCUMENT CREATE

DOCUMENT CREATE signals the start of the document creation process. The document being created can be an empty document, or it can be based on an existing document, a template, or data contained in an application buffer.

```
DOCUMENT CREATE
  DOCTOKEN(data-area)
  [[ FROM(data-area) | TEXT(data-area) | BINARY(data-area) ] LENGTH(data-area) ]
  [ FROMDOC(data-area) ] [ TEMPLATE(name) ]
  [ SYMBOOLLIST(data-area) LISTLENGTH(data-value) ]
  [ DELIMITER(data-value) ]
  [ DOCSIZE(data-area) ]
  [ HOSTCODEPAGE(name) ]
```

**Options**

**BINARY(DATA-AREA)**

specifies a buffer of data which is to be used as the contents of the new document being created. The data is copied unchanged to the document content and no attempt is made to parse the data for symbol substitution. The purpose of the BINARY option is to allow the application to insert blocks of data that must not undergo conversion to a client code page when the data is sent. Use the LENGTH option to specify the length of this buffer.

**DELIMITER(DATA-VALUE)**

specifies an optional 1-byte value used to delimit symbol name-value pairs in the SYMBOOLLIST buffer. If this option is not specified, the value defaults to an ampersand. There are several disallowed DELIMITER values, all of which cause an INVREQ condition if used. The disallowed values are:

- null (binary X'00'
- shift in (binary X'0E'
- shift out (binary X'0F'
- space (binary X'40'
- plus sign (binary X'4E'
- colon (binary X'7A'
- equals (binary X'7E'
- percent sign (binary X'6C'
- backslash (binary X'E0')

If this option is used, the application must ensure that the DELIMITER does not appear in any symbol value in the SYMBOOLLIST buffer. For this reason, the application should not use alphanumeric and other printable characters as the DELIMITER value.

**DOCSIZE(DATA-AREA)**

specifies a binary fullword area that will be updated with the current size of the document in bytes. This is the maximum size of the buffer needed to contain a copy of the document when a RETRIEVE command is issued.

**DOCTOKEN(DATA-AREA)**

specifies a data area to contain the symbolic name of the document. The area must be 16 bytes in length and will be set to an XCICS-generated name by which the document can be referred to in later commands.
FROM(DATA-AREA) specifies that data supplied by the application is to be used to create the contents of the new document. The data content could be a template or a document which was created and retrieved. If the data is a template, symbol substitution will take place where the symbols exist in the symbol table. If the data is a previously retrieved document, the conversion and bookmark tags which were inserted during retrieval will be removed from the content and stored in the internal format required by the API commands. Note that symbol substitution will not be attempted on any unresolved symbols contained in a retrieved document. Use the LENGTH option to specify the length of this buffer.

FROMDOC(DATA-AREA) specifies the symbolic name (see the DOCTOKEN option) of a document whose contents are to be copied to the new document being created. All bookmark and conversion tags are copied to the new document. The symbol table will not be copied.

HOSTCODEPAGE(NAME) This option has no effect. Supported for compatibility reason.

LENGTH(DATA-VALUE) specifies the length, as a fullword binary value, of the buffer containing the TEXT, BINARY or FROM data.

LISTLENGTH(DATA-VALUE) specifies the length, as a fullword binary value, of the symbol list.

SYMBOLLIST(DATA-AREA) specifies a buffer which contains a symbol list. Use the LISTLENGTH option to specify the length of this buffer. A symbol list is a character string consisting of one or more symbol definitions separated by ampersands. Each symbol definition consists of a name, an equals sign, and a value. By default, symbols in the symbol list are separated by the & character, but you can override this by using the DELIMITER keyword to specify a different symbol separator. Here is an example of a symbol list:

applid=IYCQ&jobname=test

The following rules apply when setting symbols using a SYMBOLLIST:

- The name is case-sensitive. It may only contain uppercase and lowercase letters, numbers, and the special characters dollar ('$'), underscore ('_'), hyphen ('-'), number sign ('#'), period ('.') and at sign ('@'). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters. Unlike the symbols in the template, the names in the symbol list have neither an ampersand at the beginning, nor a semicolon at the end. For example, the symbol &mytitle; in the template corresponds to the name mytitle in the symbol list.

- The values in the symbol list can contain any characters. However, special coding is required if you need to include the following characters in symbol values in the symbol list:
  - The character that you have used as the symbol separator (which defaults to an ampersand, but can be overridden by use of the DELIMITER option).
  - The plus sign and the percent sign.

You can use the percent sign, followed by two characters that are hexadecimal digits (that is, 0-9, a-f, and A-F), to include characters such as these that have a special meaning. When the value is put into the symbol table, a percent sign and the two hexadecimal digits following it are interpreted as the EBCDIC equivalent of the single ASCII character denoted by the two digits. %2B produces a plus sign, %25 produces a percent sign, and %28 produces an ampersand. If the characters following the percent sign are not two valid hexadecimal digits, the percent sign and the following characters are put into the symbol table as they appear in the symbol list.

If you prefer not to use this special coding, you can specify the UNESCAPED option. When you...
specify this option, no conversion takes place, and the symbol values are put into the symbol table exactly as you entered them. However, the UNESCAPED option does not allow you to include the character that you have used as the symbol separator within a symbol value in a symbol list. If you want to use the UNESCAPED option, choose a symbol separator that will never be used within a symbol value.

- If you want to include spaces in a value, XCICS allows you to use the space character, a plus sign, or an escape sequence (%20). However, you cannot use a plus sign or escape sequence when the UNESCAPED option is used. In this case, you must only use a space character to indicate a space.

**TEMPLATE(NAME)**

specifies the 48-byte name of a template. The template must be defined to XCICS using the “define template” configuration statement. If the name is shorter than 48 bytes, it must be padded on the right with blanks.

**Note**

If you insert a template before the symbols contained in it are set, the symbols will never be substituted. This can occur if you create a document from a template without specifying a symbol list.

**TEXT(DATA-AREA)**

specifies a buffer of data which is to be used as the contents of the new document being created. The data is copied unchanged to the document content and no attempt is made to parse the data for symbol substitution. The data will be marked as requiring conversion to the client code page when the document is sent. Use the LENGTH option to specify the length of this buffer.

**UNESCAPED**

prevents XCICS from unescaping symbol values contained in the SYMBOLLIST buffer. If this option is used, plus signs are not converted to spaces, and sequences such %2B are not converted to single byte values.

The UNESCAPED option does not allow you to include the character that you have used as the symbol separator within a symbol value in a symbol list. If you want to use the UNESCAPED option, choose a symbol separator that will never be used within a symbol value.

**Conditions**

**INVREQ**

RESP2 values are:

1

The retrieved document specified on the FROM option is not in a valid RETRIEVE format.

**NOTFND**

RESP2 values:

2

The document specified on the FROMDOC option could not be found or was named incorrectly.

3

The template specified on the TEMPLATE option could not be found or was named incorrectly.

7

The host codepage specified on the HOSTCODEPAGE option could not be found or was named incorrectly.
The value specified for DELIMITER is not valid.

**SYMBOLERR**

a symbol specified in the symbol list does not conform to the naming rules for symbols. RESP2 contains the offset of the symbol in the list.

**TEMPLATERR**

an invalid #set, #include or #echo command has been encountered while processing the supplied template data. RESP2 contains the offset of the invalid command.
3.2.2.1 DOCUMENT INSERT

DOCUMENT INSERT allows the application to insert document objects at insertion points within the document. The insertion points (bookmarks) define relative positions within the document. Bookmarks must be defined before being referenced. Data is always inserted after the position identified by the bookmark.

### Options

**AT(NAME)**

specifies the 16-byte symbolic name of a bookmark which identifies the position of the insertion point in the document. Data is inserted after the bookmark, and any data following the bookmark is shifted down. The application can use a combination of the AT and TO options to perform an overlay operation. If the AT operand is not specified, the data is inserted at the end of the document. A pre-defined bookmark of TOP is provided to allow the application to insert data at the beginning of the document.

**BINARY(DATA-AREA)**

specifies a buffer of data to be inserted into the document. The data is copied unchanged to the insertion point in the document, and no attempt is made to parse the data for symbol substitution. The BINARY option allows the application to insert blocks of data that must not undergo conversion to a client code page when the data is sent. Use the LENGTH option to specify the length of this buffer.

**BOOKMARK(NAME)**

specifies a bookmark to be inserted into the document. A bookmark is a symbolic name which identifies an insertion point in the document. The name can be up to 16 characters in length, and must not contain any imbedded spaces.

**DOCSIZE(DATA-VALUE)**

specifies a binary fullword area to be updated with the current size of the document in bytes. This is the maximum size of the buffer needed to contain a copy of the document when a RETRIEVE command is issued.

**DOCTOKEN(DATA-AREA)**

specifies the 16-byte symbolic name of the document into which data is to be inserted.

**FROM(DATA-AREA)**

specifies that a buffer of data supplied by the application is to be inserted into the document. The data content can be a template or a document that was previously created and retrieved. If the data is a template, symbol substitution takes place where the symbols exist in the symbol table. If the data is a previously retrieved document, the conversion and bookmark tags which were inserted during the retrieval will be removed from the content and stored in the internal form required by the API commands. Note that symbol substitution will not be attempted on any unresolved symbols contained in a retrieved document. Use the LENGTH option to specify the length of this
buffer.

**FROMDOC(DATA-AREA)**

specifies the symbolic name of a document (see the DOCTOKEN option) whose contents are copied to the insertion point of the target document. All bookmarks and conversion tags are copied to the target document. The symbol table is not copied.

**HOSTCODEPAGE(NAME)**

This option has no effect. Supported for compatibility reason.

**LENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the buffer containing the TEXT, BINARY or FROM data.

When the DOCUMENT INSERT command follows a DOCUMENT RETRIEVE command, without the use of the DATAONLY option, and the retrieved document is being inserted using the FROM option, the LENGTH specified must be equal to the length of the retrieved document.

**SYMBOL(NAME)**

specifies the 32-byte name of a symbol in the symbol table. The data associated with the symbol in the symbol table is inserted, but not the symbol itself. Note that when data associated with a symbol has been inserted into a document, you cannot change that data in the document that is being composed. If you set a different value for the symbol, the new value will be used the next time that symbol is inserted into a document. Your change will not affect the value that has already been inserted into the document.

**TEMPLATE(NAME)**

specifies the 48-byte name of a template. The template must be defined to XCICS using the "define template" configuration statement. If the name is less than 48 bytes, it must be padded on the right with blanks. The current values of any symbols are substituted into the template.

**Note**

When a template containing symbols has been inserted into a document, you cannot change the substituted values of those symbols in the document that is being composed. If you set different values for the symbols, the new values will be used the next time that the template is inserted into a document. Your changes will not affect the values that have already been inserted into the document.

**TEXT(DATA-AREA)**

specifies a buffer of data to be inserted into the document. The data is copied unchanged to the insertion point in the document, and no attempt is made to parse the data for symbol substitution. When the document is sent, it is marked as requiring conversion to the client code page. Use the LENGTH option to specify the length of this buffer.

**TO(NAME)**

specifies the symbolic name of a bookmark identifying the end position of an overlay operation. Data between the bookmarks identified by the AT and TO operands is deleted, and new data is inserted in its place. It is possible to delete data between two bookmarks by specifying a null string on the TEXT or BINARY option with a LENGTH of zero.

**Conditions**

**DUPREC**

the bookmark has already been defined.

**INVREQ**
RESP2 values are:

0
The bookmark specified on the TO option appears before the bookmark specified on the AT bookmark.

1
The retrieved document specified on the FROM option is not in a valid RETRIEVE format.

2
The bookmark name on the BOOKMARK option is invalid.

NOTFND
one of the following documents or templates could not be found, or its name was incorrect.
RESP2 values:

1
The document specified on the DOCUMENT option.

2
The document specified on the FROMDOC option.

3
The template specified on the TEMPLATE option.

4
The document specified on the SYMBOL option.

5
The document specified on the AT option.

6
The document specified on the TO option.

7
The document specified on the HOSTCODEPAGE option.

TEMPLATERR
an invalid #set, #include or #echo command has been encountered while processing the supplied template data. RESP2 contains either a zero (if the maximum of 32 levels of embedded templates is exceeded), or the offset of the invalid command.
3.2.2.1 DOCUMENT RETRIEVE

DOCUMENT RETRIEVE allows the application to obtain a copy of the document in its own buffer, which it can then manipulate directly. The document is managed by XCICS, and the application does not have direct access to the buffer containing the contents of the document. The document exists only for the duration of the current transaction, so the application must retrieve the document and store it if the document is to exist over transaction boundaries. The retrieved document can be used as a basis for a new document by using the FROM option of the DOCUMENT CREATE command.

```
DOCUMENT RETRIEVE
  DOCTOKEN(data-area)
  INTO(data-area)
  LENGTH(data-value)
  [ MAXLENGTH(data-value)
  [ CLNTCODEPAGE(name)]
  [ DATAONLY]
```

**Options**

**CLNTCODEPAGE(NAME)**

This option has no effect. Supported for compatibility reason.

**DATAONLY**

specifies that the data should be retrieved without any imbedded tags.

**DOCTOKEN(DATA-AREA)**

specifies the 16-byte symbolic name of the document to be retrieved.

**INTO(DATA-AREA)**

specifies the buffer that is to contain the copy of the document content.

**LENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the amount of data being returned to the application.

**MAXLENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the maximum amount of data the buffer can receive.

**Conditions**

**LENGERR**

RESP2 values:

1

MAXLENGTH is less than or equal to zero. The document is truncated.

2

The length of the receiving buffer is zero, or is too short to contain the document contents. The document is truncated.
NOTFND

RESP2 values:

1

The document has not been created, or the name is incorrectly specified.

7

The specified client codepage can not be found.
3.2.2.1 DOCUMENT SET

DOCUMENT SET allows the application to add symbols and their associated values to the symbol table. If the symbol being added already exists in the table, it is replaced by the new definition.

<table>
<thead>
<tr>
<th>DOCUMENT</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCTOKEN(data-area)</td>
<td>SYMBOL(name) VALUE(data-area)</td>
</tr>
</tbody>
</table>

**Options**

**DELIMITER(DATA-VALUE)**

specifies an optional 1-byte value used to delimit symbol name-value pairs in the SYMBOLLIST buffer. If this option is not specified, the value defaults to an ampersand. There are several disallowed DELIMITER values, all of which cause an INVREQ condition if used. The disallowed values are:

- null (binary X'00')
- shift in (binary X'0E')
- shift out (binary X'0F')
- space (binary X'40')
- plus sign (binary X'4E')
- colon (binary X'7A')
- equals (binary X'7E')
- percent sign (binary X'6C')
- backslash (binary X'E0')

If this option is used, the application must ensure that the DELIMITER does not appear in any symbol value in the SYMBOLLIST buffer. For this reason, the application should not use alphanumeric and other printable characters as the DELIMITER value.

**DOCTOKEN(DATA-AREA)**

specifies the 16-byte symbolic name of the document that owns the symbol table.

**LENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the buffer containing the data value associated with the symbol, or the length of the buffer containing the symbol list when the SYMBOLLIST option is used.

**SYMBOL(NAME)**

specifies the name of the symbol that is to be added to the table. The name can be 1 to 32 characters in length with no embedded spaces. The name of the symbol must contain only uppercase and lowercase letters, numbers and the special characters dollar ("$"), underscore ("_"), hyphen ("-"), number sign ("#"), period ("."), and at sign ("@"). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters. If you want to define more than one symbol in the same command, use the SYMBOLLIST option instead.

**SYMBOLLIST(DATA-AREA)**

specifies a buffer which contains a symbol list. Use the LISTLENGTH option to specify the length of this buffer. A
symbol list is a character string consisting of one or more symbol definitions separated by ampersands. Each symbol definition consists of a name, an equals sign, and a value. By default, symbols in the symbol list are separated by the & character, but you can override this by using the DELIMITER keyword to specify a different symbol separator. Here is an example of a symbol list:

\[ \text{applid=IYCQ&jobname=test} \]

The following rules apply when setting symbols using a SYMBOLLIST:

- The name is case-sensitive. It may only contain uppercase and lowercase letters, numbers, and the special characters dollar ($), underscore (_), hyphen (-), number sign (#), period (.), and at sign (@). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters. Unlike the symbols in the template, the names in the symbol list have neither an ampersand at the beginning, nor a semicolon at the end. For example, the symbol &mytitle; in the template corresponds to the name mytitle in the symbol list.

- The values in the symbol list can contain any characters. However, special coding is required if you need to include the following characters in symbol values in the symbol list:
  - The character that you have used as the symbol separator (which defaults to an ampersand, but can be overridden by use of the DELIMITER option).
  - The plus sign and the percent sign.

You can use the percent sign, followed by two characters that are hexadecimal digits (that is, 0-9, a-f, and A-F), to include characters such as these that have a special meaning. When the value is put into the symbol table, a percent sign and the two hexadecimal digits following it are interpreted as the EBCDIC equivalent of the single ASCII character denoted by the two digits. %2B produces a plus sign, %25 produces a percent sign, and %26 produces an ampersand. If the characters following the percent sign are not two valid hexadecimal digits, the percent sign and the following characters are put into the symbol table as they appear in the symbol list. If the UNESCAPED option is used, no conversion takes place, and all the characters are put into the symbol table as they appear in the symbol list.

- If you want to include spaces in a value, XCICS allows you to use the space character, a plus sign, or an escape sequence (%20). However, you cannot use a plus sign or escape sequence to indicate a space character when the UNESCAPED option is used. In this case, you must only use a space character to indicate a space.

UNESCAPED

prevents XCICS from unescaping symbol values contained in the SYMBOLLIST buffer. If this option is used, plus signs are not converted to spaces, and sequences such as %2B are not converted to single byte values.

The UNESCAPED option does not allow you to include the character that you have used as the symbol separator within a symbol value in a symbol list. If you want to use the UNESCAPED option, choose a symbol separator that will never be used within a symbol value. Alternatively, you can use the SYMBOL and VALUE options to specify symbol values that contain the character you have used as the symbol separator, because the symbol separator has no special meaning when used in the VALUE option.

VALUE(DATA-AREA)

specifies an area containing the value to be associated with the SYMBOL.

The rules for including spaces in a symbol value in a symbol list also apply to the VALUE option: you can use a simple space character or a plus sign, unless the UNESCAPED option has been specified, in which case you must use a space character. Also, the special coding that is required to include a plus sign or percent sign in symbol lists is similarly required in the VALUE option, unless the UNESCAPED option has been specified. However, ampersands, or any other character that you have specified as a symbol separator for the symbol list, have no special significance when used in the VALUE option.
Conditions

INVREQ

RESP2 values:

8

The value specified for DELIMITER is not valid.

NOTFND

RESP2 values:

1

The document has not been created or the name is incorrectly specified.

SYMBOLERR

a symbol name is invalid. RESP2 values:

0

SYMBOLLIST was not used.

OFFSET

RESP2 contains the offset of the invalid symbol in the list.
3.2.2.1 DUMP

The XCICS dump control program allows the specification of areas of main storage to be dumped, by means of the DUMP command, onto a sequential data set, which can be analyzed afterwards for diagnose. This command is used to dump any or all of the main storage areas related to a task.

<table>
<thead>
<tr>
<th>DUMP [ TRANSACTION ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUMP CODE (dcode)</td>
</tr>
<tr>
<td>[ FROM (area) ]</td>
</tr>
<tr>
<td>[ LENGTH (len) ]</td>
</tr>
</tbody>
</table>

**Options**

**LENGTH(LEN)**

specifies the length (halfword binary) of the data area specified in the FROM option.

**FROM(AREA)**

dumps the specified data area, which must be a valid area; that is, storage allocated by the operating system within the CICS region.
3.2.2.2 ENDBR

Ends a browse on a data set on a local or remote system.

```plaintext
ENDBR      DATASET(dsname)  
            [ REQID(reqid) ]  
            [ SYSID(systemname) ]
```

**Options**

**DATASET(DNAME)**

specifies the name of the file being browsed.

**REQID(REQID)**

specifies a unique (halfword binary value) request identifier for a browse, used to control multiple browse operations on a data set. If this option is not specified, a default value of zero is assumed.

**SYSID(SYSTEMNAME)**

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.2 ENTER
Write a trace entry.

```
ENTER TRACENUM( data-value )
[ FROM( data-area )]
[ FROMLENGTH( data-value )]
[ RESOURCE( data-area )]
```

**Options**

**TRACENUM(DATA-VALUE)**

specifies an halfword binary identifier of the trace. TRACEID is also accepted as synonym of TRACENUM

**FROM(DATA-AREA)**

specifies an area whose contents are to be logged

**FROMLENGTH(DATA-VALUE)**

specifies an halfword binary containing the length of the trace data.

**RESOURCE(DATA-AREA)**

specifies an 8-bytes generic resource identifier to be logged
3.2.2.2 ENQ

Schedule use of a resource by a task (enqueue). If a task enqueues on a resource but does not dequeue from it, CICS automatically releases the resource during syncpoint processing, or when the task is terminated.

<table>
<thead>
<tr>
<th>ENQ</th>
<th>RESOURCE(area)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ LENGTH(len) ]</td>
</tr>
<tr>
<td></td>
<td>[ NOSUSPEND ]</td>
</tr>
<tr>
<td></td>
<td>[ TASK ]</td>
</tr>
</tbody>
</table>

**Options**

**LENGTH(LEN)**

specifies as a halfword binary value the length of the resource to be enqueued on.

**NOSUSPEND**

specifies that the application program is not to be suspended if the resource on the ENQ command is unavailable, but the ENQBUSY condition occurs.

**RESOURCE(AREA)**

specifies either an area whose address represents the resource to be enqueued from, or a variable that contains the resource (an employee name, for example). In the latter case, you must use the LENGTH option.

**TASK**

The duration of the ENQ is a task.
3.2.2.2 EXTRACT ATTRIBUTES

EXTRACT ATTRIBUTES extracts conversation state information for APPC mapped conversations.

```
EXTRACT ATTRIBUTES STATE(cvda) [ CONVID(name) ]
```

**Options**

**CONVID(NAME)**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal session (returned by a previously executed ASSIGN command).

**STATE(CVDA)**

gets the state of the transaction program. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
### EXTRACT CERTIFICATE

EXTRACT CERTIFICATE allows the application to obtain information from the X.509 certificate that was received from a client during a Secure Sockets Layer (SSL) handshake over a TCPIP SERVICE that specified SSL=YES and auth=certificate. The certificate contains fields that identify the owner (or subject) of the certificate, and fields that identify the Certificate Authority that issued the certificate. You can select the fields that you require by specifying the OWNER or ISSUER option. You cannot retrieve both OWNER and ISSUER fields with one command.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CERTIFICATE(ptr-ref)</code></td>
<td>specifies a pointer reference to be set to the address of the full binary certificate received from the client. The pointer reference is valid until the next CICS command or the end of task.</td>
</tr>
<tr>
<td><code>COMMONNAME(ptr-ref)</code></td>
<td>specifies a pointer reference to be set to the common name from the client certificate. The pointer reference is valid until the next CICS command or the end of task.</td>
</tr>
<tr>
<td><code>COMMONNAMLEN(data-area)</code></td>
<td>specifies a fullword binary data area to be set to the length of the common name from the client certificate.</td>
</tr>
<tr>
<td><code>COUNTRY(ptr-ref)</code></td>
<td>specifies a pointer reference to be set to the address of the country from the client certificate. The pointer reference is valid until the next CICS command or the end of task.</td>
</tr>
<tr>
<td><code>COUNTRYLEN(data-area)</code></td>
<td>specifies a fullword binary data area to be set to the length of the country from the client certificate.</td>
</tr>
<tr>
<td><code>ISSUER</code></td>
<td>indicates that the values returned by this command refer to the Certificate Authority that issued this certificate.</td>
</tr>
</tbody>
</table>
LENGTH(DATA-AREA)
specifies a fullword binary data area to be set to the length of the body of the client certificate.

LOCALITY(PTR-REF)
specifies a pointer reference to be set to the address of the locality from the client certificate. The pointer reference is valid until the next CICS command or the end of task.

LOCALITYLEN(DATA-AREA)
specifies a fullword binary data area to be set to the length of the locality from the client certificate.

ORGANIZATION(PTR-REF)
specifies a pointer reference to be set to the address of the organization from the client certificate. The pointer reference is valid until the next CICS command or the end of task.

ORGANIZATIONLEN(DATA-AREA)
specifies a fullword binary data area to be set to the length of the organization from the client certificate.

ORGUNIT(PTR-REF)
specifies a pointer reference to be set to the address of the organization unit from the client certificate. The pointer reference is valid until the next CICS command or the end of task.

ORGUNITLEN(DATA-AREA)
specifies a fullword binary data area to be set to the length of the organization unit from the client certificate.

OWNER
indicates that the values returned by this command refer to the owner of the certificate.

SERIALNUM(PTR-REF)
specifies a pointer reference to be set to the address of the serial number of the certificate assigned by the certificate issuer. The pointer reference is valid until the next CICS command or the end of task.

SERIALNUMLEN(DATA-AREA)
specifies a fullword binary data area to be set to the length of the serial number.

STATE(PTR-REF)
specifies a pointer reference to be set to the address of the state or province from the client certificate. The pointer reference is valid until the next CICS command or the end of task.

STATELEN(DATA-AREA)
specifies a fullword binary data area to be set to the length of the state or province from the client certificate.

USERID(PTR-REF)
specifies a pointer reference to be set to the userid connected with the client certificate. The pointer reference is valid until the next CICS command or the end of task.
Conditions

INVREQ
occurs for the following conditions:

- the command is being issued in a non-CICS Web Interface application.
- the command is being issued for a non-HTTP request.
- if an error occurs retrieving the certificate data from CICS intermediate storage.

LENGERR
the string being extracted is longer than the length specified for one of the options.
3.2.2.2 EXTRACT PROCESS

EXTRACT PROCESS lets an application program access conversation-related data, specified to CICS when the program is attached. The attach receiver does not have to execute an EXTRACT PROCESS command unless it requires this information.

The EXTRACT PROCESS command is valid only on an APPC conversation that is the principal facility for the task.

```
EXTRACT PROCESS
  [ PROCNAME(data-area) ]
  [ MAXPROCLEN(data-value) ]
  [ CONVID(name) ]
  [ SYNCLEVEL(data-area) ]
  [ PROCLNGTH(data-area) ]
```

**Options**

CONVID(NAME)

identifies the conversation to which the command relates. The 4-character name identifies the token representing the principal session (EIBTRMID).

For compatibility with earlier releases, SESSION is accepted as a synonym for CONVID. New programs should use CONVID.

If CONVID and SESSION are both omitted, the principal facility for the task is used by default.

MAXPROCLEN(DATA-VALUE)

specifies the buffer length of PROCNAME. If MAXPROCLEN is not specified, the buffer is assumed to have 32 bytes.

PROCLNGTH(DATA-AREA)

specifies a halfword data area that is set by CICS to the length of the process name. If PROCNAME is specified, this option must be specified.

PROCNAME(DATA-AREA)

specifies the data area to receive the process name specified by the remote system that caused the task to start. The data area can be 1-64 bytes long. The process name is padded on the right with blanks if it is too short. The PROCNAME data area should not be shorter than the MAXPROCLEN value.

SYNCLEVEL(DATA-AREA)

specifies a halfword data area that is set by CICS to the SYNCLEVEL value.
3.2.2.2 EXTRACT TCPIP

Obtain information about TCPIP characteristics of the current transaction.

```
EXTRACT TCPIP
  [ CLIENTNAME( cname ) ]
  [ CNAMELENGTH( cnamelen ) ]
  [ CLIENTADDR( caddrtext ) ]
  [ CADDRLENGTH( addrlen ) ]
  [ CLIENTADDRNU( caddrnum ) ]
  [ SERVERNAME( server ) ]
  [ SNAMELENGTH( serverlen ) ]
  [ SERVERADDR( servertext ) ]
  [ SADDRLENGTH( saddrlen ) ]
  [ SERVERADDRNU( serveraddnum ) ]
  [ PORTNUMBER( port ) ]
  [ PORTNUMBERNU( portnumnum ) ]
  [ TCPIPSERVICE( tcpservice ) ]
  [ PRIVACY( prvlen ) ]
  [ SSLTYPE( ssltype ) ]
  [ AUTHENTICATE( authtype ) ]
```

**Options**

**AUTHENTICATE(AUTTYPE)**

returns a CVDA indicating the authentication requested for the client using this transaction. Possible values are:

- AUTOAUTH
- AUTOREGISTER
- BASICAUTH
- CERTIFICATE
- NOAUTHENTIC

**CADDRLENGTH(ADDRLEN)**

specifies the length of the buffer supplied on the CLIENTADDR option, and is set to the actual length of the data returned to the application.

**CLIENTADDR(CADDRTEXT)**

specifies a buffer to contain the client's TCP/IP address.

**CLIENTADDRNU(CADDNUM)**

specifies a fullword binary field containing the client's TCP/IP address in binary form.

**CLIENTNAME(CNAME)**

specifies a buffer to contain the client's name as known by the Domain Name Server.

**CNAMELENGTH(CNAMELEN)**

specifies the length of the buffer supplied on the CLIENTNAME option, and is set to the actual length of the data returned to the application, or zero if the client's name is not known to the domain name server.
PORTNUMBER(PORT)

Specifies a 5-character field to contain the port number associated with this transaction in character form.

PORTNUMBERNU(PORTNUMNUM)

Fullword field to contain the port number associated with this transaction in binary form.

SADDRLENGTH(SADDRLEN)

specifies the length of the buffer supplied on the SERVERADDR option, and is set to the actual length of the data returned to the application.

SERVERADDR(SERVERTEXT)

specifies a buffer to contain the server's TCP/IP address in dotted decimal character form (nnn.nnn.nnn.nnn).

SERVERADDRNU(SERVERADDNUM)

specifies a fullword binary field containing the server's TCP/IP address in binary form.

SERVERNAME(SERVER)

specifies a buffer to contain the server's name as known by the Domain Name Server.

SNAMELENGTH(SERVERLEN)

specifies the length of the buffer supplied on the SERVERNAME option, and is set to the actual length of the data returned to the application.

SSLTYPE(SSLTYPE)

Returns a CVDA indicating whether the Secure Sockets Layer (SSL) is being used to secure communications for this transaction. Possible values are:

- SSL
- NOSSL
- CLIENTAUTH

TCPIPSERVICE

An 8-byte field to contain the name of the TCPIPSERVICE associated with this transaction.
3.2.2.2 FREEMAIN

This command is used to release main storage previously acquired by a GETMAIN command.

```
FREEMAIN [ DATA(area) | DATAPORTER(ptr) ]
```

**Options**

**DATA(AREA)**

specifies the data area of main storage to be released.

**DATAPORTER(PTR)**

specifies the address of the main storage to be released. This option is an alternative to the DATA option, and specifies the pointer reference that was returned by a GETMAIN command using the SET option.
3.2.2.2 FORMATTIME

This command is used to transform the absolute date and/or time into any of a variety of formats, as described in the list of options.

```
FORMATTIME ABSTIME(time)
   [ YYDDD(area) ]
   [ YYMMDD(area) ]
   [ YYDDMM(area) ]
   [ DDMMYY(area) ]
   [ MMDDYY(area) ]
   [ DATE(area) ]
   [ DATEFORM(area) ]
   [ DATESEP(area) ]
   [ DAYCOUNT(area) ]
   [ DAYOFWEEK(area) ]
   [ DAYOFMONTH(area) ]
   [ YEAR(area) ]
   [ TIME(area) ]
   [ TIMESEP(area) ]
```

### Options

#### ABSTIME(DATA-AREA)

specifies the data value for the time, in packed decimal, since 00:00 hours on 1 January 1900 (in milliseconds rounded to the nearest hundredth of a second) that is to be converted to an alternative format.

```
01 DATA-AREA PIC S9(15) COMP-3.
```

#### DATEFORM(DATA-AREA)

specifies the format of the installation-defined date. CICS returns YYMMDD, DDMMYY, or MMDDYY (six characters) according to the DATFORM system initialization parameter.

#### DATESEP(DATA-VALUE)

specifies the character to be inserted as the separator between the year and the month, and between the day and the month; or between the year and the day if form YYDDD is specified.

#### DAYCOUNT(DATA-AREA)

returns the number of days since 1 January 1900 (day 1), as a fullword binary number. This is useful if you need to compare the current date with a previous date that has, for example, been stored in a data set.

#### DAYOFMONTH(DATA-AREA)

returns the number of the day in the month as a fullword binary number.

#### DAYOFWEEK(DATA-AREA)

returns the relative day number of the week as a fullword binary number: Sunday=0, Saturday=6. This number can be converted to a textual form of day in any language.

#### DDMMYY(DATA-AREA)
specifies the 8-character user field where CICS is to return the date, in day/month/year format (for example, 21/10/95). A separator is present if requested by the DATESEP option.

**DDMMYYYY**(DATA-AREA)

specifies the 10-character user field where CICS is to return the date, in day/month/year format (for example 17/06/1995). A separator is present if requested by the DATESEP option.

**FULLDATE**(DATA-AREA)

specifies the 10-character user field where CICS is to return the date, in the format specified in the DATFORM system initialization parameter, with the year expanded to 4 digits. You should normally use this option only when a date is needed for output purposes. A separator is present if requested by the DATESEP option.

**MMDDYY**(DATA-AREA)

specifies the 8-character user field in which CICS is to return the date, in month/day/year format (for example, 10/21/95).

**MMDDYYYY**(DATA-AREA)

specifies the 10-character user field where CICS is to return the date, in month/day/year format (for example 11/21/1995). A separator is present if requested by the DATESEP option.

**MONTHOFYEAR**(DATA-AREA)

"data-area" is set to the relative month number of the year as a fullword binary number (January=1, December=12). You can convert this number, in your application program, to the name of the month in any language.

**TIME**(DATA-AREA)

"data-area" is set as an 8-character field to the current 24-hour clock time in the form hh:mm:ss, where the separator is specified by the TIMESEP option.

**TIMESEP**(DATA-VALUE)

specifies the character to be used as the separator in the returned time. If you omit this option, no separator is assumed and six bytes are returned in an 8-character field. If you omit the "data-value", a colon (:) is used as a separator.

**YEAR**(DATA-AREA)

specifies the full 4-figure number of the year as a fullword binary number (for example, 1995, 2001)

**YYDDD**(DATA-AREA)

specifies the 6-character user field where CICS is to return the date, in year/day format (for example, 95/301). A separator is present if requested by the DATESEP option.

**YYDDMM**(DATA-AREA)

specifies the 8-character user field where CICS is to return the date, in year/month format (for example, 95/30/10). A separator is present if requested by the DATESEP option.

**YYMMDD**(DATA-AREA)

specifies the 8-character user field where CICS is to return the date, in year/month/day format (for example, 95/10/21). A separator is present if requested by the DATESEP option.
YYYDDD(DATA-AREA)
specifies the 8-character user field where CICS is to return the date, in year/day format (for example 1995/200). A separator is present if requested by the DATESEP option.

YYYDDMM(DATA-AREA)
specifies the 10-character user field where CICS is to return the date, in year/day/month format (for example 1995/21/06). A separator is present if requested by the DATESEP option.

YYMMDD(DATA-AREA)
specifies the 10-character user field where CICS is to return the date, in year/month/day format (for example 1995/06/21). A separator is present if requested by the DATESEP option.
3.2.2.2 GETMAIN

This command is used to obtain a specified amount of main storage and, optionally, to initialize that storage to a specified bit configuration.

The pointer reference specified in the SET option is set to the address of the acquired storage.

```
GETMAIN SET(ptr) LENGTH(len) | FLENGTH(len) [ SHARED ] [ INITMIG(area) ] [ USERDATAKEY ] [ CICSDATAKEY ]
```

**Options**

**FLENGTH(LEN)**

specifies the number of bytes of storage required, in fullword binary format.

**INITMIG(AREA)**

specifies an optional 1-byte initialization value. If you specify INITMIG, CICS sets every byte of the acquired storage to the bit string you provide.

**LENGTH(LEN)**

specifies the number of bytes (unsigned halfword binary value) of storage required. LENGTH implies storage from below the 16MB line and has an upper limit of 65 520 bytes. If you want storage above the 16MB line or a larger area, use FLENGTH.

**SET(PTR)**

sets the pointer reference to the address of the acquired main storage. The pointer is set to the first byte of the storage area.

**SHARED**

prevents the automatic release of storage obtained by a GETMAIN command at the end of the task that requested it.

**USERDATAKEY**

Only for mainframe compatibility reasons: produces no effect.

**CICSDATAKEY**

Only for mainframe compatibility reasons: produces no effect.
3.2.2.3 HANDLE AID

The HANDLE AID command is used to pass control to a specified label when XCICS receives an AID from a display device; control is passed after the input operation is completed. In the absence of a HANDLE AID for an AID, control returns to the application program at the point immediately following the input request.

<table>
<thead>
<tr>
<th>HANDLE AID</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clear</td>
</tr>
<tr>
<td></td>
<td>Enter</td>
</tr>
<tr>
<td></td>
<td>Anykey</td>
</tr>
<tr>
<td></td>
<td>Operid</td>
</tr>
<tr>
<td></td>
<td>Pa1-Pa3</td>
</tr>
<tr>
<td></td>
<td>Pf1-Pf24</td>
</tr>
</tbody>
</table>

**Options**

Look in EIBAID to determine which key was pressed. The options that can be specified are:

- CLEAR
- ENTER
- ANYKEY
- OPERID
- PA1-PA3
- PF1-PF24
3.2.2.3 HANDLE ABEND

This command is used to activate, cancel or reactivate an exit for abnormal termination processing. The command can be suspended by means of the PUSH and POP commands as described below.

<table>
<thead>
<tr>
<th>HANDLE ABEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ PROGRAM(pgmname) ]</td>
</tr>
<tr>
<td>[ LABEL(label) ]</td>
</tr>
<tr>
<td>[ CANCEL ]</td>
</tr>
<tr>
<td>[ RESET ]</td>
</tr>
</tbody>
</table>

**Options**

CANCEL
 specifies that a previously established exit at the logical level of the application program in control is to be canceled.

LABEL(LABEL)
 specifies the program label to which control branches if abnormal termination occurs.

RESET
 specifies that an exit canceled by a HANDLE ABEND CANCEL command, or by CICS, is to be reactivated.

PROGRAM(PGMNAME)
 specifies the name of the program to which control is to be passed if the task is terminated abnormally. If this program has not already been defined, the program will be autoinstalled in the event of the abend condition being raised.
3.2.2.3 HANDLE CONDITION

You use HANDLE CONDITION to specify the label to which control is to be passed if a condition occurs. You must include the name of the condition and, optionally, a label to which control is to be passed if the condition occurs. You must ensure that the HANDLE CONDITION command is executed before the command that may give rise to the associated condition.

```
HANDLE CONDITION <condition> (<label>)
```

**Note**

Refer to "Appendix A : [Supported Conditions]" topic, to knows how conditions are supported.
3.2.2.3 IGNORE CONDITION

You use IGNORE CONDITION to specify that no action is to be taken if a condition occurs. The IGNORE CONDITION command for a given condition applies only to the program in which it is specified, and it remains active while the program is being executed, or until a HANDLE CONDITION command for the same condition is encountered, in which case the IGNORE CONDITION command is overridden.

```
IGNORE CONDITION <condition>
```

**Note**

Refer to "Appendix A : [Supported Conditions]" topic, to know how conditions are supported.
3.2.2.3 INQUIRE

Please refer to appendix B for a list of supported INQUIRE/SET parameters
3.2.2.3 ISSUE ABEND

Abend the mapped conversation with an APPC partner.

**ISSUE ABEND**

<table>
<thead>
<tr>
<th>ISSUE</th>
<th>ABEND</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ CONVID(name) ]</td>
<td></td>
</tr>
<tr>
<td>[ STATE(cvda) ]</td>
<td></td>
</tr>
</tbody>
</table>

**Options**

**CONVID**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal facility (returned by a previously executed ASSIGN command).

**STATE**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.3 ISSUE CONFIRMATION

ISSUE CONFIRMATION allows an application program to respond positively when the CONFIRM option has been specified on a SEND command executed by a partner transaction.

```
ISSUE CONFIRMATION
   [ CONVID(name) ]
   [ STATE(cvda) ]
```

**Options**

**CONVID**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal facility (returned by a previously executed ASSIGN command).

**STATE**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.3 ISSUE ERROR

Inform APPC mapped conversation partner of error.

<table>
<thead>
<tr>
<th>ISSUE</th>
<th>ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ CONVID(name) ]</td>
</tr>
<tr>
<td></td>
<td>[ STATE(cvda) ]</td>
</tr>
</tbody>
</table>

**Options**

**CONVID**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal facility (returned by a previously executed ASSIGN command).

**STATE**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.3 ISSUE PASS

ISSUE PASS instruct the terminal to disconnect and reconnect to another XCICS region, eventually providing a data area to be passed to the other region and to be extracted with EXTRACT LOGONMSG.

```
ISSUE   PASS
   [ LUNAME(luname) ]
   [ FROM(area) ]
   [ LENGTH(lenght) ]
```

**Options**

**LUNAME(LUNAME)**

identifies the name of the APPLID to connect to. This name must be 8 characters long and must be defined in the terminal sessions lists.

Terminals connected with X4J must have the definition of a session with the same name specified in LUNAME. For XTND connections, the xtnd.xml configuration file must contain the definition of the application specified in LUNAME.

**FROM(AREA)**

specifies the data area containing the logon user data that is to be passed to the application named in the LUNAME option.

**LENGTH(LENGTH)**

specifies the length, as a halfword binary value, of the data issued.
ISSUE PREPARE applies only to distributed transaction processing over APPC links. It enables a syncpoint initiator to prepare a syncpoint slave for syncpointing by sending only the first flow (prepare-to-commit) of the syncpoint exchange. Depending on the reply from the syncpoint slave, the initiator can proceed with the syncpoint by issuing a SYNCPPOINT command, or initiate back-out by issuing a SYNCPPOINT ROLLBACK command.

ISSUE PREPARE
[ CONVID(name) ]
[ STATE(cvda) ]

Options

CONVID(NAME)
identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal session (returned by a previously executed ASSIGN command).

STATE(CVDA)
gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFCOUNT
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.4 ISSUE PRINT

ISSUE PRINT prints displayed data on the first available printer that can respond to a print request.
### 3.2.2.4 ISSUE SIGNAL

ISSUE SIGNAL, in a transaction in receive mode, signals to the sending transaction that a mode change is needed. It raises the SIGNAL condition on the next SEND, RECEIVE, or CONVERSE command executed in the sending transaction, and a previously executed HANDLE CONDITION command for this condition can be used either to take some action, or to ignore the request.

```plaintext
ISSUE SIGNAL
[ CONVID(name) ]
[ STATE(cvda) ]
```

**Options**

**CONVID(NAME)**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal session (returned by a previously executed ASSIGN command).

**STATE(CVDA)**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNCFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.4 JOURNAL

This command creates a journal record.

This command is supported for compatibility with earlier releases of CICS. It is superseded by the WRITE JOURNALNAME command, which you are recommended to use instead.
3.2.2.4 LINK

This command is used to pass control from an application program at one logical level to an application program to the next lower logical level, with the capability to pass data to it.

In some circumstances, the linked-to program may reside on another XCICS or CICS region (see Distributed Program Link below).

```
LINK PROGRAM(pgmname)
    [ COMMAREA(area) ]
    [ LENGTH(data-area) ]
    [ DATALENGTH(data-area) ]
    [ SYSID(sysid) ]
    [ SYNCONRETURN ]
    [ TRANSID(data-area) ]
```

**Options**

**COMMAREA(AREA)**

specifies a communication area to be made available to the invoked program. In this option the contents of the data-area are passed.

**LENGTH(LEN)**

specifies the length (halfword binary data value) in bytes of the communication area.

**PROGRAM(PGMNAME)**

specifies the identifier (1-8 alphanumeric characters) of the program to link.

**DATALENGTH(DATA-VALUE)**

specifies a halfword binary value that is the length of a contiguous area of storage, from the start of the COMMAREA, to be passed to the invoked program. If the amount of data being passed in a COMMAREA is small, but the COMMAREA itself is large so that the linked-to program can return the requested data, you should specify DATALENGTH in the interest of performance.

**SYNCONRETURN**

specifies that the server region named on the SYSID option is to take a syncpoint on successful completion of the server program.

Changes to recoverable resources made by the server program are committed or rolled-back independently of changes to recoverable resources made by the client program issuing the LINK request, or changes made by the server in any subsequent LINK.

The NORMAL condition is returned if changes to recoverable resources are committed before return from the server program.

The ROLLEDBACK condition is returned if changes to recoverable resources are rolled back before return from the server program.

The TERMERR condition is raised following failure of the communications link or the system in which the server program is executing. The client program is responsible for handling the condition and ensuring that data consistency is restored.

Synconreturn is only applicable to remote LINKs, it is ignored if the LINK is local.

**SYSID(SYSTEMNAME)**
specifies the system name of an XCICS or CICS server region to where the program link request is to be routed.

If SYSID specifies the local system, CICS treats the LINK request as if SYSID was not specified.

A remote system name specified on the SYSID option takes priority over any remote system name specified on the PROGRAM resource definition.

TRANSID(NAME)

specifies the name of the mirror transaction that the remote region is to attach, and under which it is to run the server program. If you omit the TRANSID option, the server region attaches either CSMI, CPMI, or CVMI by default.

Distributed program link

In any of the following cases, the link is a distributed program link (DPL):

- You specify a remote region name on the SYSID option (with or without the associated TRANSID and SYNCONRETURN options).
- The SYSID option on the installed PROGRAM definition (PPT) specifies the name of a remote region.

In response to a distributed program link, the local XCICS region (the client region) ships the link request to the remote region (the server region). The server region executes the linked-to program (the server program) on behalf of the program issuing the link request (the client program).

The server program running in the server region is restricted to a DPL subset of the CICS API. Briefly, the server program cannot issue:

- Terminal control commands that reference the principal facility
- Options of ASSIGN that return terminal attributes
- BMS commands
- Signon and signoff commands
- Batch data interchange commands
- Commands that address the TCTUA.

The remote server region may be an XCICS/TS or an IBM CICS region.

Of course, when the server region is an IBM CICS region running on a mainframe, the commarea must be correctly translated to and from EBCDIC. The translation may be automated by defining a converter routine in the PPT definition of the program itself. Please refer to the XCICS Customization Guide for further information about dynamic conversion routines.
3.2.2.4 LOAD

This command is used to fetch application programs, tables or maps from the library where they reside and load them into main storage. This facility is used to load an application program that will be used repeatedly, thereby system overhead can be reduced through a single load or with the load of a table to which control is not to be passed.

```
LOAD      PROGRAM(pgmname)
          [ SET(area) ]
          [ ENTRY(ptr) ]
          [ LENGTH(len) ]
          [ FLENGTH(flen) ]
          [ HOLD ]
```

**Options**

**ENTRY(PTR)**

specifies the pointer reference that is to be set to the address of the entry point in the program that has been loaded.

**FLENGTH(FLEN)**

specifies a fullword binary area to be set to the length of the loaded program, table, or map.

**HOLD**

specifies that the loaded program, table, or map is not to be released (if still available) when the task issuing the LOAD command is terminated; it is to be released only in response to a RELEASE command from this task or from another task.

**FLENGTH(FLEN)**

specifies a halfword binary value to be set to the length of the loaded program, table, or map. To avoid raising the LENGERR condition, use FLENGTH if the length of the loaded program is likely to be greater than 32KB.

**PROGRAM(PGMNAME)**

specifies the identifier (1-8 characters) of a program, table, or map to be loaded.

**SET(PTR)**

specifies the pointer reference that is to be set to the address at which a program, table, or map is loaded.
3.2.2.4 POP HANDLE

POP HANDLE enables you to restore the effect of IGNORE CONDITION, HANDLE ABEND, HANDLE AID, and HANDLE CONDITION commands to the state they were in before a PUSH HANDLE command was executed at the current link level. This can be useful, for example, during a branch to a subroutine embedded in a main program.

**Note**

Normally, when a CICS program calls a subroutine (at the same logical level), the program or routine that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use PUSH HANDLE to suspend existing HANDLE commands, and before returning control to the caller, can then restore the original commands using the POP HANDLE command.
3.2.2.4 POST

You use this command to request notification that a specified time has expired.

```
POST INTERVAL(int)  
[ TIME(t) ]  
[ SET(a) ]  
[ REQID(reqid) ]
```

**Options**

**INTERVAL(HHMMSS)**

specifies an interval of time that is to elapse from the time at which the POST command is issued. The mm and ss are in the range 0-59. The time specified is added to the current clock time by CICS when the command is executed to calculate the expiration time.

**REQID(REQQID)**

specifies a name (1-8 characters), which should be unique, to identify the POST request. Using this option to specify an application-defined name is one way to enable another transaction to cancel the POST request.

**SET(PTR)**

specifies the pointer reference to be set to the address of the 4-byte timer-event control area generated by CICS.

**TIME(HHMMSS)**

specifies the time when the posting of the timer-event control area should occur.
3.2.2.4 PURGE MESSAGE

The PURGE MESSAGE command simply deletes the current logical message, including any pages of device dependent data stream already prepared. The syntax of the PURGE MESSAGE command is the following:

```
PURGE MESSAGE
```
3.2.2.4 PUSH HANDLE

PUSH HANDLE enables you to suspend the current effect of IGNORE CONDITION, HANDLE ABEND, HANDLE AID, and HANDLE CONDITION commands. This can be useful, for example, during a branch to a subroutine embedded in a main program.

```
PUSH HANDLE
```

**Note**

Normally, when a CICS program calls a subroutine at the same logical level, the program or routine that receives control inherits the current HANDLE commands. These commands may not be appropriate within the called program. The called program can use PUSH HANDLE to suspend existing HANDLE commands.
### 3.2.2.4 READ

You use this command to read a record from a direct access data set on a local or remote system. If you include the UPDATE option, you must identify the record to be updated by the record identification field specified in the RIDFLD option.

<table>
<thead>
<tr>
<th>READ</th>
<th>DATASET(dsname)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ INTO(area)</td>
</tr>
<tr>
<td></td>
<td>RIDFLD(ridfld)</td>
</tr>
<tr>
<td></td>
<td>[ LENGTH(arealen) ]</td>
</tr>
<tr>
<td></td>
<td>[ KEYLENGTH(klen) [ GENERIC ] ]</td>
</tr>
<tr>
<td></td>
<td>[ UPDATE ]</td>
</tr>
<tr>
<td></td>
<td>[ SYSID(sysid) ]</td>
</tr>
<tr>
<td></td>
<td>[ RBA</td>
</tr>
<tr>
<td></td>
<td>[ GTEQ</td>
</tr>
</tbody>
</table>

#### Options

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**EQUAL**

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**GTEQ**

specifies that, if the search for a record having the same key (complete or generic) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key is retrieved.

**INTO(AREA)**

specifies the data area into which the record retrieved from the data set is to be written.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**LENGTH**

specifies the length, as a halfword binary value, of the data area where the record is to be put. On completion of the READ command, the LENGTH parameter contains the actual length of the record.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record.
number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.

**SET(PTR)**

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.

**UPDATE**

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.
3.2.2.5 READNEXT

You use this command to read records in sequential order from a data set on a local or remote system.

You can also use it during skip sequential processing.

```
READNEXT DATASET(dsname)
  [ INTO(area) | SET(pointer) ]
  RIDFLD(ridfld)
  [ LENGTH(arealen) ]
  [ KEYLENGTH(klen) [ GENERIC ] ]
  [ UPDATE ]
  [ SYSID(sysid)]
  [ RBA | RRN ]
  [ REQID(reqid) ]
```

**Options**

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**INTO(AREA)**

specifies the data area into which the record retrieved from the data set is to be written.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**LENGTH**

specifies the length, as a halfword binary value, of the data area where the record is to be put. On completion of the READ command, the LENGTH parameter contains the actual length of the record.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
SET(PTR)

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.

UPDATE

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.
3.2.2.5 READPREV

You use this command to read records in reverse sequential order from a data set on a local or remote system.

```
READPREV DATASET(dsname)
   [ INTO(area) | SET(pointer) ]
   RIDFLD(ridfld)
   [ LENGTH(arealen) ]
   [ KEYLENGTH(klen) | GENERIC | ]
   [ UPDATE ]
   [ SYSSID(sysid) ]
   [ RBA | RRN ]
   [ REQID(reqid) ]
```

**Options**

**DATASET(DSNNAME)**

specifies the name of the file to be accessed.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**INTO(AREA)**

specifies the data area into which the record retrieved from the data set is to be written.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**LENGTH**

specifies the length, as a halfword binary value, of the data area where the record is to be put. On completion of the READ command, the LENGTH parameter contains the actual length of the record.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.

**SET(PTR)**

specifies the record identification field specified in the RIDFLD option contains a relative byte address.
indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.

**UPDATE**

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.
3.2.2.5 READQ TD

This command is used to read transient data from a predefined symbolic resource.

```
READQ TD  QUEUE(qname)
            SET(ptr) | INTO(area)
            [ LENGTH(len) ]
            [ SYSID(systemname) ]
```

**Options**

**INTO(AREA)**

specifies the user data area into which the data read from the transient data queue is to be placed.

**LENGTH(LEN)**

specifies the length, as a halfword binary value, of the record to be read.

**QUEUE(QNAME)**

specifies the symbolic name (1-4 alphanumeric characters) of the queue to be read from. The name must have been defined in the DCT.

**SET(PTR)**

specifies a pointer reference that is to be set to the address of the data read from the queue.

**SYSID(SYSTEMNAME)**

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.5 READQ TS

This command is used to retrieve data from a temporary storage queue in main or auxiliary storage.

```
READQ TS [ QUEUE(qname) | QNAME(qname) ]
          SET(ptr) | INTO(area)
          [ LENGTH(len) ]
          [ ITEM(itemno) ]
          [ NEXT ]
          [ SYSID(systemname) ]
```

**Options**

**INTO (AREA)**

specifies the data area into which the data is to be written. The data area can be any variable, array, or structure.

**ITEM (ITEMNO)**

provides a halfword binary value that specifies the item number of the logical record to be retrieved from the queue.

**LENGTH (LEN)**

specifies the length, as a halfword binary value, of the record to be read.

**NEXT**

specifies retrieval for the next sequential logical record following the last record retrieved (by any task), or the first record if no previous record has been retrieved.

**QUEUE(QNAME) | QNAME(QNAME)**

specifies the symbolic name (1-8 alphanumeric characters) of the queue to be deleted. The name must be unique within the XCICS system.

**SET (PTR)**

specifies the pointer reference that is set to the address of the retrieved data. The pointer reference, unless changed by other commands or statements, is valid until the next READQ TS command or the end of task. If you specify the SET option, the LENGTH must be specified.

**SYSID(SYSTEMNAME)**

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.5 RECEIVE

The RECEIVE command is used to read data from a terminal or logical unit. The INTO option is used to specify the area into which the data is to be placed. Alternatively, a pointer reference can be specified in the set option. XCICS acquires then an area large enough to hold the data and sets the pointer reference to the address of that data.

```
RECEIVE INTO(area) [ SET(ptr) ] [ LENGTH(len) ] [ ASIS ] [ MAXLENGTH(max) ] [ NOTRUNCATE ] [ SESSION(s) ] [ CONVID(convid) ] [ TRANSLATE | NOTRANSLETE ]
```

**Options**

**ASIS**

specifies that lowercase characters in the 3270 input data stream are not translated to uppercase; this allows the current task to receive a message containing both uppercase and lowercase data.

**CONVID(CONVID)**

identifies the conversation to which the command relates.

**INTO(AREA)**

specifies the data area into which the mapped data is to be written. If this field is not specified, the name defaults to the name of the map suffixed with an I.

**LENGTH(LEN)**

specifies the length of the data to be formatted as a halfword binary value. It must not exceed the length of the FROM data area, but this should include the length of the 12-byte prefix generated by the TIOAPFX=YES option.

**MAXLENGTH(MAX)**

specifies the maximum amount (halfword binary value) of data that CICS is to recover.

**NOTRUNCATE**

specifies that, when the data available exceeds the length requested, the remaining data is not to be discarded but is to be retained for retrieval by subsequent RECEIVE commands.

**SET(PTR)**

specifies the pointer that is to be set to the address of the 12-byte prefix to the mapped data.

**SESSION(SESSION)**

specifies the symbolic identifier (1-4 characters) of a session TCTTE. This option specifies the alternate facility to be used.

**TRANSLATE**
Only for APPC: data are automatically translated to EBCDIC whenever the remote system is defined as EBCDIC in CCT. Be careful; translation assumes the data to be converted are is not aware of binary fields.

**NOTTRANSLATE**

Only for terminal connection: data are automatically not translated to EBCDIC.
3.2.2.5 RECEIVE MAP

The RECEIVE MAP command maps data from a terminal into a data area in an application program.

```
RECEIVE MAP(mapname)
    [ MAPSET(mapsetname) ]
    [ SET(ptr) | INTO(area) ]
    [ LENGTH(len) ]
    [ FROM(area) ]
    [ TERMINAL ]
    [ ASIS ]
```

### Options

**ASIS**

specifies that lowercase characters in the 3270 input data stream are not translated to uppercase; this allows the current task to receive a message containing both uppercase and lowercase data.

**FROM(LEN)**

specifies the data area containing the data to be mapped by a RECEIVE MAP command. This includes the 12-byte prefix generated by the TIOAPFX=YES option on the DFHMDI and DFHMSD BMS map definitions.

**INTO(AREA)**

specifies the data area into which the mapped data is to be written. If this field is not specified, the name defaults to the name of the map suffixed with an I.

**LENGTH(LEN)**

specifies the length of the data to be formatted as a halfword binary value. It must not exceed the length of the FROM data area, but this should include the length of the 12-byte prefix generated by the TIOAPFX=YES option.

**MAP(MAPNAME)**

specifies the name (1-7 characters) of the map to be used.

**MAPSET(MAPSETNAME)**

specifies the unsuffixed name (1-7 characters) of the mapset to be used. The mapset must reside in the CICS program library.

**SET(PTR)**

specifies the pointer that is to be set to the address of the 12-byte prefix to the mapped data.

**TERMINAL**

specifies that input data is to be read from the terminal that originated the transaction.
3.2.2.5 RELEASE

This command is used to delete from main storage a program, table or map previously loaded by a load command.

```
RELEASE PROGRAM(pgmname)
```

**Options**

**PROGRAM(PGMNAME)**

specifies the identifier (1-8 characters) of a program, table, or map to be loaded.
3.2.2.5 RESETBR

RESETBR specifies the record in a file where you want the browse to be repositioned.

```
RESETBR  DATASET(dsname)
          RIDFLD(ridfld)
          [ KEYLENGTH(klen) [ GENERIC ] ]
          [ SYSID(sysid) ]
          [ RBA | RRN ]
          [ GTEQ | EQUAL ]
          [ REQID(reqid) ]
```

Options

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**EQUAL**

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**GTEQ**

specifies that, if the search for a record having the same key as that specified in the RIDFLD option is unsuccessful, the first record having a greater key is retrieved.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address

**REQID(REQID)**

specifies a unique (halfword binary value) request identifier for a browse, used to control multiple browse operations on a data set.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
3.2.2.5 RETRIEVE

This command is used to retrieve data stored by expired START commands. It is the only method available for accessing such data.

```plaintext
RETRIEVE    INTO(area) | SET(ptr)
            LENGTH(len)
            [ RTRANSID(rtransid) ]
            [ RTERMD(rtermid) ]
            [ QUEUE(qname) ]
```

**Options**

**INTO (AREA)**

specifies the user data area into which retrieved data is to be written.

**LENGTH (LEN)**

specifies a halfword binary value to define the length of the data area the retrieved data is written into.

**RTERMD (RTERMD)**

specifies a 4-character area that can be used in the TERMID option of a START command that may be executed subsequently.

**RTRANSID (RTRANSID)**

specifies a 4-character area that can be used in the TRANSID option of a START command that may be executed subsequently.

**QUEUE (QNAME)**

specifies the 8-character area for the temporary storage queue name that may be accessed by the transaction issuing the RETRIEVE command.

**SET (PTR)**

specifies the pointer reference to be set to the address of the retrieved data. If you specify the SET option, the argument must be a data area.
3.2.2.5 RETURN

This command is used to return control from an application program either to an application program at the next higher logical level or to XCICS.

```
RETURN [ TRANSID(transid) ]
[ COMMAREA(area) ]
[ LENGTH(len) ]
```

**Options**

**COMMAREA(AREA)**

specifies a communication area that is to be made available to the next program that receives control.

**LENGTH(LEN)**

specifies the length (halfword binary data value) in bytes of the communication area.

**TRANSID(TRANSID)**

specifies the transaction identifier (1-4 characters) to be used with the next input message entered from the terminal with which the task that issued the RETURN command has been associated. The specified name must have been defined as a transaction to CICS.
3.2.2.6 REWRITE

You use this command to update a record in a direct-access data set on a local or remote system. You must always precede this command with a READ UPDATE to read the record to be updated.

```
REWRITE     DATASET(dsname)
            FROM(area)
            [ LENGTH(arealen) ]
            [ SYSID(sysid) ]
```

**Options**

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**FROM(AREA)**

specifies the record that is to be written to the data set referred to by this file

**LENGTH**

specifies the length, as a halfword binary value, of the data area where the record is to be put. On completion of the READ command, the LENGTH parameter contains the actual length of the record

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
3.2.2.6SEND

The SEND command is used to write data to a terminal or logical unit.

```
SEND        FROM(area)
             LENGTH(len)
             [ WAIT ]
             [ ERASE ]
             [ CTLCHAR(c) ]
             [ TRANSLATE | NOTRANSLETE ]
```

**Options**

**CTLCHAR(C)**

specifies a 1-byte write control character that controls a SEND command for a 3270.

**ERASE**

specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of the screen.

**FROM(AREA)**

specifies the data area containing the data to be processed. If this field is not specified, the name defaults to the name of the map suffixed with an O.

**LENGTH(AREALEN)**

specifies the length of the data to be formatted as a halfword binary value.

**WAIT**

specifies that control should not be returned to the application program until the output operation has been completed.

**TRANSLATE**

Only for APPC: data are automatically translated to EBCDIC whenever the remote system is defined as EBCDIC in CCT. Be careful: translation assumes the data to be converted are is not aware of binary fields.

**NOTRANSLETE**

Only for terminal connection: data are automatically not translated to EBCDIC.
3.2.2.6 SEND CONTROL

Send device controls to a terminal without map or text data.

```
SEND   CONTROL
     [ CURSOR( pos ) ]
     [ ERASE ]
     [ FREEKB ]
     [ FRSET ]
     [ ALARM ]
     [ TERMINAL ]
     [ WAIT ]
```

**Options**

**ALARM**

specifies that the 3270 audible alarm feature is to be activated.

**CURSOR**

specifies the location the 3270 or 3604 cursor is returned to on completion of a SEND CONTROL command.

**ERASE**

specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of the screen.

**FREEKB**

specifies that the 3270 keyboard is to be unlocked. If FREEKB is omitted, the keyboard remains locked.

**FRSET**

specifies that the modified data tags (MDTs) of all fields currently in the 3270 (or partition) buffer are to be reset to the unmodified condition (that is, field reset).

**TERMINAL**

specifies that the output data is to be sent to the terminal that originated the transaction.

**WAIT**

specifies that control should not be returned to the application program until the output operation has been completed.
3.2.2.6 SEND MAP

The SEND MAP command is used to send mapped data to a terminal. Three kinds of data can be sent, depending on the specified options:

- constant display data such as headings, footings, prompt fields and comments
- variable display data such as user data or warning messages
- device control data such as instructions to clear the screen, or sound an alarm, before displaying data.

The SEND MAP command identifies the map BMS must use to format data, and the map set to which it belongs.

```
SEND MAP(mapname)
    [ MAPSET(mapsetname) ]
    [ FROM(area) [ LENGTH(arealen) ]
    [ SET(ptr) ]
    [ REQUID(requid) ]
    [ DATAONLY ]
    [ MAPONLY ]
    [ CURSOR(pos) ]
    [ ERASE ]
    [ ERASEAUP ]
    [ FREEKB ]
    [ FORMFEED ]
    [ ALARM ]
    [ FRSET ]
    [ ACCUM ]
    [ TERMINAL ]
    [ WAIT ]
    [ NLEOM ]
    [ NOFLUSH ]
    [ PAGING ]
```

**Options**

**ACCUM**

specifies that this command is one of a number of commands that are used to build a logical message. The logical message is completed by a SEND PAGE command, or deleted by a PURGE MESSAGE command.

**ALARM**

specifies that the 3270 audible alarm feature is to be activated.

**CURSOR(POS)**

specifies the location to which the 3270 or 3604 cursor is to be returned upon completion of a SEND MAP command. The data value must be a halfword binary value that specifies the cursor position relative to zero; the range of values that can be specified depends on the size of the screen being used. If no data value is specified, symbolic cursor positioning is assumed.

**DATAONLY**

specifies that only application program data is to be written.

**ERASE**

specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of
the screen.

**ERASEUP**

specifies that before this page of output is displayed, all unprotected character locations in the partition or the entire screen are to be erased.

**FORMFEED**

specifies that a new page is required. For 3270 printers and displays, the FORMFEED character is positioned at the start of the buffer. The application program must thus ensure that this buffer position is not overwritten by map or text data.

**FREEKB**

specifies that the 3270 keyboard should be unlocked after the data is written. If FREEKB is omitted, the keyboard remains locked.

**FROM(AREA)**

specifies the data area containing the data to be processed. If this field is not specified, the name defaults to the name of the map suffixed with an O.

**FRSET**

specifies that the modified data tags (MDTs) of all fields currently in the 3270 (or partition) buffer are to be reset to the unmodified condition (that is, field reset) before any map data is written to the buffer.

**LENGTH(AREALEN)**

specifies the length of the data to be formatted as a halfword binary value.

**MAP(MAPNAME)**

specifies the name (1-7 characters) of the map to be used.

**MAPONLY**

specifies that only default data from the map is to be written.

**MAPSET(MAPSETNAME)**

specifies the unsuffixed name (1-7 characters) of the mapset to be used. The mapset must reside in the CICS program library.

**NLEOM**

specifies that data for a 3270 printer or a 3275 display with the printer adapter feature should be built with blanks and new-line (NL) characters, and that an end-of-message (EM) character should be placed at the end of the data.

**NOFLUSH**

specifies that CICS does not clear pages on completion but returns control to the program (having set the OVERFLOW condition in EIBRESP).

**PAGING**

specifies that the output data is not to be sent immediately to the terminal, but is to be placed in temporary storage and displayed in response to paging commands entered by the terminal operator.
REQUID(REQUID)

specifies a 2-character prefix to be used as part of a temporary storage identifier for CICS message recovery.

SET(PTR)

specifies the pointer to be set to the address of the input or output data.

The SET option specifies that completed pages are to be returned to the application program. The pointer is set to the address of a list of completed pages. See the description of the SET option in the section on full BMS in the "IBM CICS Application Programming Guide" for more guidance about using the SET option.

TERMINAL

specifies that the output data is to be sent to the terminal that originated the transaction.

WAIT

specifies that control should not be returned to the application program until the output operation has been completed.
3.2.2.6 SEND PAGE

The SEND PAGE command causes BMS to generate a device dependent data stream for the last (perhaps the only) page of data.

```
SEND PAGE
    [ RELEASE ]
    [ RETAIN ]
    [ SET(ptr)]
    [ AUTOPAGE | NOAUTOPAGE ]
```

**Options**

**AUTOPAGE**

specifies that each page of a BMS logical message is to be sent to the terminal as soon as it is available.

**NOAUTOPAGE**

specifies that pages of a BMS logical message are to be sent one at a time to the terminal. BMS sends the first page to the terminal when the terminal becomes available or on request of the terminal operator.

**RELEASE**

specifies that, after the SEND PAGE command, control is to be returned to CICS.

**RETAIN**

specifies that after the SEND PAGE command, control is returned to the application program when the operator has finished displaying the pages.

**SET(PTR)**

specifies the pointer to be set to the address of the input or output data.

The SET option specifies that completed pages are to be returned to the application program. The pointer is set to the address of a list of completed pages. See the description of the SET option in the section on full BMS in the "IBM CICS Application Programming Guide" for more guidance about using the SET option.
3.2.2.6 SEND TEXT

The SEND TEXT command is used to send text to a terminal. The syntax of the SEND TEXT command is the following:

```
SEND    TEXT
FROM(area)
[ LENGTH(len) ]
[ CURSOR(pos) ]
[ REQID(reqid) ]
[ SET(ptr) ]
[ ERASE ]
[ FREEKB ]
[ ALARM ]
[ ACCUM ]
[ L40 | L64 | L80 ]
[ HEADER ]
[ TRAILER ]
[ WAIT ]
[ JUSTFIRST ]
[ JUSTLAT ]
[ JUSTIFY(line) ]
[ NLEOM ]
[ PAGING ]
[ HONEOM ]
```

**Options**

**ACCUM**

specifies that this command is one of a number of commands that are used to build a logical message. The logical message is completed by a SEND PAGE command, or deleted by a PURGE MESSAGE command.

**ALARM**

specifies that the 3270 audible alarm feature is to be activated.

**CUROSOR(POS)**

specifies the location to which the 3270 or 3604 cursor is to be returned on completion of a SEND TEXT command.

**ERASE**

specifies that the screen printer buffer or partition is to be erased and the cursor returned to the upper left corner of the screen.

**FORMFEED**

specifies that a new page is required.

**FREEKB**

specifies that the 3270 keyboard should be unlocked after the data is written. If FREEKB is omitted, the keyboard remains locked.

**FROM(AREA)**
specifies the data area containing the data to be sent.

**HONEOM**

specifies that the default printer line length is to be used.

**LENGTH(LEN)**

specifies the length of the data to be sent as a halfword binary value.

**NLEOM**

specifies that data for a 3270 printer or a 3275 display with the printer adapter feature should be built with blanks and new-line (NL) characters, and that an end-of-message (EM) character should be placed at the end of the data.

**PAGING**

specifies that the output data is not to be sent immediately to the terminal, but is to be placed in temporary storage and displayed in response to paging commands entered by the terminal operator.

**REQID(REQUID)**

specifies a 2-character prefix to be used as part of a temporary storage identifier for CICS message recovery.

**SET(PTR)**

specifies the pointer to be set to the address of the data. It specifies that completed pages are to be returned to the application program. The pointer is set to the address of a list of completed pages.

**WAIT**

specifies that control should not be returned to the application program until the output operation has been completed.

**accepted but not supported**

**HEADER**

specifies the header data to be placed at the beginning of each page of text data.

**JUSTFIRST**

specifies that the text data is to be placed at the top of the page.

**JUSTLAST**

specifies that the text data is to be positioned at the bottom of the page.

**JUSTIFY(LINE)**

specifies the line of the page at which the text data is to be positioned. The data value must be a halfword binary value in the range 1 through 240.

L40 | L64 | L80

specifies the line length for a 3270 printer; a carrier return and line feed are forced after 40, 64, or 80 characters have been printed on a line.
TRAILER

specifies the text data area that contains trailer data to be placed at the bottom of each output page.
3.2.2.6 SEND TEXT MAPPED

The SEND TEXT MAPPED command is used to send mapped data to a terminal, as returned by SEND MAP/PAGE SET commands.

The syntax of the SEND TEXT MAPPED command is the following:

```
SEND    TEXT MAPPED
FROM(area)
[ LENGTH(len) ]
[ REQID(requd) ]
[ PAGING ]
```

**Options**

**FROM(AREA)**

specifies the data area containing the data to be sent.

**LENGTH(LEN)**

specifies the length of the data to be sent as a halfword binary value.

**PAGING**

specifies that the output data is not to be sent immediately to the terminal, but is to be placed in temporary storage and displayed in response to paging commands entered by the terminal operator (not supported)

**REQID(REQID)**

specifies a 2-character prefix to be used as part of a temporary storage identifier for CICS message recovery.
3.2.2.6 SET

Please refer to appendix B for a list of supported INQUIRE/SET parameters.
3.2.2.6 SIGNOFF

Sign off from a terminal.

SIGNOFF
3.2.2.6 SIGNON

Sign on to a terminal.

SIGNON
  USERID(userid)
  [ PASSWORD(password) ]
  [ NEWPASSWORD(password) ]

Options

NEWPASSWORD(PASSWORD)

  specifies an optional 8-byte field defining a new password. This option is only valid if PASSWORD is also specified.

USERID(USERID)

  specifies the 8-byte sign-on USERID.

PASSWORD(PASSWORD)

  specifies an 8-byte password.
3.2.2.7 SPOOLCLOSE

The SPOOLCLOSE command closes a CICS spool report.

```
SPOOLCLOSE  TOKEN(token)
             [  DELETE ]
             [  KEEP  ]
```

**Options**

DELETE

For an OUTPUT report, DELETE specifies that the report is to be purged.

KEEP

For an OUTPUT report, KEEP specifies that the report is to be sent to its destination node.

TOKEN(TOKEN)

specifies the 8-character CICS-allocated token used to identify a report. It is a receiver on SPOOLOPEN and a sender on all other commands.
### 3.2.2.7 SPOOLOPEN OUTPUT

The SPOOLOPEN OUTPUT command opens a spool report for output from CICS to the system spooler and defines its characteristics.

```
SPOOLOPEN OUTPUT
  TOKEN(token)
  USERID(userid)
  NODE(node)
  [ CLASS(class) ]
  [ OUTDESCR(ptr) ]
  [ ASA | MCC | NOCC ]
  [ RECORDLENGTH(len) ]
```

#### Options

**ASA**

Specifies that the report has each record prefixed with an ASA carriage-control character, and this character must be used by the operating system to control formatting when the report is printed.

**CLASS(class)**

Specifies a 1-character class designation. If it is omitted, class A is assumed.

**NODE(node)**

Specifies the 8-character identifier of a destination node that the system spooler uses to route the file. It is a sender field.

**OUTDESCR(ptr)**

Specifies a pointer variable to be set to the address of a field that contains the address of a string of parameters to the OUTPUT statement of JCL.

**TOKEN(token)**

Specifies the 8-character CICS-allocated token used to identify a report.

**USERID(userid)**

Specifies the 8-character identifier of the destination userid that processes the report.

**accepted but not supported**

**MCC**

Specifies that the report has each record prefixed with an IBM machine command code carriage-control character, and this character must be used by the operating system to control formatting when the report is printed.

**NOCC**

Specifies that the report has no internal formatting controls.
3.2.2.7 SPOOLWRITE

The SPOOLOPEN OUTPUT command opens a spool report for output from CICS to the system spooler and defines its characteristics.

```
SPOOLWRITE  TOKEN(token)
           FROM(area)
           FLENGTH(len)
[   LINE   ]
```

**Options**

**FLENGTH(LEN)**

specifies the fullword binary variable that is to be set to the length of the data that is transferred. This is set by the user on output. It is optional and, if it is omitted, CICS uses the length of the data area.

**FROM(AREA)**

specifies the data area from which to take the variable length data.

**LINE**

specifies the format of the data to be sent. The default action is LINE.

**TOKEN(TOKEN)**

specifies the 8-character CICS-allocated token used to identify a report. It is a receiver on SPOOLOPEN and a sender on all other commands.
3.2.2.7 START

The START command is used to start a background task on a local or remote system, at a specified time. The starting task may pass data to the started task and may also specify a terminal to be used by the started task as its principal facility.

```
START   TRANSID(tac)
[ FROM(area) ]
[ LENGTH(len) ]
[ TERMID(term) ]
[ INTERVAL(hhmss) | TIME(hhmss) | AFTER SECONDS (sss)]
[ REQID(r) ]
[ RTRANSID(rt) ]
[ RTERMID(rterm) ]
[ QUEUE(qname) ]
[ PROTECT ]
[ NOCHECK ]
[ SYSID(systemname) ]
```

**Options**

**FROM (AREA)**

specifies the data to be stored for a task that is to be started at some future time.

**INTERVAL (HHMMSS)**

specifies the expiration time as an interval of time that is to elapse from the time at which the START command is issued. The mm and ss are each in the range 0-59. The time specified is added to the current clock time by XCICS when the command is executed, to calculate the expiration time.

**AFTER SECONDS (SSS)**

specifies a fullword binary value in the range 0-359 999 as an interval of time that is to elapse from the time at which the START command is issued. The time specified is added to the current clock time by XCICS when the command is executed, to calculate the expiration time.

**LENGTH (LEN)**

specifies a halfword binary data value that is the length of the data to be stored for the new task.

**NOCHECK**

specifies that, for a remote system, CICS should improve performance of the START command by providing less error checking and slightly less function

**PROTECT**

specifies that the new task is not started until the starting task has taken a syncpoint. If the starting task abends before the syncpoint is taken, the request to start the new task is canceled.

**QUEUE (QNAME)**

specifies a name (1-8 characters) that is passed to the started task. If this name represents a temporary storage queue, the queue must be local to the started task.
REQUID (R)

specifies a name (1-8 characters), which must be unique, to identify a command. This option can be used when another task is to be provided with the capability of canceling an unexpired command.

RTERMID (RTERMID)

specifies a value (1-4 characters), for example a terminal name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started.

RTRANSID (RT)

specifies a value (1-4 characters), for example a transaction name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started.

TERMID (TERM)

specifies the symbolic identifier (1-4 alphanumeric characters) of the principal facility associated with a transaction to be started as a result of a START command.

TIME (HHMMSS)

specifies the time when a new task should be started.

TRANSID(TAC)

specifies the symbolic identifier (1-4 characters) of the transaction to be executed by a task started as the result of a START command.

SYSID(SystemName)

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.7 STARTBR

You use this command to specify the record in a data set, on a local or remote system, at which you want the browse to start. No record will be read until a READNEXT or a READPREV command is executed.

```
STARTBR  DATASET(dsname)
         RIDFLD(ridfld)
         [ KEYLENGTH(klen) [ GENERIC ] ]
         [ UPDATE ]
         [ SYSID(sysid) ]
         [ RBA | RRN ]
         [ GTEQ | EQUAL ]
         [ REQID(reqid) ]
```

**Options**

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**EQUAL**

specifies that the search is satisfied only by a record having the same key (complete or generic) as that specified in the RIDFLD option.

**GENERIC**

specifies that the search key is a generic key with a length specified in the KEYLENGTH option.

**GTEQ**

specifies that, if the search for a record having the same key (complete or generic) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key is retrieved.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address.

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.

**SET(PTR)**
indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.

**UPDATE**

indicates that CICS is to supply a buffer where the record is read, and specifies the pointer reference that is to contain the address of the retrieved record.
3.2.2.7 SUSPEND

This command is used to relinquish control to a task of higher dispatching priority. Control is returned to the task issuing the command as soon as no other task of a higher priority is ready to be processed.

SUSPEND
3.2.2.7 SYNCPNT

To facilitate recovery in the event of abnormal termination of an XCICS task or of a system failure, the system programmer can define certain resources as recoverable during XCICS table generation. If a task is terminated abnormally, these resources are restored to the condition in which they were at the start of the task, which can then be rerun. This command is used to divide a task (usually running alone) into smaller logical units of work (LUWs). Each SYNCPNT command causes a sync point to be established to mark the completion of an LUW.
3.2.2.7 SYNCPOINT ROLLBACK

Back out to last syncpoint.

```
SYNCPOINT  ROLLBACK
```
3.2.2.7 UNLOCK

You use this command to release exclusive control position made in response to a READ command with the UPDATE option. You use it if you retrieve a record for update, and then decide that you do not want to update the record. The record can be in a data set on local or remote system.

```
UNLOCK    DATASET(dsname)
[  SYSID(sysid)  ]
```

**Options**

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
3.2.2.7 WAIT CONVID

WAIT CONVID allows an application program to ensure that any accumulated application data and control indicators from a SEND command, or the results of a CONNECT PROCESS command, are transmitted to the partner transaction.

```
WAIT [ CONVID(name) ]
[ STATE(cvda) ]
```

**Options**

**CONVID(NAME)**

identifies the conversation to which the command relates. The 4-character name identifies either the token returned by a previously executed ALLOCATE command in EIBRSRCE in the EIB, or the token representing the principal session (returned by a previously executed ASSIGN command).

**STATE(cvda)**

gets the state of the current conversation. The cvda values returned by CICS are:

- ALLOCATED
- CONFFREE
- CONFRECEIVE
- CONFSEND
- FREE
- PENDFREE
- PENDRECEIVE
- RECEIVE
- ROLLBACK
- SEND
- SYNFREE
- SYNCRECEIVE
- SYNCSEND
3.2.2.8 WAIT EVENT

You use this command to synchronize a task with the completion of an event initiated by the same task or by another task.

```
WAIT EVENT ECADDR(ptr)
```

**Options**

**ECADDR(PTR)**

specifies the address of the timer-event control area that must be posted before task activity can be resumed.
3.2.2.8 WAIT TERMINAL

This command is used to ensure that a terminal operation has completed before further process occur in a task under which more than one terminal or logical unit operation is performed.

**Note**

In XCICS the WAIT TERMINAL command is supported only for compatibility and it produces no effect.

```
WAIT TERMINAL
   [ SESSION(session) ]
   [ CONVID(convid) ]
```

**Options**

- **CONVID(CONVID)**
  - identifies the conversation to which the command relates.

- **SESSION(SESSION)**
  - specifies the symbolic identifier (1-4 characters) of a session TCTTE. This option specifies the alternate facility to be used.
3.2.2.8 WEB CLOSE

WEB CLOSE enables an application program to close a connection with a server. The session token identifies the connection that is to be closed. When the connection is closed, the session token that applies to it is no longer valid for use. The session token is required to receive a response from the server and to read the HTTP headers for the response, so the WEB CLOSE command should not be issued until all interaction with the server and with the response that it sends is complete. The command releases XCICS resources involved with the connection.

The WEB CLOSE command does not cause XCICS to notify the server that the connection should be terminated. It only makes XCICS close the connection on the client side. On the final request that you make using the connection, you should specify the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command. When this option is specified, XCICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. The information in the headers means that the server can close its connection with you immediately after sending the final response, rather than waiting to see if you send further requests before timing out.

The connection might also be closed at the request of the server before the WEB CLOSE command is issued. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server.

If the server does request termination of the connection, the data relating to that connection is still kept available within XCICS until the WEB CLOSE command is issued. The available data includes the most recent message received from the server, and the parameters used to open the connection (such as the scheme and the host name of the server). When a server has terminated the connection, the application program cannot:

- Send further requests on that connection, using the WEB SEND or WEB CONVERSE commands.
- Write HTTP headers, using the WEB WRITE HTTPHEADER command.

However, the application program can still:

- Receive a response, using the WEB RECEIVE command.
- Examine HTTP headers, using the WEB READ HTTPHEADER and HTTP header browsing commands.
- Extract connection information, using the WEB EXTRACT command.

When the WEB CLOSE command is issued, the data relating to the connection is cleared.

If the WEB CLOSE command is not issued by the application program, then at end of task XCICS clears the data relating to the connection and closes the connection, if it has not already been closed.

```
WEB CLOSE SESSTOKEN(data-value)
```

**Options**

**SESSTOKEN(DATA-VALUE)**

specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. When you issue the WEB CLOSE command for the connection identified by the session token, XCICS ends that connection and clears the data associated with it, and makes the session token invalid for further use by the application program. "Session tokens" in the XCICS Internet Guide explains the use of the session token.

**Conditions**

**NOTOPEN**

RESP2 values are:

27

Invalid session token.
WEB CONVERSE enables an application program to compose and send an HTTP client request, and receive a response from the server. A session token must be included on this command. For guidance on the correct use of the WEB CONVERSE command, see the CICS Internet Guide.

- The HTTP client request is made using a connection that has been opened using the WEB OPEN command. The WEB CONVERSE command can be used in place of the WEB SEND command to compose and send the request.

- The response from the server is received by XCICS Web support and passed to the application. The WEB CONVERSE command can be used in place of the WEB RECEIVE command to make the application program wait for and receive the HTTP response. The headers for the HTTP response can be examined separately using the WEB READ HTTPHEADER command or the HTTP header browsing commands.

**Note**

The RTIMOUT value specified for the transaction that starts the user application indicates the time that the application is prepared to wait to receive the incoming message. (RTIMOUT is specified on the transaction profile definition). When the period specified by RTIMOUT expires, XCICS returns a TIMEDOUT response to the application. An RTIMOUT value of zero means that the application is prepared to wait indefinitely. The default setting for RTIMOUT on transaction profile definitions is zero, so it is important to check and change that setting for applications that are making HTTP client requests.

The WEB CONVERSE command does not support chunked transfer-coding for the request, because this requires a sequence of send actions, and the WEB CONVERSE command provides a single send action. If you want to send a chunked message, use the WEB SEND command to send it, and the WEB RECEIVE command to receive it. If the server sends a chunked response, this can be received using the WEB CONVERSE command.

The WEB CONVERSE command cannot be used after the connection to the server has been closed. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server.

The WEB CONVERSE command performs a single send action and a single receive action, and it is designed to be used in place of a WEB SEND command and a WEB RECEIVE command. You may use WEB SEND and WEB RECEIVE commands, and WEB CONVERSE commands, in relation to the same connection (that is, with the same SESSTOKEN). However, if you are pipelining requests (that is, sending a sequence of requests without waiting for a response), you must not follow a WEB SEND command with a WEB CONVERSE command. XCICS checks at program run time that each WEB SEND command has a subsequent WEB RECEIVE command before any WEB CONVERSE command is issued.

For example, if you use the WEB SEND command three times to issue a pipelined sequence of requests, you must use the WEB RECEIVE command three times to receive the responses for those requests, before you may use the WEB CONVERSE command.

```plaintext
WEB CONVERSE
SESSTOKEN(data-value)
[[PATH(data-area) | URIMAP(data-value)]
PATHLENGTH(data-value)]
METHOD(cvda)
[ MEDIATYPE(data-area)]
[ QUERYSTRING(data-area) QUERYSTRLEN(data-value)]
[[DOCTYPE(data-value) | FROM(data-area)]
FROMLENGTH(data-value)]
[ ACTION(cvda)]
[ CLOSESTATUS(cvda)]
INTO(data-area) | SET(ptr-ref)
TOLENGTH(data-area)
[ MAXLENGTH(data-value)]
[ NOTRUNCATE]
[ STATUSCODE(data-area)]
[ STATUSTEXT(data-area) STATUSLEN(data-value)]
[ CHARACTERSET(data-value)]
[ CLIENTCONV(cvda)]
```
Options for sending the HTTP client request

ACTION(CVDA)

This option is used to specify how the message should be sent out. The CVDA value that applies for XCICS as an HTTP client is:

EXPECT

makes XCICS send an Expect header along with the request line and headers for the request, and await a 100-Continue response before sending the message body to the server. If a response other than 100-Continue is received, XCICS informs the application program and cancels the send. If no response is received after a period of waiting, XCICS sends the message body anyway.

This option must only be used if your request has a message body.

CLOSESTATUS(CVDA)

specifies whether or not a Connection header with the "close" connection option (Connection: close) should be included on the request. The default is that the header is not included. The CVDA values are:

CLOSE

makes XCICS write a Connection: close header for this request. The header notifies the server that the connection should be closed after the server has sent its response to the request. (For a server at HTTP/1.0 level, XCICS achieves the same effect by omitting the Connection: Keep-Alive header.)

NOCLOSE

means that the Connection: close header is not used for this request. If the server is identified as HTTP/1.0, XCICS sends a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), to notify that a persistent connection is desired.

DOCTOKEN(DATA-VALUE)

specifies the 16-byte binary token of a document to be sent as the message body. The document must be created using the XCICS Document interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands), as described in the XCICS Application Programming Guide. The FROM option provides an alternative way to create a message body. XCICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

FROM(DATA-AREA)

specifies a buffer of data which holds the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body.

There is no set maximum limit for the size of the data-area, but its size is limited in practice by storage considerations. The XCICS Internet Guide has more information about these.

FROMLENGTH(DATA-AREA)

specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option (the message body). It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

MEDIATYPE(DATA-AREA)

specifies the data content of any request body, for example text/xml. The media type is up to 56 alphanumeric characters, including appropriate punctuation. See the XCICS Internet Guide for more information about media types. XCICS checks that the format of the media type is correct, but does not check the validity of the media type against the data content.
For requests which require a body, you must specify the MEDIATYPE option, and there is no default.

For code page conversion to take place, the MEDIATYPE option must specify a type of data content that can be identified as text according to the IANA definitions. For non-text media types, XCICS does not convert the message body. The MEDIATYPE option is used for both the sending and receiving functions of the WEB CONVERSE command. If it is specified with a value, the value is used to construct the Content-Type header in the request, and the same field is used to receive the media type of the response that is returned by the server. If it is specified without a value, it is used only to receive the media type of the response.

**METHOD(CVDA)**

specifies the HTTP method for the request. The GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE methods are supported by this command. However, some HTTP servers, particularly HTTP/1.0 servers, might not implement all of these methods.

The XCICS Internet Guide has more information about the correct use of methods, including the HTTP versions that apply to each.

XCICS bars the sending of a message body for methods where it is inappropriate, and requires it for methods where it is appropriate. The CVDA values are:

**GET**

Obtain a resource from the server. A request body is not allowed.

**HEAD**

Obtain the HTTP headers, but not the response body, for a resource. A request body is not allowed.

**POST**

Send data to a server. A request body is required.

**PUT**

Create or modify a resource on the server. A request body is required.

**TRACE**

Trace the route of your request to the server. A request body is not allowed.

**OPTIONS**

Obtain information about the server. A request body is allowed, but there is no defined purpose for the body. If you do use a request body, then you must specify a media type.

**DELETE**

Delete a resource on the server. A request body is not allowed.

**PATH(DATA-AREA)**

specifies the path information for the specific resource within the server that the application needs to access.

If the URIMAP option was used to specify an existing URIMAP definition on the WEB OPEN command for this connection, the path specified in that URIMAP definition is the default path for the WEB SEND command. In these circumstances, if you do not specify path information on the WEB SEND command, the path from the URIMAP definition is used. If you specify a different path from that given in the URIMAP definition, this overrides the path from the URIMAP definition.

If the URIMAP option was not used on the WEB OPEN command, there is no default path, and you must provide path information. Path information can be extracted from a known URL using the WEB PARSE URL command. As an alternative to using the PATH option to provide the path information, you can use the URIMAP option on the WEB CONVERSE command to specify a URIMAP definition from which the path information is taken directly.

**PATHLENGTH(DATA-VALUE)**

specifies the length of the path, as a fullword binary value. If you are providing path information using the PATH
option, you need to specify the PATHLENGTH option. Path length information is returned if you use the WEB PARSE URL command to parse a URL.

**QUERYSTRING**(DATA-AREA)

specifies a query string that is to be supplied to the server as part of the request. You do not need to include a question mark (?) at the beginning of the query string; if you do not include it, XCICS supplies it for you automatically when constructing the request. If you include escaped characters in the query string, XCICS passes them to the server in their escaped format.

**QUERYSTRLEN**(DATA-VALUE)

specifies the length of the query string supplied on the QUERYSTRING option, as a fullword binary value.

**SESSTOKEN**(DATA-VALUE)

specifies the session token, an 8-byte binary value that uniquely identifies this connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the XCICS Internet Guide explains the use of the session token.

**URIMAP**(DATA-VALUE)

specifies the name (up to 8 characters, in mixed case) of a URIMAP definition that provides the path information for the specific resource within the server that the application needs to access. The URIMAP definition must be for XCICS as an HTTP client (with USAGE(CLIENT) specified). Its HOST attribute must be the same as the HOST attribute of the URIMAP definition that was specified on the WEB OPEN command for this connection, or the same as the host name specified in the HOST option on the WEB OPEN command for this connection. A URIMAP definition specified on the WEB CONVERSE command applies only to this request.

If the URIMAP option is specified, do not specify the PATH or PATHLENGTH options.

---

**Options for receiving the server's response**

**INTO**(DATA-AREA)

specifies the buffer that is to contain the data being received.

**MAXLENGTH**(DATA-VALUE)

specifies the maximum amount, as a fullword binary value, of data that XCICS is to pass to the application. The MAXLENGTH option applies whether the INTO or the SET option is specified for receiving data. If the data has been sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each individual chunk. The data is measured after any code page conversion has taken place.

If the length of data exceeds the value specified and the NOTRUNCATE option is not specified, the data is truncated to that value, and the remainder of the data is discarded.

If the length of data exceeds the value specified and the NOTRUNCATE option is specified, XCICS retains the remaining data and can use it to satisfy subsequent RECEIVE commands.

**MEDIATYPE**(DATA-AREA)

specifies a 56-character data-area to receive the media type (that is, the type of data content) for the body, for example text/xml. See the XCICS Internet Guide for more information about media types. The MEDIATYPE option is used for both the sending and receiving functions of the WEB CONVERSE command. If it is specified with a value, the value is used to construct the Content-Type header in the request, and the same field is used to receive the media type of the response that is returned by the server. If it is specified without a value, it is used only to receive the media type of the response.

**NOTRUNCATE**
specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no further RECEIVE commands are issued, the data is discarded during transaction termination.) A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining data, whatever its length. Alternatively, you can use a series of RECEIVE commands with the NOTRUNCATE option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no longer getting a LENGERR response. Bear in mind that if you receive less than the length requested on the MAXLENGTH option, this does not necessarily indicate the end of the data; this could happen if XCICS needs to avoid returning a partial character at the end of the data.

**SET(PTR-REF)**

specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the next receive command or the end of task.

**STATUSCODE(DATA-AREA)**

specifies a data-area to receive the HTTP status code sent by the server. The code is a binary halfword value. Examples are 200 (normal) or 404 (not found). Receiving the status code is optional, but you should always receive and check the status code in the following circumstances:

- If you intend to make an identical request to the server, now or during a future connection.
- If you intend to make further requests to the server using this connection.
- If your application is carrying out any further processing that depends on the information you receive in the response.

The XCICS Internet Guide has basic guidance on appropriate actions for an application to take in response to the status codes for HTTP/1.1.

**STATUSTEXT(DATA-AREA)**

specifies a data-area to receive any text returned by the server to describe the status code. The text is known as a reason phrase. Examples are "OK" (accompanying a 200 status code), or "Bad Request" (accompanying a 400 status code). The STATUSLEN option gives the length allowed for the text.

**STATUSLEN(DATA-VALUE)**

specifies, as a fullword binary value, the length of the data-area to receive any text returned by the server to describe the status code (the STATUSTEXT option). The text is known as a reason phrase. Most reason phrases recommended for HTTP are short, but a data-area length of 256 characters is suggested here, in case the server replaces the recommended reason phrase with more detailed information.

**TOLENGTH(DATA-AREA)**

specifies a fullword binary variable which is set to the amount of data that XCICS has returned to the application. Note that this might be slightly less than the limit that you set using the MAXLENGTH option, especially if a double-byte or multi-byte character set is involved, because XCICS does not return a partial character at the end of the data.

- If the NOTRUNCATE option is not specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.
- If the NOTRUNCATE option is specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.

This option is the equivalent of the LENGTH option on the WEB RECEIVE command.

### Options for converting items sent and received

**_CHARACTERSET(DATA-VALUE)**
specifies the character set into which XCICS translates the entity body of the HTTP request before sending. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. XCICS does not support all the character sets named by IANA. The XCICS Internet Guide lists the IANA character sets that are supported by XCICS for code page conversion. For conversion of the request body to take place, the CLIENTCONV option must be allowed to default to CLICONVERT, or specified as NOINCONVERT. Specifying NOCLICONVERT or NOOUTCONVERT suppresses conversion of the request body. If conversion is requested, ISO-8859-1 is used as the default if the CHARACTERSET attribute is not specified.

CLIENTCONV(CVDA)

specifies whether or not XCICS translates the entity body of the HTTP request before sending, and translates the entity body of the server's response. The default is that the entity body is converted both when the request is sent out, and when the response is received (CLICONVERT).

- For the request body, you can use the CHARACTERSET option on this command to specify a character set that is suitable for the server. If conversion is requested (or happens as the default) but you do not specify a character set, the default is that XCICS converts the entity body to the ISO-8859-1 character set.

- For the response body, you do not need to specify the character set used by the server. XCICS identifies this by examining the Content-Type header of the message. If the header does not provide this information, or if the named character set is not supported by XCICS for code page conversion, the ISO-8859-1 character set is used.

- For the application's code page, the default code page for the local XCICS region (as specified in the LOCALCCSID system initialization parameter) is used, or an alternative EBCDIC code page that you specified on the WEB OPEN COMMAND.

Note

For message bodies with non-text media types, XCICS does not convert the message body, even if an appropriate conversion option is specified. CVDA values are:

CLICONVERT

XCICS converts the entity body of the request into the character set that you identify for the server, and converts the entity body of the response into a code page suitable for the application.

NOINCONVERT

XCICS converts the entity body of the request into the character set that you identify for the server. However, XCICS does not convert the entity body of the response, and it is passed to the application in the character set used by the server.

NOOUTCONVERT

XCICS does not convert the entity body of the request, and it is sent to the server in the code page used by the application. However, XCICS does convert the entity body of the response into a code page suitable for the application.

NOCLICONVERT

XCICS does not convert the entity body of the request, and it is sent to the server in the code page used by the application. XCICS does not convert the entity body of the response, and it is passed to the application in the character set used by the server.

Conditions

NOTOPEN

RESP2 values are:

27
Invalid session token.

**INVREQ**

RESP2 values are:

10
Invalid response header.

11
Action code invalid.

13
Close status invalid.

15
Code page conversion failure.

17
Expect-100 request was rejected by the server.

22
Invalid chunk size.

32
Media type invalid.

33
Method does not support a body.

34
Method requires a body.

41
The connection has been closed.

45
The character set specified is invalid.

46
The CLIENTCONV option is invalid.

49
The format of the path option is invalid.

54
The HTTP method is not valid.

63
URIMAP object disabled.

64
Host in URIMAP definition does not match host specified when this session was opened.
HTTP error in response.

The connection has been closed (XCICS sent a Connection: close header to the server).

MEDIATYPE option required.

Pipelining is in progress. WEB CONVERSE command cannot be used.

**LENGERR**

RESP2 values are:

5
The PATHLENGTH option value was not greater than zero.

8
The QUERYSTRLEN option value was not greater than zero.

16
Invalid MAXLENGTH.

36
Partial response body returned. Use additional RECEIVEs to obtain remainder.

50
The FROMLENGTH option value was not greater than zero.

57
The response body exceeds the length specified, and the remainder of the body has been discarded.

58
The status text exceeds the length specified.

59
The STATUSLEN option value was not greater than zero.

**NOTFND**

RESP2 values are:

61
The URIMAP object specified was not found.

**TOKENERR**

RESP2 values are:

47
The document token specified is invalid.
IOERR
RESP2 values are:
42
Socket error.

TIMEDOUT
RESP2 values are:
62
Timeout on socket receive.

NOTAUTH
RESP2 values are:
100
Path barred by security exit.
3.2.2.8 WEB ENDBROWSE HTTPHEADER

WEB ENDBROWSE HTTPHEADER terminates the browse. No information is returned on the ENDBROWSE. The SESSTOKEN option is required if the HTTP header information is part of a response sent to XCICS as an HTTP client.

```
WEB       ENDBROWSE HTTPHEADER
        [ SESSTOKEN(data-value) ]
```

### Options

**SESSTOKEN(DATA-VALUE)**

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the CICS Internet Guide explains the use of the session token.

### Conditions

**INVREQ**

RESP2 values are:

1. The command is being issued in a non-CICS Web support application.
2. The command is being issued for a non-HTTP request.
3. The command is being issued before a WEB STARTBROWSE command is issued.

**NOTOPEN**

RESP2 value is:

27

Invalid session token.
WEB ENDBROWSE FORMFIELD

WEB ENDBROWSE FORMFIELD terminates the browse of a set of name-value pairs in an HTML form. The form is part of the body of an HTTP request being processed by the current XCICS task. No information is returned on the ENDBROWSE.

### Conditions

**INVREQ**

RESP2 values are:

1. The command is being issued in a non-CICS Web support application.

3. The command is being issued for a non-HTTP request.

4. The command is being issued before a WEB STARTBROWSE command is issued.
### 3.2.2.8 WEB EXTRACT

For CICS as an HTTP server, WEB EXTRACT enables an application to obtain information about the most recent HTTP request that has been made to CICS by a Web client and assigned to the application for handling.

For CICS as an HTTP client, when the SESSTOKEN option is specified, the command enables an application to obtain information about a connection that it has opened with a server on the Internet. The information returned to the application comprises global information about the connection, such as the host name of the server and its HTTP version. Information about specific requests made by the application, and responses made by the server, is not available using this command. The WEB RECEIVE command is used to receive information from a server’s response.

<table>
<thead>
<tr>
<th>WEB</th>
<th>EXTRACT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ SCHEME(cvda) ]</td>
</tr>
<tr>
<td></td>
<td>[ HOST(data-area) __HOSTLENGTH(data-value) ]</td>
</tr>
<tr>
<td></td>
<td>[ HTTPMETHOD(data-area) __METHODLENGTH(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ HTTPVERSION(data-area) __VERSIONLEN(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ PATH(data-area) __PATHLENGTH(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ PORTNUMBER(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ QUERYSTRING(data-area) __QUERYSTRLEN(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ REQUESTTYPE(cvda) ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WEB</th>
<th>EXTRACT SESSTOKEN(data-area)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ SCHEME(cvda) ]</td>
</tr>
<tr>
<td></td>
<td>[ HOST(data-area) __HOSTLENGTH(data-value) ]</td>
</tr>
<tr>
<td></td>
<td>[ HTTPVERSION(data-area) __VERSIONLEN(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ PATH(data-area) __PATHLENGTH(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ PORTNUMBER(data-area) ]</td>
</tr>
<tr>
<td></td>
<td>[ URIMAP(data-area) ]</td>
</tr>
</tbody>
</table>

#### Options

**HOST(DATA-AREA)**

For XCICS as an HTTP server, this option specifies a buffer to contain the host component of the URL, as specified either in the Host header field for the request, or in the request line (if an absolute URI was used for the request). The port number is presented separately using the PORTNUMBER option.

For XCICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the host name of the server in the connection identified by the SESSTOKEN option. The port number is presented separately using the PORTNUMBER option.

**HOSTLENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the HOST option, as a fullword binary variable, and is set to the actual length of the data returned to the application. 116 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**HTTPMETHOD(DATA-AREA)**

For XCICS as an HTTP server, this option specifies a buffer to contain the HTTP method string on the request line of the message.

This option is not relevant for XCICS as an HTTP client.

**HTTPVERSION(DATA-AREA)**

For XCICS as an HTTP server, this option specifies a buffer to contain the HTTP version for the Web client, as
stated on its request.

For XCICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the HTTP version of the server in the connection identified by the SESSTOKEN option.

"1.1" indicates HTTP/1.1, and "1.0" indicates HTTP/1.0.

**Note**

XCICS does not make any special provision for a server or Web client that is below HTTP/1.0 level. XCICS behaves as though they were at HTTP/1.0 level, and returns "1.0" as the HTTP version.

If your application program writes HTTP headers that might be unsuitable for a Web client or server at HTTP/1.0 level, or if you intend to send chunked information to the Web client or server (which cannot be received by a client or server at HTTP/1.0 level), your application program should check the HTTP version information.

**METHODLENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the HTTPMETHOD option, as a fullword binary variable, and is set to the actual length of the data returned to the application. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**PATH(DATA-AREA)**

For XCICS as an HTTP server, this option specifies a buffer to contain the path specified in the request line of the message.

For XCICS as an HTTP client (with the SESSTOKEN option), this option specifies a buffer to contain the default path that applies to requests made using the connection. If a URIMAP definition was specified on the WEB OPEN command for the connection, the default path is the path specified in the URIMAP definition. Otherwise, the default path is a single forward slash.

**PATHLENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the PATH option, as a fullword binary variable, and is set to the actual length of the data returned to the application. 256 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**PORTNUMBER(DATA-AREA)**

For XCICS as an HTTP server, this option returns a data area containing the port number specified in the request line of the message.

For XCICS as an HTTP client (with the SESSTOKEN option), this option returns a data containing the port number used to access the server in the connection specified by the SESSTOKEN option.

The value returned in the data area is a fullword binary value.

Well-known port numbers for a service are normally omitted from the URL. If the port number is not present in the URL, the WEB EXTRACT command identifies and returns it based on the scheme. For HTTP, the well-known port number is 80, and for HTTPS, the well-known port number is 443. If a port number is returned which is not the default for the scheme, you need to specify the port number explicitly to gain access to the URL (for example, if you are using this information in a WEB OPEN command).

**QUERYSTRING(DATA-AREA)**

For XCICS as an HTTP server, this option specifies a buffer to contain the query string on the request line of the message. The query string is the value or values encoded after the question mark (?) delimiting the end of the path. The query string is returned in its escaped form.

This option is not relevant for XCICS as an HTTP client.

**QUERYSTRLEN(DATA-AREA)**
specifies the length of the buffer supplied on the QUERY option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the query string). 256 characters is an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

REQUESTTYPE(CVDA)

For XCICS as an HTTP server, this option specifies the type of request received. This option is not relevant for XCICS as an HTTP client. CVDA values are:

HTTPYES
indicates an HTTP request.

HTTPNO
indicates a non-HTTP request.

SCHEME(CVDA)

For both XCICS as an HTTP server, and XCICS as an HTTP client (with the SESSTOKEN option), this option returns the scheme used for the connection between XCICS and the Web client or server. CVDA values are:

HTTP
is the HTTP protocol, without SSL.

HTTPS
is the HTTPS protocol, which is HTTP with SSL.

SESSTOKEN(DATA-VALUE)

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the XCICS Internet Guide explains the use of the session token. For the WEB EXTRACT command, information is returned about the specified connection.

This option is not relevant for XCICS as an HTTP server.

URIMAP(DATA-VALUE)

For XCICS as an HTTP client (with the SESSTOKEN option), this option returns the 8-character name (in mixed case) of any URIMAP definition that was specified on the WEB OPEN command to open the connection specified by the SESSTOKEN option. The INQUIRE URIMAP command can be used to find information about the attributes of this URIMAP definition.

This option is not relevant for XCICS as an HTTP server.

VERSIONLEN(DATA-AREA)

specifies the length of the buffer supplied on the HTTPVERSION option, as a fullword binary variable, and is set to the actual length of the data returned to the application.

Conditions

INVREQ

RESP2 values are:

1
The command is being issued in a non-CICS Web support application.

3

The command is being issued for a non-HTTP request (this is set only if one or more of HTTPMETHOD, HTTPVERSION, or PATH is specified and the request is a non-HTTP request).

**LENGERR**

RESP2 values are:

4
The method exceeds the length specified (METHODLENGTH option).

5
The PATHLENGTH option value was not greater than zero.

6
The HTTP version exceeds the length specified (VERSIONLEN option).

7
The VERSIONLEN option value was not greater than zero.

8
The query string exceeds the length specified (QUERYSTRLEN option).

21
The HOSTLENGTH option value was not greater than zero.

29
The host name exceeds the length specified (HOSTLENGTH option).

30
The path exceeds the length specified (PATHLENGTH option).

**NOTOPEN**

RESP2 values are:

27
Invalid session token.

**ILLOGIC**

RESP2 values are:

9
XCICS logic error.
WEB OPEN enables an application program, through XCICS Web support, to open a connection with a specified host on an HTTP server on the Internet. The host name and scheme can be used from a pre-set URIMAP definition, which also supplies a default path for requests.

When the connection is open, the application program can make HTTP client requests to the server and receive responses from it. XCICS queries the HTTP version of the server (using an OPTIONS request) when the connection is opened, and uses this information for subsequent communications. XCICS also returns the HTTP version information to the application program, to be checked if you plan to write HTTP headers or send chunked information.

The WEB OPEN command drives the XWBOPEN user exit, which can make the connection to the server go through a proxy server, if required.

```
WEB OPEN
URIMAP(data-value) | HOST(data-value)
HOSTLENGTH(data-value) PORTNUMBER(data-value)
[CERTIFICATE(data-value)]
[CIPHERS(data-value) NUMCIPHERS(data-value)]
[CODEPAGE(data-value)]
SESSTOKEN(data-area) HTTPVNUM(data-area) HTTPRNUM(data-area)
```

**Options**

**CERTIFICATE(DATA-VALUE)**

specifies the label of the certificate and key that are to be used as the SSL client certificate during the SSL handshake. Certificate labels can consist of up to 32 alphanumeric characters. This option is only relevant when SCHEME(HTTPS) is specified, if SCHEME(HTTPS) is specified, but the CERTIFICATE option is omitted, the default certificate is used. The certificate and key files must be respectively stored in the XCICS certificates repository (client_ssl_certificate_path) and keys repository (client_ssl_key_path).

**CIPHERS(DATA-VALUE)**

specifies a string of up to 56 hexadecimal digits that is interpreted as a list of up to 28 2-digit cipher suite codes. The cipher suite codes are used when SSL is active for the connection, so this option is only relevant when SCHEME(HTTPS) is specified. They indicate the method of encryption to be used for this connection.

Use the NUMCIPHERS option to specify the number of cipher suite codes in your list. The codes that are actually used depend on what level of encryption has been specified by the ENCRYPTION system initialization parameter.

- With ENCRYPTION=WEAK, the set of possible codes is 03060102
- With ENCRYPTION=MEDIUM, the set of possible codes is 0903060102
- With ENCRYPTION=STRONG, the set of possible codes is 0504352F0A0903060102, or with z/OS 1.6, 050435363738392F3031323330A1613100D915120F0C03060201, which is the full set

If you specify any cipher codes that are not in the default list for the active encryption level, they are ignored.

You can specify the URIMAP option to use this information directly from an existing URIMAP definition, in which case the CIPHERS option is not required. You may still specify the CIPHERS option, and the setting in the URIMAP definition is overridden by any codes that you specify for this option.

**CODEPAGE(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**HOST(DATA-VALUE)**

specifies the host name on the server to which you want to connect. This information can be extracted from a known URL using the WEB PARSE URL command, or from an existing URIMAP definition using the WEB
EXTRACT URIMAP command. You can specify the URIMAP option to use this information directly from an existing URIMAP definition, in which case the HOST option is not required.

An IPv4 address can be used as a host name, but IPv6 addresses are not supported.

If a port number is required, do not include this with the host name, but use the PORTNUMBER option to specify it.

**HOSTLENGTH**(DATA-VALUE)

specifies, as a fullword binary value, the length of the host name. This information is returned if you use the WEB EXTRACT URIMAP command. You can specify the URIMAP option to use this information directly from an existing URIMAP definition, in which case the HOSTLENGTH option is not required.

**HTTPRNUM**(DATA-AREA)

returns the release number for the HTTP version of the server, as a halfword binary value. (HTTPVNUM returns the version number.) For example, if the server is at HTTP/1.0 level, HTTPRNUM returns 0.

**HTTPVNUM**(DATA-AREA)

returns the version number for the HTTP version of the server, as a halfword binary value. (HTTPRNUM returns the release number.) For example, if the server is at HTTP/1.0 level, HTTPVNUM returns 1.

XCICS obtains the HTTP version information when it opens the connection to the server. If the server does not provide HTTP version information, XCICS assumes that it is at HTTP/1.0 level.

If your application program writes HTTP headers that might be unsuitable for a server at HTTP/1.0 level, or if you intend to send a chunked message to the server (which cannot be received by a server at HTTP/1.0 level), your application program should also consult the HTTP version information.

**Note**

XCICS does not make any special provision for a server that is below HTTP/1.0 level. XCICS behaves as though these servers were at HTTP/1.0 level, and returns HTTP/1.0 as the HTTP version.

**NUMCIPHERS**(DATA-VALUE)

specifies, as a halfword binary value, the number of cipher suite codes that you specified for the CIPHERS option.

**PORTNUMBER**(DATA-VALUE)

specifies the port number, as a fullword binary value. You only need to specify the port number if it is not the default for the specified scheme. For HTTP, the default port number is 80, and for HTTPS, the default port number is 443. Port number information can be extracted from a known URL using the WEB PARSE URL command, or from an existing URIMAP definition using the WEB EXTRACT URIMAP command. You can specify the URIMAP option to use this information directly from an existing URIMAP definition, in which case the PORTNUMBER option is not required.

**SCHEME**(CVDA)

specifies the scheme that is to be used for the connection to the server, which can be with or without SSL. CVDA values are:

**HTTP**

is the HTTP protocol, without SSL.

**HTTPS**

is the HTTPS protocol, which is HTTP with SSL. If HTTPS is used, the XCICS address space must be enabled for SSL.

This information can be extracted from a known URL using the WEB PARSE URL command, or from an existing URIMAP definition using the WEB EXTRACT URIMAP command. You can specify the URIMAP option to use this...
information directly from an existing URIMAP definition, in which case the SCHEME option is not required.

**SESSTOKEN(DATA-AREA)**

returns the session token, an 8-byte binary value that uniquely identifies this connection between XCICS and a server. It is returned when the connection is opened successfully. The session token must be used on all XCICS WEB commands that relate to this connection. “Session tokens” in the XCICS Internet Guide explains the use of the session token.

**URIMAP(DATA-VALUE)**

specifies the name (up to 8 characters, in mixed case) of a URIMAP definition that provides the following information:

- The scheme that is to be used for the connection to the server.
- The host name on the server to which you want to connect.
- A port number, if required.
- A path component for the URI, representing the resource on the server that you want to access. This path becomes the default path for WEB SEND or WEB CONVERSE commands relating to this connection, but it can be overridden by specifying another path on the WEB SEND or WEB CONVERSE command.
- The label of the X.509 certificate that is to be used as the SSL client certificate, if required.
- The cipher suite codes that can be used for the connection.

If the URIMAP option is specified, do not specify the CERTIFICATE, HOST, HOSTLENGTH, PORTNUMBER, PORTLENGTH, or SCHEME options. The CIPHERS and NUMCIPHERS options can be omitted or specified in the command; if specified, they override these settings in the URIMAP definition. The URIMAP definition must be for XCICS as an HTTP client, with USAGE(CLIENT) specified.

**Conditions**

**IOERR**

RESP2 values are:

38
Proxy error.

42
Socket error.

**INVREQ**

RESP2 values are:

14
Code page invalid.

22
Invalid chunk received during the initial OPTIONS request.

23
Invalid client certificate.
40
Scheme invalid.

41
Server closed the connection during the initial OPTIONS request.

48
The format of the host option is invalid.

63
URIMAP object disabled.

67
HTTP error in response.

LENGERR
RESP2 values are:

21
Invalid host length.

NOTFND
RESP2 values are:

20
Host name not resolved by name server.

39
Unknown proxy.

61
The URIMAP object specified was not found.

NOTAUTH
RESP2 values are:

100
Host name barred by security exit.

TIMEDOUT
RESP2 values are:

62
Timeout on socket receive.
3.2.2.8 WEB PARSE URL

WEB PARSE URL enables you to break down a URL string into its component parts: scheme, host, port, path and query string. You can use this process to examine the construction of the URL, and to separate out the components. The returned information can be used in the WEB OPEN command to open a client connection to the host named in the URL.

Any escape sequences found in the URL are checked for validity. An escape sequence consists of the percent character (%) followed by two hexadecimal characters. Valid hexadecimal characters are the digits 0 to 9 and the letters A to F.

Note that where the string input to the WEB PARSE URL command has been delimited in the correct way for a URL, the command does not detect invalid content, such as a host name that does not represent an existing host on the Internet, or a character that is not permitted in a URL.

```
WEB     PARSE URL(data-value)
        URLLENGTH(data-value)
        [SCHEMENAME(data-area)]
        [HOST(data-area) HOSTLENGTH(data-area)]
        [PORTNUMBER(data-area)]
        [PATH(data-area) PATHLENGTH(data-area)]
        [QUERYSTRING(data-area) QUERYSTRLEN(data-area)]
```

**Options**

**HOST(DATA-AREA)**

returns the host component of the URL. This can be either an alphanumeric host name or a numeric IP address. If a port number was specified explicitly in the URL, this is returned separately as the PORTNUMBER option.

An IPv4 address can be used as a host name in the WEB OPEN command, but IPv6 addresses are not supported. IPv6 addresses are rejected as invalid by the WEB PARSE URL command because they do not conform to the expected structure.

**HOSTLENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the HOST option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the host name). 116 characters is suggested as an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**PATH(DATA-AREA)**

returns the path component of the URL.

**PATHLENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the PATH option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the path component of the URL). 256 characters is suggested as an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**PORTNUMBER(DATA-AREA)**

returns (as a fullword binary data area) the port number that is specified in, or appropriate for, the URL. Port numbers are sometimes specified explicitly in a URL, following the host name. However, well-known port numbers for a service are normally omitted from a URL. If the port number is not present in the URL, the WEB PARSE URL command identifies and returns it based on the scheme. For HTTP, the well-known port number is 80, and for HTTPS, the well-known port number is 443. If a port number is returned which is not the default for the scheme, you need to specify the port number explicitly to gain access to the URL (for example, if you are using this information in a WEB OPEN command).
QUERYSTRING(DATA-AREA)

returns the query string from the URL. The query string is the value or values encoded after the question mark (?) delimiting the end of the path. The query string is returned in its escaped form.

QUERYSTRLEN(DATA-AREA)

specifies the length of the buffer supplied on the QUERYSTRING option, as a fullword binary variable, and is set to the actual length of the data returned to the application (the query string). 256 characters is suggested as an appropriate size to specify for this data-area. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

SCHEMENAME(DATA-AREA)

returns the scheme component of the URL, as a 16-character data area. Only the HTTP and HTTPS schemes (the HTTP protocol with and without SSL) are supported by XCICS and can be used in a WEB OPEN command.

URL(DATA-VALUE)

specifies the complete URL string.

URLLENGTH(DATA-VALUE)

specifies the length of the buffer containing the URL string, as a fullword binary value.

Conditions

INVREQ

RESP2 values are:

28

Invalid URL.

65

Bad escape sequence.

LENGERR

RESP2 values are:

8

Length of query string returned is greater than QUERYSTRLEN.

29

Length of host name returned is greater than HOSTLENGTH.

30

Length of path returned is greater than PATHLENGTH.
### 3.2.2.8 WEB READ FORMFIELD

WEB READ FORMFIELD retrieves the value of a specific field from an HTML form. The name of the form field is given in the FORMFIELD parameter. The form data is sent as part of an HTTP request being processed by the current XCICS task.

The Web client sends form data in a query string when the GET method is used, and in the entity body when the POST method is used. XCICS can extract the data from either of these locations.

The form data is returned in its unescaped form (see the XCICS Internet Guide for an explanation of this).

If the data that is received represents a file, the uploaded file does not undergo code page conversion.

XCICS only reads form data when XCICS is the HTTP server. The facility is not available when XCICS is an HTTP client.

```
WEB     READ FORMFIELD(data-area) 
      [NAMELENGTH(data-value)] 
      VALUE(data-area) | SET(ptr-ref) 
      VALUELENGTH(data-area) 
      [CLNTCODEPAGE(data-value)] 
      HOSTCODEPAGE(data-value)]
```

### Options

**CLNTCODEPAGE(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**FORMFIELD(DATA-AREA)**

specifies the name of the form field to extract. It is a string of text containing the name of the requested field. The string of text supplied is not case sensitive.

**HOSTCODEPAGE(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**NAMELENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the form field name.

**SET(PTR-REF)**

specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the end of the task.

**VALUE(DATA-AREA)**

specifies the buffer to contain the value of the named form field. XCICS unescapes any escaped characters before placing them in the buffer.

**VALUELENGTH(DATA-AREA)**

specifies the length, as a fullword binary value, of the buffer that is to contain the form field value. If you specify the VALUE option, VALUELENGTH specifies the maximum length of the data that the program accepts. If the value exceeds the length of the buffer, it is truncated. If the length of the form field value is less than the size of the buffer, the form field value is placed in the leftmost byte positions.
Conditions

INVREQ

RESP2 values are:

1
The command is being issued in a non-CICS Web support application.

3
The command is being issued for a non-HTTP request.

11
The client code page cannot be found.

12
The host code page cannot be found.

13
No form data has been supplied in the body of the HTTP request.

14
The code page combination for client and server is invalid.

LENGERR

RESP2 values are:

1
The length in VALUELENGTH is less than or equal to zero.

5
The form field value has been truncated during a read operation because the receiving buffer is too small.

NOTFND

RESP2 values are:

1
The form field with the given name cannot be found.
3.2.2.9 WEB READ HTTPHEADER

WEB READ HTTPHEADER enables an application to extract HTTP header information from a message. When XCICS is an HTTP server, the message is a request from a Web client. When XCICS is an HTTP client, the message is a response from a server, and the SESSTOKEN option is specified.

The XCICS Internet Guide lists HTTP/1.1 headers that you are likely to receive, and gives guidance for the actions that you might take in response to them.

The HTTP header browsing commands (WEB STARTBROWSE HTTPHEADER, WEB READNEXT HTTPHEADER, WEB ENDBROWSE HTTPHEADER) can be used to browse through all the HTTP header information for a message.

<table>
<thead>
<tr>
<th>WEB</th>
<th>READ HTTPHEADER(data-area)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAMELENGTH(data-value)</td>
</tr>
<tr>
<td></td>
<td>VALUE(data-area)</td>
</tr>
<tr>
<td></td>
<td>[SESSTOKEN(data-area)]</td>
</tr>
<tr>
<td></td>
<td>VALUELENGTH(data-area)</td>
</tr>
</tbody>
</table>

**Options**

HTTPHEADER(DATA-AREA)

specifies the name of the HTTP header to be extracted.

NAMELENGTH(DATA-VALUE)

specifies the length, as a fullword binary value, of the HTTP header name.

SESSTOKEN(DATA-VALUE)

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the XCICS Internet Guide explains the use of the session token.

VALUE(DATA-AREA)

specifies the buffer to contain the value of the HTTP header being extracted.

VALUELENGTH(DATA-AREA)

specifies the length of the buffer supplied on the VALUE option, as a fullword binary variable, and is set to the actual length of the data returned to the application. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**Conditions**

INVREQ

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

3

The command is being issued for a non-HTTP request.
43
No HTTP headers found.

LENGERR
RESP2 values are:
1
The length in VALUELENGTH is not greater than zero (XCICS as an HTTP server).
2
The header value has been truncated because the receiving buffer is too small (XCICS as an HTTP server).
35
The length in NAMELENGTH is not greater than zero (XCICS as an HTTP client).
52
The header value has been truncated because the receiving buffer is too small (XCICS as an HTTP client).
55
The length in VALUELENGTH is not greater than zero (XCICS as an HTTP client).

NOTFND
RESP2 value is:
1
The header with the given name could not be found.

NOTOPEN
RESP2 value is:
27
Invalid session token.
WEB READNEXT FORMFIELD retrieves the next name-value pair in an HTML form. The data is returned in its unescaped form (see the XCICS Internet Guide for an explanation of this).

<table>
<thead>
<tr>
<th>WEB</th>
<th>READNEXT FORMFIELD(data-area)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NAMELENGTH(data-area)</td>
</tr>
<tr>
<td></td>
<td>VALUE(data-area)</td>
</tr>
<tr>
<td></td>
<td>VALUELENGTH(data-area)</td>
</tr>
</tbody>
</table>

**Options**

**FORMFIELD(DATA-AREA)**

specifies the buffer to contain the name of the form field being retrieved. The case of the name is as it is stored in the form.

**NAMELENGTH(DATA-AREA)**

specifies the length, as a fullword binary value, of the form field name. If the length of the form field value is less than the size of the buffer, the form field value is placed in the leftmost byte positions.

**VALUE(DATA-AREA)**

specifies the buffer to contain the value corresponding to the name returned in the FORMFIELD data area. XCICS unescapes any escaped characters before placing them in the buffer.

**VALUELENGTH(DATA-AREA)**

specifies the length, as a fullword binary value, of the buffer that is to contain the form field value. If the value exceeds the buffer length, it is truncated. If the length of the form field value is less than the size of the buffer, the form field value is placed in the leftmost byte positions.

**Conditions**

**ENDFILE**

The end of the list of name-value pairs has been reached.

**INVREQ**

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

3

The command is being issued for a non-HTTP request.

4

The command is being issued before a WEB STARTBROWSE FORMFIELD has been issued.
6
A form field has been found that is not in the format NAME:VALUE.

LENGERR

RESP2 values are:

1
NAMELENGTH or VALUELENGTH is less than or equal to zero.

4
The form field name has been truncated during a browse operation because the receiving buffer is too small.

5
The form field value has been truncated because the receiving buffer is too small.
## 3.2.2.9 WEB READNEXT HTTPHEADER

WEB READNEXT HTTPHEADER retrieves the next HTTP header in the list of headers. The SESSTOKEN option is required if the HTTP header information is part of a response sent to XCICS as an HTTP client.

```plaintext
WEB READNEXT HTTPHEADER(data-area)
NAMELENGTH(data-area)
VALUE(data-area)
[SESSTOKEN(data-value)]
VALUELENGTH(data-area)
```

### Options

**HTTPHEADER(DATA-AREA)**

specifies the buffer to contain the name of the HTTP header being extracted.

**NAMELENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the HTTPHEADER option, as a fullword binary data area, and is set to the actual length of the data returned to the application. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

**SESSTOKEN(DATA-VALUE)**

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. “Session tokens” in the XCICS Internet Guide explains the use of the session token.

**VALUE(DATA-AREA)**

specifies the buffer to contain the value of the HTTP header being extracted.

**VALUELENGTH(DATA-AREA)**

specifies the length of the buffer supplied on the VALUE option, as a fullword binary data area, and is set to the actual length of the data returned to the application. If the data exceeds the buffer length, a LENGERR condition is raised and the data is truncated.

### Conditions

**ENDFILE**

The end of the list of HTTP headers has been reached.

**INVREQ**

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

3
The command is being issued for a non-HTTP request.

4

The command is being issued before a WEB STARTBROWSE has been issued.

6

A header has been found which is not in the format NAME:VALUE.

LENGERR

RESP2 values are:

1

NAMELENGTH or VALUELENGTH is less than or equal to zero.

4

The header name has been truncated during a browse operation because the receiving buffer is too small.

5

The header value has been truncated because the receiving buffer is too small.

NOTOPEN

RESP2 value is:

27

Invalid session token.
3.2.2.9 WEB RECEIVE(SERVER)

WEB RECEIVE receives the body of an HTTP request, or all the data for a non-HTTP message, into an application-supplied buffer. The headers for an HTTP request can be examined separately using the WEB HTTPHEADER commands.

The item received by the WEB RECEIVE command can be:

- The body of an HTTP request that a Web client has made to XCICS as an HTTP server. For guidance on the correct use of the WEB RECEIVE command for this purpose, see "Writing Web-aware application programs for XCICS as an HTTP server" in the XCICS Internet Guide.

- A non-HTTP message handled by XCICS Web support facilities, with the user-defined (USER) protocol on the TCPIPSERVICE definition. For guidance on non-HTTP messages, see "XCICS Web support and non-HTTP requests" in the XCICS Internet Guide.

The data is returned in its escaped form. The type of code page conversion used for incoming data received by the XCICS application program can be specified on this command.

```
WEB     RECEIVE
       INTO(data-area) | SET(ptr-ref)
       LENGTH(data-area)
       [ MAXLENGTH(data-value)]
       [ NOTRUNCATE]
       [ TYPE(cvda)]
       [ SERVERCONV(cvda)]
       [ CHARACTERSET(data-value) | CLNTCODEPAGE(data-value)]
       [ HOSTCODEPAGE(data-value)]
```

**Options**

**CHARACTERSET(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**CLNTCODEPAGE(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**HOSTCODEPAGE(DATA-VALUE)**

This option has no effect. Supported for compatibility reason.

**INTO(DATA-AREA)**

specifies the buffer that is to contain the data being received.

**LENGTH(DATA-AREA)**

specifies a fullword binary variable which is set to the amount of data that XCICS has returned to the application. Note that this might be slightly less than the limit that you set using the MAXLENGTH option, especially if a double-byte or multi-byte character set is involved, because XCICS does not return a partial character at the end of the data.

- If the NOTRUNCATE option is not specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.

- If the NOTRUNCATE option is specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.
MAXLENGTH(DATA-VALUE)

specifies the maximum amount, as a fullword binary value, of data that XCICS is to pass to the application. The
MAXLENGTH option applies whether the INTO or the SET option is specified for receiving data. If the data has
been sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to
the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each
individual chunk. The data is measured after any code page conversion has taken place.

If the length of data exceeds the value specified and the NOTRUNCATE options not specified, the data is truncated
to that value, and the remainder of the data is discarded.

If the length of data exceeds the value specified and the NOTRUNCATE option is specified, XCICS retains the
remaining data and can use it to satisfy subsequent RECEIVE commands.

NOTRUNCATE

specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining
data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no
further RECEIVE commands are issued, the data is discarded during transaction termination.)

A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining
data, whatever its length. Alternatively, you can use a series of RECEIVED commands with the NOTRUNCATE
option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no
longer getting a LENGERR response. Bear in mind that if you receive less than the length requested on the
MAXLENGTH option, this does not necessarily indicate the end of the data; this could happen if XCICS needs to
avoid returning a partial character at the end of the data.

SERVERCONV(CVDA)

This option has no effect. Supported for compatibility reason.

SET(PTR-REF)

specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the
next receive command or the end of task.

TYPE(CVDA)

returns the type of request received. CVDA values are:

HTTPYES

indicates an HTTP request.

HTTPNO

indicates a non-HTTP request.

Conditions

INVREQ

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

14

Invalid code page combination.
The SERVERCONV option is invalid.

CHARACTERSET cannot be specified with SERVERCONV(NOSRVCONVERT).

HOSTCODEPAGE cannot be specified with SERVERCONV(NOSRVCONVERT).

Body incomplete.

LENGERR

RESP2 values are:

1

The MAXLENGTH option value was not greater than zero.

36

Partial response body returned. Use additional RECEIVES to obtain remainder.

57

The response body exceeds the length specified, and the remainder of the body has been discarded.

NOTFND

RESP2 values are:

7

Code page not found.

82

Client code page (character set) not found.

83

Host code page (for server) not found.
3.2.2.9 WEB RECEIVE(CLIENT)

WEB RECEIVE for CICS as an HTTP client receives the body of an HTTP response that a server has made. The headers for the HTTP response can be examined separately using the WEB READ HTTPHEADER command or the HTTP header browsing commands. A session token must be included on this command. For guidance on the correct use of the WEB RECEIVE command for XCICS as an HTTP client, see "HTTP client requests from a XCICS application“ in the XCICS Internet Guide.

Code page conversion for the incoming message can be specified on this command.

Note

The RTIMOUT value specified for the transaction that starts the user application indicates the time that the application is prepared to wait to receive the incoming message. (RTIMOUT is specified on the transaction profile definition). When the period specified by RTIMOUT expires, XCICS returns a TIMEDOUT response to the application. An RTIMOUT value of zero means that the application is prepared to wait indefinitely. The default setting for RTIMOUT on transaction profile definitions is zero, so it is important to check and change that setting for applications that are making HTTP client requests.

Note

For XCICS as an HTTP client, the CONVERSE command can be used as an alternative to issuing a WEB SEND command followed by a WEB RECEIVE command.

```
WEB RECEIVE SESSTOKEN(data-value)
  [MEDIATYPE(data-area)]
  [STATUSCODE(data-value)]
  [STATUSTEXT(data-area) STATUSLEN(data-value)]
  INTO(data-area) | SET(ptr-ref)
  LENGTH(data-area)
  [MAXLENGTH(data-value)]
  [NOTRUNCATE]
  [CLIENTCONV(cvda)]
```

Options

CLIENTCONV(CVDA)

This option has no effect. Supported for compatibility reason.

INTO(DATA-AREA)

specifies the buffer that is to contain the data being received.

LENGTH(DATA-AREA)

specifies a fullword binary variable which is set to the amount of data that XCICS has returned to the application. Note that this might be slightly less than the limit that you set using the MAXLENGTH option, especially if a double-byte or multi-byte character set is involved, because XCICS does not return a partial character at the end of the data.

- If the NOTRUNCATE option is not specified, any further data present in the message has now been discarded. A LENGERR response with a RESP2 value of 57 is returned if further data was present.
- If the NOTRUNCATE option is specified, any additional data is retained. A LENGERR response with a RESP2 value of 36 is returned if additional data is available. The description for the NOTRUNCATE option tells you what to do in this case.
MAXLENGTH(DATA-VALUE)

specifies the maximum amount, as a fullword binary value, of data that XCICS is to pass to the application. The MAXLENGTH option applies whether the INTO or the SET option is specified for receiving data. If the data has been sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the chunked message, rather than to each individual chunk. The data is measured after any code page conversion has taken place.

If the length of data exceeds the value specified and the NOTRUNCATE option is not specified, the data is truncated to that value, and the remainder of the data is discarded.

If the length of data exceeds the value specified and the NOTRUNCATE option is specified, XCICS retains the remaining data and can use it to satisfy subsequent RECEIVE commands.

MEDIATYPE(DATA-AREA)

specifies a 56-character data-area to receive the media type (that is, the type of data content) for the body, for example text/xml. See the XCICS Internet Guide for more information about media types.

NOTRUNCATE

specifies that when the data available exceeds the length requested on the MAXLENGTH option, the remaining data is not to be discarded immediately but is to be retained for retrieval by subsequent RECEIVE commands. (If no further RECEIVE commands are issued, the data is discarded during transaction termination.)

A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining data, whatever its length. Alternatively, you can use a series of RECEIVE commands with the NOTRUNCATE option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no longer getting a LENGERR response. Bear in mind that if you receive less than the length requested on the MAXLENGTH option, this does not necessarily indicate the end of the data; this could happen if XCICS needs to avoid returning a partial character at the end of the data.

SET(PTR-REF)

specifies a pointer reference that is to be set to the address of data received. The pointer reference is valid until the next receive command or the end of task.

SESSTOKEN(DATA-VALUE)

specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. “Session tokens” in the XCICS Internet Guide explains the use of the session token.

STATUSCODE(DATA-VALUE)

specifies a data-area to receive the HTTP status code sent by the server. The code is a binary halfword value. Examples are 200 (normal) or 404 (not found). Receiving the status code is optional, but you should always receive and check the status code in the following circumstances:

- If you intend to make an identical request to the server, now or during a future connection.
- If you intend to make further requests to the server using this connection.
- If your application is carrying out any further processing that depends on the information you receive in the response.

The XCICS Internet Guide has basic guidance on appropriate actions for an application to take in response to the status codes for HTTP/1.1.

STATUSTEXT(DATA-AREA)

specifies a data-area to receive any text returned by the server to describe the status code. The text is known as a
reason phrase. Examples are "OK" (accompanying a 200 status code), or "Bad Request" (accompanying a 400 status code). The STATUSLEN option gives the length allowed for the text.

**STATUSLEN**(DATA-VALUE)

specifies, as a fullword binary value, the length of the data-area to receive any text returned by the server to describe the status code (the STATUSTEXT option). The text is known as a reason phrase. Most reason phrases recommended for HTTP are short, but a data-area length of 256 characters is suggested here, in case the server replaces the recommended reason phrase with more detailed information.

**Conditions**

**NOTOPEN**

RESP2 values are:

27
Invalid session token.

**INVREQ**

RESP2 values are:

10
Invalid response header.

15
Code page conversion failure.

22
Invalid chunk received.

41
The connection has been closed.

46
The CLIENTCONV option is invalid.

67
HTTP error in response.

68
Message send with chunked transfer-coding is in progress.

71
Chunked transfer-coding error.

**LENGERR**

RESP2 values are:

16
Invalid MAXLENGTH.
36
Partial response body returned. Use additional RECEIVEs to obtain remainder.

57
The response body exceeds the length specified, and the remainder of the body has been discarded.

58
The status text exceeds the length specified and has been truncated.

59
The STATUSLEN option value was not greater than zero.

IOERR
RESP2 values are:

42
Socket error.

TIMEDOUT
RESP2 values are:

62
Timeout on socket receive.
3.2.2.9 WEB RETRIEVE

The WEB RETRIEVE command retrieves the DOCTOKEN of the document which was sent using an earlier WEB SEND command.

```
WEB RETRIEVE DOCTOKEN(data-area)
```

**Options**

**DOCTOKEN(DATA-AREA)**

specifies a 16-byte buffer to contain the symbolic name of the document to be retrieved.

**Conditions**

**INVREQ**

RESP2 values:

1

The command is issued in a non-CICS Web support application.

2

A WEB SEND command has not been issued.
WEB SEND for CICS as an HTTP client is used to make an HTTP request to a server. A session token must be included on this command. For guidance on the correct use of the WEB SEND command for XCICS as an HTTP client, see "HTTP client requests from a XCICS application" in the XCICS Internet Guide.

**Note**

For XCICS as an HTTP client, the WEB SEND command cannot be used after the connection to the server has been closed. This happens if either the application program, or the server, sends a Connection: close header on a message. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server.

For XCICS as an HTTP client, the CONVERSE command can be used as an alternative to issuing a WEB SEND command followed by a WEB RECEIVE command. However, bear in mind that the WEB CONVERSE command does not support chunked transfer-coding, because this requires a sequence of send actions, and the WEB CONVERSE command provides a single send action.

```plaintext
WEB SEND
SESSTOKEN(data-value)
METHOD(cvda)
[ PATH(data-area) PATHLENGTH(data-value) | URIMAP(data-value) ]
[ QUESTRING(data-area) QUERYSTRLEN(data-value) ]
[ MEDIATYPE(data-value) | [ DOCTOKEN(data-value) | FROM(data-area) FROMLENGTH(data-value)] ]
[ CLIENTCONV(cvda) ]
[ CHARSET(data-value) ]
[ ACTION(cvda) ]
[ CLOSESTATUS(cvda) ]
```

**Options**

**ACTION(cvda)**

This option is used to specify how the message should be sent out. The CVDA value that applies for XCICS as an HTTP client is:

**EXPECT**

makes XCICS send an Expect header along with the request line and headers for the request, and await a 100-Continue response before sending the message body to the server. If a response other than 100-Continue is received, XCICS informs the application program and cancels the send. If no response is received after a period of waiting, XCICS sends the message body anyway.

This option must only be used if your request has a message body.

**CHARACTERSET(data-value)**

This option has no effect. Supported for compatibility reason.

**CHUNKING(cvda)**

is used for controlling the message send when the message is being sent in chunks (known as chunked transfer-coding). The default when the option is not specified is that chunked transfer-coding is not in use.

The content of a chunked message can be divided into chunks in whatever way is most convenient for the application program. The body of a chunked message cannot be formed directly from XCICS documents, so the DOCTOKEN option cannot be used.
Use a separate WEB SEND command with CHUNKING(CHUNKYES) for each chunk of the message. Use the FROM option to specify the chunk of data, and the FROMLENGTH option to specify the length of the chunk. Specify other options for the message, such as the CLOSESTATUS option, on the first WEB SEND command of the sequence, but do not specify them on subsequent commands. When you have sent the last chunk of the data, specify a further WEB SEND command with CHUNKING(CHUNKEND) and no FROM or FROMLENGTH option. XCICS then sends an empty chunk to the recipient to end the chunked message.

If your application program is informed of an error at any point in the chunking process, use the WEB CLOSE command to stop the process and close the connection. The recipient of the chunked message will not receive the final empty chunk, and so should ignore and discard the data that you have sent so far.

The XCICS Internet Guide has a full description of the procedure for chunked transfer-coding, which should be followed in order for your chunked message to be acceptable to the recipient. CVDA values are:

CHUNKNO

Chunked transfer-coding is not used for the message. This is the default if the CHUNKING option is not specified.

CHUNKYES

Chunked transfer-coding is in progress. The data specified by the FROM option represents a chunk of the message.

CHUNKEND

Chunked transfer-coding is complete. No data is specified for this send. XCICS sends an empty chunk to the recipient to complete the chunked message.

Note

1. The method (METHOD option) must be compatible with chunked transfer-coding.

2. When you have begun sending the parts of a chunked message, the application program cannot send any different messages or receive any items until the final empty chunk is sent and the chunked message is complete.

CLOSESTATUS(CVDA)

specifies whether or not a Connection header with the "close" connection option (Connection: close) should be included on the message. The default is that the header is not included. The CVDA values are:

CLOSE

makes XCICS write a Connection: close header for this request. The header notifies the server that the connection should be closed after the server has sent its response to the request. (For a server at HTTP/1.0 level, XCICS achieves the same effect by omitting the Connection: Keep-Alive header.)

If chunked transfer-coding is in use, the CLOSESTATUS(CLOSE) option can be specified on the first chunk of the message, to inform the server that the connection should be closed after the chunked message is complete and a response has been sent.

If chunked transfer-coding is not in use, and the CLOSESTATUS(CLOSE) option is specified on a WEB SEND command, no further messages can be sent to the server until a new connection is made.

NOCLOSE

means that the Connection: close header is not used for this request. If the server is identified as HTTP/1.0, XCICS sends a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), to notify that a persistent connection is desired.

CLIENTCONV(CVDA)
This option has no effect. Supported for compatibility reason.

**DOCTOKEN(DATA-VALUE)**

specifies the 16-byte binary token of a document to be sent as the message body. The document must be created using the XCICS Document interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands), as described in the XCICS Application Programming Guide. You do not need to retrieve the document before sending it. The FROM option provides an alternative way to create a message body.

The body of a chunked message cannot be formed from XCICS documents, so the DOCTOKEN option cannot be used for chunked transfer-coding.

XCICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

**FROM(DATA-AREA)**

specifies a buffer of data which holds the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body, but that option cannot be used for the body of a chunked message.

There is no set maximum limit for the size of the data-area, but its size is limited in practice by storage considerations. The XCICS Internet Guide has more information about these.

**FROMLENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option (the message body). It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

**MEDIATYPE(DATA-VALUE)**

specifies the data content of any message body provided, for example text/xml. The media type is up to 56 alphanumeric characters, including appropriate punctuation. See the XCICS Internet Guide for more information about media types. XCICS checks that the format of the media type is correct, but does not check the validity of the media type against the data content.

For requests which require a body, you must specify the MEDIATYPE option, and there is no default.

For code page conversion to take place, the MEDIATYPE option must specify a type of data content that can be identified as text according to the IANA definitions. For non-text media types, XCICS does not convert the message body.

**METHOD(CVDA)**

specifies the HTTP method for the request.

The GET, HEAD, POST, PUT, TRACE, OPTIONS, and DELETE methods are supported by this command. However, some HTTP servers, particularly HTTP/1.0 servers, might not implement all of these methods.

The XCICS Internet Guide has more information about the correct use of methods, including the HTTP versions that apply to each.

XCICS bars the sending of a message body for methods where it is inappropriate, and requires it for methods where it is appropriate. Chunked transfer-coding is not relevant for methods that do not have a request body. The CVDA values are:

**GET**

Obtain a resource from the server. A request body is not allowed.

**HEAD**

Obtain the HTTP headers, but not the response body, for a resource. A request body is not allowed.

**POST**
Send data to a server. A request body is required.

**PUT**
Create or modify a resource on the server. A request body is required.

**TRACE**
Trace the route of your request to the server. A request body is not allowed.

**OPTIONS**
Obtain information about the server. A request body is allowed, but there is no defined purpose for the body. If you do use a request body, then you must specify a media type.

**DELETE**
Delete a resource on the server. A request body is not allowed.

**PATH(DATA-AREA)**
specifies the path information for the specific resource within the server that the application needs to access.

If the URIMAP option was used to specify an existing URIMAP definition on the WEB OPEN command for this connection, the path specified in that URIMAP definition is the default path for the WEB SEND command. In these circumstances, if you do not specify path information on the WEB SEND command, the path from the URIMAP definition is used. If you specify a different path from that given in the URIMAP definition, this overrides the path from the URIMAP definition.

If the URIMAP option was not used on the WEB OPEN command, there is no default path, and you must provide path information. Path information can be extracted from a known URL using the WEB PARSE URL command.

As an alternative to using the PATH option to provide the path information, you can use the URIMAP option on the WEB SEND command to specify a URIMAP definition from which the path information is taken directly.

**PATHLENGTH(DATA-VALUE)**
specifies the length of the path, as a fullword binary value. If you are providing path information using the PATH option, you need to specify the PATHLENGTH option. Path length information is returned if you use the WEB PARSE URL command to parse a URL.

**QUERYSTRING(DATA-AREA)**
specifies a query string that is to be supplied to the server as part of the request. You do not need to include a question mark (?) at the beginning of the query string; if you do not include it, XCICS supplies it for you automatically when constructing the request. If you include escaped characters in the query string, XCICS passes them to the server in their escaped format.

**QUERYSTRLEN(DATA-VALUE)**
specifies the length of the query string supplied on the QUERYSTRING option, as a fullword binary value.

**SESSTOKEN(DATA-VALUE)**
specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the XCICS Internet Guide explains the use of the session token.

**URIMAP(DATA-VALUE)**
specifies the name (up to eight characters, in mixed case) of a URIMAP definition that provides the path information for the specific resource within the server that the application needs to access. The URIMAP definition must be for XCICS as an HTTP client (with USAGE(CLIENT) specified). Its HOST attribute must be the same as the HOST attribute of the URIMAP definition that was specified on the WEB OPEN command for this connection, or the same as the host name specified in the HOST option on the WEB OPEN command for this connection. A URIMAP
definition specified on the WEB SEND command applies only to this request. If the URIMAP option is specified, do not specify the PATH or PATHLENGTH options.

**Conditions**

**NOTOPEN**

RESP2 values are:

27
Invalid session token.

**INVREQ**

RESP2 values are:

11
Action code invalid.

12
URIMAP and PATH specified. Only one allowed.

13
Close status invalid.

15
Code page conversion failure.

17
Expect-100 request was rejected by the server.

22
Invalid chunk size.

32
Media type invalid.

33
Method does not support a body.

34
Method requires a body.

45
The character set specified is invalid.

46
The CLIENTCONV option is invalid.

49
The format of the path option is invalid.

54
The HTTP method is not valid.

63

URIMAP object disabled.

64

Host in URIMAP definition does not match host specified when this session was opened.

69

Chunked transfer-coding not supported with this HTTP version.

71

Chunked transfer-coding error.

74

The connection has been closed.

76

MEDIATYPE option required.

79

Pipelining is in progress. Expect header cannot be sent.

120

The CHUNKING option is invalid.

121

FROMLENGTH option required.

122

FROM option required.

123

No message body specified. Use FROM, DOCTOKEN or CHUNKING(CHUNKEND).

124

CHUNKING option not specified, FROMLENGTH option required.

125

CHUNKNO specified, FROM option required.

126

CHUNKNO specified, FROMLENGTH option required.

127

CHUNKYES specified, FROM option required.

128

CHUNKYES specified, FROMLENGTH option required.

129

FROM option not allowed with CHUNKING(CHUNKEND).
130
FROMLENGTH option not allowed with CHUNKING(CHUNKEND).

131
FROMLENGTH option specified as zero.

132
METHOD option not allowed for second or subsequent chunks.

133
MEDIATYPE option not allowed for second or subsequent chunks.

LENGERR
RESP2 values are:

5
The PATHLENGTH option value was not greater than zero.

8
The QUERYSTRLEN option value was not greater than zero.

50
The FROMLENGTH option value was not greater than zero.

NOTFND
RESP2 values are:

61
The URIMAP object specified was not found.

TOKENERR
RESP2 values are:

47
The document token specified is invalid.

IOERR
RESP2 values are:

42
Socket error.

NOTAUTH
RESP2 values are:

100
Path barred by security exit.
WEB SEND (server)

WEB SEND for CICS as an HTTP server selects an item for delivery by XCICS Web support or the XCICS business logic interface, and specifies options for sending it. The item can be:

A response to an HTTP request that was made by a Web client, to XCICS as an HTTP server. For guidance on the correct use of the WEB SEND command for this purpose, see "Writing Web-aware application programs for XCICS as an HTTP server" in the XCICS Internet Guide.

A non-HTTP message handled by XCICS Web support facilities, with the user-defined (USER) protocol on the TCPIPSERVICE definition. For guidance on non-HTTP messages, see "XCICS Web support and non-HTTP requests" in the XCICS Internet Guide.

Only one response can be sent during a task. This can be a standard response using one WEB SEND command, or a chunked response using a sequence of WEB SEND commands.

If you attempt to send a second response during the same task, the result depends on whether the ACTION(IMMEDIATE) option or the ACTION(EVENTUAL) option was specified on the WEB SEND command for the first response.

If the ACTION(IMMEDIATE) option was used for the first response, an error is returned when you attempt the second response.

If the ACTION(EVENTUAL) option was used for the first response, the second response overwrites the components of the previous response (status line, HTTP headers and message body). The first response is lost, and the second response is sent.

Each time a request from a Web client is received, XCICS starts a new task to process the request.

```
WEB SEND
   DOCTOKEN(data-value) | [FROM(data-area)]
   [FROMLENGTH(data-value) | CHUNKING(cvda)]
   [MEDIATYPE(data-value)]
   [SERVERCONV(cvda)]
   [CHARACTERSET(data-value) | CLNTCODEPAGE(data-value)]
   [HOSTCODEPAGE(data-value)]
   [STATUSCODE(data-value)]
   [STATUSTEXT(data-area) | STATUSLEN(data-value)]
   [LENGTH(data-value)]
   [ACTION(cvda)]
   [CLOSESTATUS(cvda)]
```

Options

* **ACTION(CVDA)**

specifies how the message should be sent out. The CVDA values that apply for XCICS as an HTTP server are:

  - **IMMEDIATE**
    
    sends the response immediately to the Web client. If CHUNKING is specified, the IMMEDIATE option is assumed. For message sends that do not use chunked transfer-coding, EVENTUAL is the default, which sends the response at end of task.

  - **EVENTUAL**
    
    sends the response to the Web client at end of task. If CHUNKING is specified, the EVENTUAL option is ignored.
This option produces the same behavior as XCICS Web support had in releases before XCICS Transaction Server for z/OS, Version 3 Release 1, and it is the default for XCICS as an HTTP server.

**CHARACTERSET(DATA-VALUE)**

specifies a character set into which XCICS translates the entity body of the item sent by the command before sending. The name of the character set can consist of up to 40 alphanumeric characters, including appropriate punctuation. XCICS does not support all the character sets named by IANA. The XCICS Internet Guide lists the IANA character sets that are supported by XCICS for code page conversion.

When the CHARACTERSET option is specified, SERVERCONV/SRVCONVERT is assumed, so code page conversion of the entity body takes place. As an alternative to selecting the character set yourself, specifying either SERVERCONV/SRVCONVERT, or HOSTCODEPAGE (if allowed), or both, and omitting CHARACTERSET, lets XCICS determine a suitable character set for the message body. The description for the SERVERCONV option tells you what happens in this case.

If you omit all of the code page conversion options, no code page conversion takes place.

**CHUNKING(CVDA)**

is used for controlling the message send when the message is being sent in chunks (known as chunked transfer-coding). The default when the option is not specified is that chunked transfer-coding is not in use. Chunked transfer-coding is only acceptable to HTTP/1.1 clients, and it cannot be used with HTTP/1.0 clients or non-HTTP messages.

The content of a chunked message can be divided into chunks in whatever way is most convenient for the application program. The body of a chunked message cannot be formed directly from XCICS documents, so the DOCTOKEN option cannot be used.

Use a separate WEB SEND command with CHUNKING(CHUNKYES) for each chunk of the message. Use the FROM option to specify the chunk of data, and the FROMLENGTH option to specify the length of the chunk. Specify other options for the message, such as the CLOSESTATUS option, on the first WEB SEND command of the sequence, but do not specify them on subsequent commands. When you have sent the last chunk of the data, specify a further WEB SEND command with CHUNKING(CHUNKEND) and no FROM or FROMLENGTH option.

XCICS then sends an empty chunk to the recipient to complete the chunked message.

If one of the WEB SEND commands fails during the sequence, an error response is returned, and subsequent sends will also fail. The application should handle this situation appropriately. If all of the chunks are sent successfully but the application does not issue the final WEB SEND command with CHUNKING(CHUNKEND), the transaction is ended with abort code AWBP. An incomplete chunked message should be ignored and discarded by the recipient.

The XCICS Internet Guide has a full description of the procedure for chunked transfer-coding, which should be followed in order for your chunked message to be acceptable to the recipient. CVDA values are:

**CHUNKNO**  
Chunked transfer-coding is not used for the message. This is the default if the CHUNKING option is not specified.

**CHUNKYES**  
Chunked transfer-coding is in progress. The data specified by the FROM option represents a chunk of the message.

**CHUNKEND**  
Chunked transfer-coding is complete. No data is specified for this send. XCICS sends an empty chunk to the recipient to complete the chunked message.

**CLNTCODEPAGE(DATA-VALUE)**

This option is supported for migration purposes only. CHARACTERSET replaces it. The action taken by XCICS is the same for both keywords. This means that code page conversion does take place when CLNTCODEPAGE is specified, even if the SERVERCONV option is not specified. Code page conversion does not take place if all the code page conversion options are omitted.
CLOSESTATUS(CVDA)

specifies whether or not XCICS closes the connection after sending the message. The default is that the connection is not closed. The CVDA values are:

CLOSE

XCICS writes a Connection header with the "close" connection option (Connection: close) for this response, and closes the connection with the Web client after sending the response. The header notifies the Web client of the closure. (For a Web client at HTTP/1.0 level, XCICS achieves the same effect by omitting the Connection: Keep-Alive header.)

If chunked transfer-coding is in use, the CLOSESTATUS(CLOSE) option can be specified on the first chunk of the message, to inform the Web client that the connection is closed after the chunked message is complete.

NOCLOSE

means that the Connection: close header is not used for this response, and the connection is kept open. If the Web client is identified as HTTP/1.0 and has sent a Connection header with the "Keep-Alive" connection option (Connection: Keep-Alive), XCICS sends the same header, to notify that a persistent connection will be maintained.

DOCTOKEN(DATA-VALUE)

specifies the 16-byte binary token of a document to be sent as the message body. The document is created using the XCICS Document interface (EXEC XCICS DOCUMENT CREATE, INSERT, and SET commands), as described in the XCICS Application Programming Guide. The FROM option provides an alternative way to create a message body.

The body of a chunked message cannot be formed from XCICS documents, so the DOCTOKEN option cannot be used for chunked transfer-coding.

XCICS documents cannot be converted to the UTF-8 and UTF-16 character encodings.

FROM(DATA-AREA)

specifies a buffer of data which holds the complete message body, or a chunk of the message body. The message body is built by the application program. When you specify the FROM option, use the FROMLENGTH option to specify the length of the buffer of data. The DOCTOKEN option provides an alternative way to create the message body, but that option cannot be used for the body of a chunked message.

There is no set maximum limit for the size of the data-area, but its size is limited in practice by storage considerations. The XCICS Internet Guide has more information about these.

FROMLENGTH(DATA-VALUE)

specifies the length, as a fullword binary value, of the buffer of data supplied on the FROM option. It is important to state this value correctly, because an incorrect data length can cause problems for the recipient of the message.

HOSTCODEPAGE(DATA-VALUE)

specifies the 8-character name of the XCICS (host) code page that was used by the application program for the entity body of the response. When the HOSTCODEPAGE option is specified, SERVERCONV(SRVCONVERT) is assumed, so code page conversion of the entity body takes place. Specifying either SERVERCONV(SRVCONVERT), or CHARACTERSET, or both, and omitting HOSTCODEPAGE, lets XCICS identify the host code page.

If a XCICS document is used to form the response body (DOCTOKEN option), do not specify the HOSTCODEPAGE option, because XCICS identifies the host code page from the XCICS document domain's record of the host code pages for the document.

If a buffer of data is used to form the response body (FROM option), you may need to specify HOSTCODEPAGE. The default if this option is not present is the default code page for the local XCICS region, as set in the LOCALCCSID system initialization parameter. If you require code page conversion but your application has used a different code page, use HOSTCODEPAGE to specify it.

If you omit all of the code page conversion options, no code page conversion takes place.
LENGTH(DATA-VALUE)

This option is supported for migration purposes only. STATUSLEN replaces it.

MEDIATYPE(DATA-VALUE)

specifies the data content of the message body, for example text/xml. The media type is up to 56 alphanumeric characters, including appropriate punctuation. See the XCICS Internet Guide for more information about media types. XCICS checks that the format of the media type is correct, but does not check the validity of the media type against the data content. XCICS does not provide a default. In some circumstances, the media type that you specify affects whether or not code page conversion is carried out; see the description of the SERVERCONV option for more information.

SERVERCONV(CVDA)

This option has no effect. Supported for compatibility reason.

STATUSCODE(DATA-VALUE)

specifies a standard HTTP status code determined by the application program, which is to be inserted on the status line of the HTTP response. The code is a halfword binary value. Examples are 200 (normal response) or 404 (not found). If this option is not specified, XCICS supplies a default of 200.

The XCICS Internet Guide has information about the use of status codes for XCICS Web support. For status codes 204, 205, and 304, a message body is not allowed, and XCICS returns an error response to the command if you attempt to include one. Other than that, XCICS does not check that your use of the status code is appropriate.

STATUSLEN(DATA-VALUE)

specifies the length, as a fullword binary value, of the string supplied on the STATUSTEXT option.

STATUSTEXT(DATA-AREA)

specifies a data-area containing human-readable text to describe the reason for the status code. The text is known as a reason phrase. Examples are "OK" (accompanying a 200 status code), or "Bad Request" (accompanying a 400 status code). The HTTP/1.1 specification (RFC 2616) defines a recommended reason phrase for each status code, but you do not have to use these.

Conditions

INVREQ

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

11

Action code invalid.

13

Close status invalid.

14
Invalid code page combination.

32

Media type invalid.

41

The connection has been closed.

46

The SERVERCONV option is invalid.

72

Status code does not support a message body.

75

Invalid send sequence.

77

Chunk incomplete.

80

CHARACTERSET cannot be specified with SERVERCONV(NOSRVCONVERT).

81

HOSTCODEPAGE cannot be specified with SERVERCONV(NOSRVCONVERT).

85

Chunking cannot be used with non-HTTP messages.

86

Chunking cannot be used with HTTP/1.0 clients.

87

Status code not allowed.

88

Host code page not allowed.

89

Previous send over this connection failed. No further sends permitted.

120

The CHUNKING option is invalid.

121
FROMLENGTH option required.

122
FROM option required.

123
No message body specified. Use FROM, DOCTOKEN or CHUNKING(CHUNKEND).

124
CHUNKING option not specified, FROMLENGTH option required.

125
CHUNKNO specified, FROM option required.

126
CHUNKNO specified, FROMLENGTH option required.

127
CHUNKYES specified, FROM option required.

128
CHUNKYES specified, FROMLENGTH option required.

129
FROM option not allowed with CHUNKING(CHUNKEND).

130
FROMLENGTH option not allowed with CHUNKING(CHUNKEND).

131
FROMLENGTH option specified as zero.

NOTFND

RESP2 values are:

1
The document has not been created or the name is incorrectly specified.

7
Client code page (character set) not found.

83
Host code page (for server) not found.
IOERR

RESP2 values are:

42

Socket error.
3.2.2.9 WEB STARTBROWSE FORMFIELD

WEB STARTBROWSE FORMFIELD signals the start of a browse of a set of name-value pairs in an HTML form that is part of the body of an HTTP request being processed by the current XCICS task.

```
WEB     STARTBROWSE FORMFIELD(data-area)
       NAMELENGTH(data-area)
       [ CLNTCODEPAGE(name) HOSTCODEPAGE(name) ]
```

Options

CLNTCODEPAGE(NAME)

This option has no effect. Supported for compatibility reason.

FORMFIELD(DATA-AREA)

specifies the name of the form field at which browsing is to start. It is a string of text containing the name of the requested field. If a name is not specified, browsing starts at the first name/value pair in the HTML form.

HOSTCODEPAGE(NAME)

This option has no effect. Supported for compatibility reason.

NAMELENGTH(DATA-VALUE)

specifies the length, as a fullword binary value, of the form field name. This field must be specified if FORMFIELD is specified.

Conditions

INVREQ

occurs for the following conditions. RESP2 values are:

1
The command is being issued in a non-CICS Web support application.

3
The command is being issued for a non-HTTP request.

5
There is already a WEB STARTBROWSE in progress.

11
The client code page cannot be found.

12
The host code page cannot be found.

13
No forms data has been supplied in the body of the HTTP request.

14
The code page combination for client and server is invalid.

**LENGERR**

occurs for the following conditions. RESP2 values are:

1

NAMELENGTH or VALUELENGTH is less than or equal to zero.

**NOTFND**

occurs for the following conditions. RESP2 values are:

1

The form field name given in the FORMFIELD parameter could not be found.
WEB STARTBROWSE HTTPHEADER

WEB STARTBROWSE HTTPHEADER signals the start of a browse of the HTTP header information. The SESSTOKEN option is required if the HTTP header information is part of a response sent to XCICS as an HTTP client.

WEB STARTBROWSE HTTPHEADER [SESSTOKEN(data-area)]

Options

SESSTOKEN(DATA-VALUE)

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the CICS Internet Guide explains the use of the session token.

Conditions

ILLOGIC

RESP2 value is:

10

An HTTP header browse is already in progress.

INVREQ

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

3

The command is being issued for a non-HTTP request.

43

No HTTP headers found.

NOTOPEN

RESP2 value is:

27

Invalid session token.
WEB WRITE HTTPHEADER enables an application to add HTTP header information to a message. When XCICS is an HTTP server, the message is a response to a Web client. When XCICS is an HTTP client, the message is a request to a server, and the SESSION option is specified.

Some HTTP headers are created automatically by XCICS if the message requires them, and the application does not need to write these headers. These are:

- Connection
- Content-Length
- Content-Type
- Date
- Expect
- Host
- Server
- TE (written by XCICS but further instances may be added)
- Transfer-Encoding
- User-Agent
- WWW-Authenticate

"HTTP header reference for XCICS Web support" in the XCICS Internet Guide describes the circumstances in which these headers are created. If the user application program writes a header that XCICS also generates, XCICS handles this depending on the situation:

- For XCICS as an HTTP server, if the header is appropriate for a response, XCICS does not overwrite it, but allows the application's version to be used.
- For XCICS as an HTTP client, if the header is appropriate for a request, XCICS does not allow the application to write it, and returns an error response to the WEB WRITE HTTPHEADER command. The exception is the TE header; application programs can add further instances of this header.
- If the header is not normally appropriate for the type of message (request or response), XCICS allows it, as is the case for all user-defined headers. This situation should not occur if your message is compliant with the HTTP specification to which you are working.

The WEB WRITE HTTPHEADER command adds a single header, and you can repeat the command to add further headers. If you write a header that you have already written for the request or response, XCICS adds the new header to the request or response in addition to the existing header.

The name and value of the headers you write, and the circumstances in which you choose to write them, should conform to the requirements of the HTTP specification to which you are working.

If any of the HTTP headers you use might be unsuitable for a Web client or remote server below HTTP/1.1 level, check the version information that they supply before writing those headers. For XCICS as an HTTP server, the WEB EXTRACT command with the HTTPVERSION option gives this information about the Web client. For XCICS as an HTTP client, the options HTTPVNUM and HTTPRNUM that are returned on the WEB OPEN command for the session give this information about the server.

The WEB WRITE HTTPHEADER command cannot be used if the connection with the server or Web client has been closed by either party sending a Connection: close header on a request or response.

For guidance on the correct use of this command:

- When writing headers for an HTTP response sent by XCICS as an HTTP server, see "Writing Web-aware application programs for XCICS as an HTTP server" in the XCICS Internet Guide.
- When writing headers for an HTTP request sent by XCICS as an HTTP client, see "HTTP client requests from a XCICS application" in the XCICS Internet Guide.
- When using chunked transfer-coding to send an HTTP request or response, see "Using chunked transfer-coding

to send an HTTP request or response" in the XCICS Internet Guide. That topic explains the correct procedure for writing trailing headers for a chunked message.

WEB     WRITE HTTPHEADER(data-area) [NAMELENGTH(data-value)] [SESSTOKEN(data-value)] VALUE(data-area) [VALUELENGTH(data-value)]

**Options**

**HTTPHEADER(DATA-AREA)**

specifies the name of the HTTP header to be added to the request or response. The name, which is a string of text, should conform to the standards in the HTTP specification to which you are working.

**NAMELENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the HTTP header name.

**SESSTOKEN(DATA-VALUE)**

For XCICS as an HTTP client, this option is required. It specifies the session token, an 8-byte binary value that uniquely identifies a connection between XCICS and a server. This value is returned by a WEB OPEN command for XCICS as an HTTP client. "Session tokens" in the XCICS Internet Guide explains the use of the session token.

**VALUE(DATA-AREA)**

specifies the value of the named HTTP header. The value, which is a string of text, should conform to the standards in the HTTP specification to which you are working.

**VALUELENGTH(DATA-VALUE)**

specifies the length, as a fullword binary value, of the HTTP header value.

**Conditions**

**INVREQ**

RESP2 values are:

1

The command is being issued in a non-CICS Web support application.

6

Client did not send TE: trailers on request, so trailing headers cannot be used.

19

Header not allowed. Some request headers may only be generated by XCICS.

44

Header not allowed as a trailing header (trailer).

69
Chunked transfer-coding not supported.
70
Trailer header has not been created, so trailing headers cannot be written.
71
Chunked transfer-coding error.
74
Previous send failed.
78
Too late to write trailing headers for this message.

LENGERR

RESP2 values are:
35
The length in NAMELENGTH is not greater than zero.
55
The length in VALUELENGTH is not greater than zero.

NOTOPEN

RESP2 values are:
27
Invalid session token.
3.2.2.1 WRITE

You use this command to write a record to a direct access data set on a local or remote system.

For a VSAM entry-sequenced data set (ESDS) the record is always added at the end of the data set.

For a VSAM KSDS, the record is added in the location specified by the associated key; this location may be anywhere in the data set.

```plaintext
WRITE     DATASET(dsname)    FROM(area)    RIDFLD(ridfld)  
          [ KEYLENGTH(klen) ]  [ SYSID(sysid)]  
          [ LENGTH(arealen)]  [ RBA | RRN ]
```

**Options**

**DATASET(DSNAME)**

specifies the name of the file to be accessed.

**FROM(AREA)**

specifies the record that is to be written to the data set referred to by this file.

**KEYLENGTH(KEYLEN)**

specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified, in which case it is not valid.

**LENGTH**

specifies the length, as a halfword binary value, of the record to be written

**RBA**

specifies that the record identification field specified in the RIDFLD option contains a relative byte address

**RIDFLD(RIDFLD)**

specifies the record identification field. The contents can be a key, a relative byte address, or a relative record number.

**RRN**

specifies that the record identification field specified in the RIDFLD option contains a relative record number.

**SYSID(SYSID)**

specifies the name (1-4 characters) of the system the request is directed to.
3.2.2.1 WRITE JOURNALNAME

WRITE JOURNALNAME writes a journal record from the specified data area to the XCICS log stream that corresponds to specified journal name.

```
WRITE JOURNALNAME(data-value)
  JTYPEID(data-value)
  FROM(data-area)
  [ FLENGTH(data-value) ]
  [ PREFIX(data-value) ]
  [ WAIT ]
  [ NOSUSPEND ]
```

**Options**

**FLENGTH**

specifies, as a full word binary value, the length in bytes of the user data to be built into the journal record.

**FROM(DATA-AREA)**

specifies the user data to be built into the journal record.

**JOURNALNAME(DATA-VALUE)**

specifies a 1- to 8-character journal name. The valid characters for a journal name are the upper-case letters A through Z, the numeric characters 0 through 9, and the special symbols $ @ and #.

The journal name must be defined in the XCICS Journal Control Table (JCT).

**JTYPEID(DATA-VALUE)**

specifies a 2-character identifier to be placed in the journal record to identify its origin.

**NOSUSPEND**

doing nothing: accepted only for compatibility reasons.

**PREFIX(DATA-VALUE)**

specifies the length (halfword binary value) in bytes of the user prefix data to be included in the journal record.

**WAIT**

doing nothing: accepted only for compatibility reasons.
3.2.2.1 WRITE JOURNALNUM

This command creates a journal record.

This command is supported for compatibility with earlier releases of CICS. It is superseded by the WRITE JOURNALNAME command, which you are recommended to use instead.
3.2.2.1 WRITE OPERATOR

Write a message on the system console.

```
WRITE OPERATOR
TEXT(area)
[ TEXTLENGTH(len) ]
```

**Note**

Command is accepted by XCICS precompiler but produce no result.
3.2.2.1 WRITEQ TD

This command is used to write transient data to a predefined symbolic destination. The destination (queue) is identified in the QUEUE option.

```
WRITEQ TD QUEUE(qname)
     FROM(area)
     [ LENGTH(len) ]
     [ SYSID(systemname) ]
```

**Options**

FROM(AREA)

specifies the data that is to be written to the transient data queue.

LENGTH(LEN)

specifies the length (halfword binary value) of the data to be written.

QUEUE(QNAME)

specifies the symbolic name (1-4 alphanumeric characters) of the queue to be written to. The named queue must have been defined to CICS.

SYSID(SYSTEMNAME)

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.
3.2.2.1 WRITEQ TS

This command is used to write transient data to a predefined symbolic destination. The destination (queue) is identified in the QUEUE option.

```
WRITEQ TS  
[  QUEUE(qname) | QNAME(qname) ]
FROM(area) 
[  LENGTH(len) ]
[  ITEM(itemno) ]
[  REWRITE ]
[  MAIN | AUXILIARY ]
[  SYSID(systemname) ]
```

**Options**

**FROM (AREA)**

specifies the data to be written to temporary storage.

**ITEM (ITEMNO)**

specifies, as a halfword binary value, the item number of the logical record to be replaced in the queue (REWRITE option also specified).

**LENGTH (LEN)**

specifies the length (halfword binary) of the data area specified in the FROM option.

**QUEUE(QNAME) | QNAME(QNAME)**

specifies the symbolic name (1-8 alphanumeric characters) of the queue to be deleted. The name must be unique within the XCICS system.

**REWRITE**

specifies that the existing record in the queue is to be overwritten with the data provided. If the REWRITE option is specified, the ITEM option must also be specified. If the specified queue does not exist, the QIDERR condition occurs. If the correct item within an existing queue cannot be found, the ITEMERR condition occurs and the data is not stored.

**SYSID(SYSTEMNAME)**

(remote systems only) specifies the name (1-4 characters) of the system the request is directed.

**AUXILIARY**

specifies that the temporary storage queue is on a direct access storage device in auxiliary storage (accepted but not supported)

**MAIN**

specifies that the temporary storage queue is in main storage (accepted but not supported).
3.2.2.1 XCTL

This command is used to pass control from an application program to another at the same logical level. In this case also data can be passed from the calling program to the called program.

```
   XCTL  PROGRAM( pgmname)
       [ COMMAREA( area) ]
       [ LENGTH( len) ]
```

**Options**

**COMMAREA(AREA)**

specifies a communication area to be made available to the invoked program. In this option the contents of the data-area are passed.

**LENGTH(LEN)**

specifies the length (halfword binary data value) in bytes of the communication area.

**PROGRAM(PGMNAME)**

specifies the identifier (1-8 alphanumeric characters) of the program to which control is to be passed unconditionally.
3.2.2.108.1 REXEC

This command is used to invoke a remote plugin on terminal.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REXEC</td>
<td>Plugin (plugin)</td>
</tr>
<tr>
<td></td>
<td>NAMELENGTH (plugin-length)</td>
</tr>
<tr>
<td></td>
<td>FROM (area)</td>
</tr>
<tr>
<td></td>
<td>LENGTH (arealen)</td>
</tr>
</tbody>
</table>

**Options**

**FROM(AREA)**

writes value of the specified data area, which must be a valid area.

**LENGTH(AREALEN)**

specifies the length (halfword binary) of the data area specified in the FROM option.

**NAMELENGTH(PLUGIN-LENGTH)**

specifies the length (halfword binary) of the area passed in PLUGIN parameters.

**PLUGIN(PLUGIN)**

Plugin to run. We provides two runnable plugin:

- com.hite.x4j.plugin.Execute
  - Used to run on Windows platform all commands available on Command Prompt (cmd.exe)
- com.hite.x4j.plugin.ShowMessage
  - Used to show a message on terminal.

**An example**

I.e. the following program invokes a browser on the remote system (explorer or netscape depending on the system) providing an URL as argument, and then shows a message on the terminal.

```
WORKING-STORAGE SECTION.
01 PLUGIN-EXECUTE PIC X(256) VALUE "com.hite.x4j.plugin.Execute".
01 PLUGIN-SHOW-MESSAGE PIC X(256) VALUE "com.hite.x4j.plugin.ShowMessage".
01 PLUGIN-AREA.
  03 BROWSER-NAME PIC X(8).
  03 FILLER PIC X(1) VALUE SPACE.
  03 PAGE-URL PIC X(247) VALUE SPACES.
  01 PLUGIN-NAME-LENGTH PIC 9(4) COMP.
  01 PLUGIN-AREA-LENGTH PIC 9(4) COMP.
  01 GS-NAME PIC X(64).
PROCEDURE DIVISION.
  EXEC XICCS QUERY (GS-NAME) END-EXEC.
  IF GS-NAME(1:7) = "Windows"
    MOVE "explorer" TO BROWSER-NAME
  ELSE
    MOVE "netscape" TO BROWSER-NAME
```
3.2.2.108.2 RQUERY

This command is used to query directly the terminal connected about:

- terminal emulator version
- hostname
- workstation name
- username
- windows domain name (if present)
- operating system name
- operating system version
- operating system architecture

All the data area involved in this command must be PIC X and at least 64 bytes long.

RQUERY [ VERSION(version) ]
[ HOSTNAME(hostname) ]
[ NAME(name) ]
[ USERNAME(username) ]
[ DOMAIN(domain) ]
[ OSNAME(osname) ]
[ OSVERSION(osversion) ]
[ OSARCH(osarch) ]

Options

DOMAIN(DOMAIN)

If computer belongs to domain, the domain name is returned.
HOSTNAME

Hostname where terminal was running.

OSNAME

OS running on that computers. (ex. Windows XP)

OSARCH

OS architecture (ex. x86)

OSVERSION

OS version (ex. 5.1)

USERNAME

User connected to operating system

VERSION

Terminal emulator version (ex. X4J 2.0.7)

Example

I.e. inquiry to remote system to know remote platform.

```
EXEC XCICS RQUERY OSNAME OS-NAME END-EXEC
```

OS-NAME contents "Windows XP".

### 3.2.2.108.3 SETCLIENT CASE

This command is used to force the terminal to force the specified case.

```
SETCLIENT CASE <UPPER | MIXED | INVERT>
```

#### Options

**INVERT**

Force lower case.

**MIXED**

Accept upper and lower case, how user types.

**UPPER**

Force to upper case
3.2.2.108.4 SETCLIENT SCREENSIZE

This command is used to set the terminal size

```
SETCLIENT SCREENSIZE <SIZE24X80 | SIZE43X80>
```

**Options**

**SIZE24X80**

Force standard terminal size 24 rows x 80 columns.

**SIZE43X80**

Force terminal size to 43 rows x 80 columns.

3.2.2.108.5 SYSLOG

This command is used to write a text message on the syslog files.

```
SYSLOG FROM(area) [ LENGTH(len) ] [ ERROR | WARNING | NOTICE | INFO ]
```

**Options**

**FROM(AREA)**

writes value of the specified data area, which must be a valid area.

**LENGTH(LEN)**

specifies the length (halfword binary) of the data area specified in the FROM option.

**ERROR | WARNING | NOTICE | INFO**

Specifies message severity.

3.2.2.108.6 WRITECON

This command is used to write a text message on the XCICS central console. Eventually a specific XMSG code number may be used.

```
WRITECON FROM(area) [ LENGTH(len) ] [ CODE(codeno) ]
```

**Options**

**CODE(CODENO)**
XMSG code number. (refer to $XFRAMEHOME/etc/xcicsmsg.txt for xmsg codes).

FROM(AREA)
writes value of the specified data area, which must be a valid area.

LENGTH(LEN)
specifies the length (halfword binary) of the data area specified in the FROM option.

3.2.2.108.7 WRITELOG
This command is used to write a text message on the XCICS log files. If HEXDUMP is specified the area is dumped in hexadecimal format on the log file.

```
WRITELOG FROM(area) [LENGTH(len)] [LEVEL(level)] [HEXDUMP]
```

Options

FROM(AREA)
writes value of the specified data area, which must be a valid area.

HEXDUMP
XCICS provides print area hexadecimal format.

LENGTH(LEN)
specifies the length (halfword binary) of the data area specified in the FROM option.

LEVEL(LEVEL)
specifies minimum debugging level to print message. If XCICS debug level is higher than this level, message will be printed.
3.2.2.1 Appendices

3.2.2.109.1 A Supported Conditions

The following conditions can be referred in the HANDLE CONDITION and IGNORE CONDITION statements:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSIDERR</td>
<td></td>
</tr>
<tr>
<td>DSSTAT</td>
<td></td>
</tr>
<tr>
<td>DUPKEY</td>
<td></td>
</tr>
<tr>
<td>DUPREC</td>
<td></td>
</tr>
<tr>
<td>ENDDATA</td>
<td></td>
</tr>
<tr>
<td>ENDFILE</td>
<td></td>
</tr>
<tr>
<td>ENDITION</td>
<td></td>
</tr>
<tr>
<td>ENQBUSY</td>
<td></td>
</tr>
<tr>
<td>ENVDEFERR</td>
<td></td>
</tr>
<tr>
<td>EOC</td>
<td></td>
</tr>
<tr>
<td>EODS</td>
<td></td>
</tr>
<tr>
<td>EOF</td>
<td></td>
</tr>
<tr>
<td>ERROR</td>
<td></td>
</tr>
<tr>
<td>EXPIRED</td>
<td></td>
</tr>
<tr>
<td>FUNCERR</td>
<td></td>
</tr>
<tr>
<td>ILLOGIC</td>
<td></td>
</tr>
<tr>
<td>INBFMH</td>
<td></td>
</tr>
<tr>
<td>INVERRTERM</td>
<td></td>
</tr>
<tr>
<td>INVLC</td>
<td></td>
</tr>
<tr>
<td>INVMPSZ</td>
<td></td>
</tr>
<tr>
<td>INVPARTN</td>
<td></td>
</tr>
<tr>
<td>INVPARTNSET</td>
<td></td>
</tr>
<tr>
<td>INVREQ</td>
<td></td>
</tr>
<tr>
<td>INVTSREQ</td>
<td></td>
</tr>
<tr>
<td>IOERR</td>
<td></td>
</tr>
<tr>
<td>ISCINVREQ</td>
<td></td>
</tr>
<tr>
<td>ITEMERR</td>
<td></td>
</tr>
<tr>
<td>JIDERR</td>
<td></td>
</tr>
<tr>
<td>LENGERR</td>
<td></td>
</tr>
<tr>
<td>MAPFAIL</td>
<td></td>
</tr>
<tr>
<td>NOJBUFS</td>
<td></td>
</tr>
<tr>
<td>NONVAL</td>
<td></td>
</tr>
<tr>
<td>NOPASSBKRD</td>
<td></td>
</tr>
<tr>
<td>NOPASSBKW</td>
<td></td>
</tr>
<tr>
<td>NOSPACE</td>
<td></td>
</tr>
<tr>
<td>NOSTART</td>
<td></td>
</tr>
<tr>
<td>NOSTG</td>
<td></td>
</tr>
<tr>
<td>NOTALLOC</td>
<td></td>
</tr>
<tr>
<td>NOTFND</td>
<td></td>
</tr>
<tr>
<td>NOTOPEN</td>
<td></td>
</tr>
<tr>
<td>OVERFLOW</td>
<td></td>
</tr>
<tr>
<td>PARTNFAIL</td>
<td></td>
</tr>
<tr>
<td>PGMIDERR</td>
<td></td>
</tr>
<tr>
<td>QBUSY</td>
<td></td>
</tr>
<tr>
<td>QIDERR</td>
<td></td>
</tr>
<tr>
<td>QZERO</td>
<td></td>
</tr>
<tr>
<td>RDATT</td>
<td></td>
</tr>
<tr>
<td>RETPAGE</td>
<td></td>
</tr>
<tr>
<td>RTEFAIL</td>
<td></td>
</tr>
<tr>
<td>RTESOME</td>
<td></td>
</tr>
<tr>
<td>SEGDERR</td>
<td></td>
</tr>
<tr>
<td>SELNERR</td>
<td></td>
</tr>
<tr>
<td>SESSBUSY</td>
<td></td>
</tr>
<tr>
<td>SESSIONERR</td>
<td></td>
</tr>
<tr>
<td>SIGNAL</td>
<td></td>
</tr>
<tr>
<td>SYSBUSY</td>
<td></td>
</tr>
<tr>
<td>SYSIDERR</td>
<td></td>
</tr>
<tr>
<td>TERMINERR</td>
<td></td>
</tr>
<tr>
<td>TRANSIDERR</td>
<td></td>
</tr>
<tr>
<td>TSIOERR</td>
<td></td>
</tr>
<tr>
<td>UNEXPIN</td>
<td></td>
</tr>
<tr>
<td>WRBRK</td>
<td></td>
</tr>
</tbody>
</table>

3.2.2.109.2 B INQUIRE/SET COMMANDS

**INQUIRE**

The system programming commands allow you to inquire about the definition and status of most of the resources defined to CICS, and about many elements of the CICS system as well. The resources about which you can inquire are:

**CONNECTION**
- CONNSTATUS
- NETNAME
- SERVSTATUS

**DATASET/FILE**
- ENABLESTATUS

© 2007 H.T.W.C. Srl
OPENSTATUS
EMPTYSTATUS
ADD
BROWSE
DELETE
READ
UPDATE
OBJECT
basename
RECORDSIZE
BLOCKFORMAT
RECORDFORMAT
STRINGS
RECOVSTATUS
KEYLENGTH
KEYPOSITION
TYPE
ACCESSMETHOD
DISPOSITION
REMOTESYSTEM
REMOTENAME

PROGRAM
STATUS
LENGTH
PROGTYPE
COPY
RESCOUNT

TASK
PROGRAM
FACILITY
STARTCODE
TRANSACTION
TWASIZE
USERID
CURRENTPROG

TDQUEUE
Terminal resources can be retrieved sequentially (START/NEXT/END)

NETNAME
The same as TERMINAL

SYSTEM
JOBNAME
CICSSTATUS
RUNAWAY

SET

You can change most of the system elements and resource definitions about which you can inquire, although in general you cannot change as many option values as you can retrieve. Changes are made with a SET command naming the resource or system element.

DATASET/FILE
ENABLED
DISABLED
OPEN
CLOSE

PROGRAM
  ENABLED
  DISABLED
  STATUS

TRANSACTION
  STATUS

TDQUEUE
  TRIGGERLEVEL

TERMINAL
  UCTRAN
  NOUCTRAN
  UCTRANST

CONNECTION
  ACQUIRED
  INSERVICE
  NOTPENDING

SYSTEM
  RUNAWAY
3.2.3 Programmer's tools

XCICS/TS has some command line tools to allow the translation and the compilation of the program sources. These tools and their functionalities are:

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpre</td>
<td>the CICS/XCICS command translator</td>
</tr>
<tr>
<td>xcob</td>
<td>COBOL compiling script</td>
</tr>
<tr>
<td>xpli</td>
<td>PL/I compiling scripts</td>
</tr>
<tr>
<td>xbms2mod</td>
<td>BMS compiler</td>
</tr>
</tbody>
</table>
3.2.3.1 xpre - XCICS translator

XCICS supports a large part of the original IBM's CICS Command Level services. Interfaces of user programs remain unchanged and programs using CICS, when migrated to UNIX/Linux, need only to be recompiled in the new environment.

Program compiling is performed through the following steps:

- Translation or precompiling;
- Compiling

The XCICS precompiler processes programs originally written in COBOL, PL/1 or C-language translates the EXEC CICS instructions coded in user programs into a language code that is compatible with the compiler and provides initialization for service functions.

The output of the precompilation step consists of an intermediate source program that will be compiled with appropriate compiler, producing the final object module.

Syntax

```
xpre [options] <source>
```

Options

- `-v, --verbose` verbose messaging
- `-o, --output <file>` specifies an alternative output file
- `-l, --language=<lang>` specifies source code language. lang can be:
  - COBOL2 (default)
  - COBOL
  - C
  - PL/I
- `-i, --ignore <cmd>` ignore the specified command while precompiling, option can be repeated
- `-t, --stdout` output flushes on standard output
- `-n, --stdin` input is read from standard input (`--stdout` is assumed)
- `-S, --nosevere` xprecob handles error like warnings
- `-h, --help` produces command line help
- `-d, --dump` produces the list of supported commands
- `-D, --debug` enables extra messages generation for debug
- `-P, --pooldir` enables XCICS pooldir addressing
- `-C, --cvda` changes CVDA values
- `-a, --autocommarea` automatically adds DFHCOMMAREA
  `-gbd` if not declared precompiles C language for GDB usage
  `-noautoreturn` prevents automatic EXEC CICS RETURN statemenst at end of PL/1 programs
  `-noexit` prevents exit() to be transformed in EXEC CICS RETURN in C programs
- `-P, --pooldir` provides addressing for internal XCICS areas
Variables to be set are:

\[
\text{XPREHOME}=	ext{XFRAMEHOME}/\text{xpre} \quad \text{PATH}=	ext{PATH}:\text{XPREHOMEPATH} \\
\text{PERL5LIB}=	ext{XPREHOME}/\text{bin} : \text{PERL5LIB}
\]

\text{xpre} returns 0 upon a successful work, even if warning have been found. A negative value is returned if one or more errors have been found.

Messages are logged both on standard error and output file. No listing file is generated as the produced COBOL output contains remarked:

- warnings
- errors
- messages
- command report

**Precompiler Messages**

**PROCEDURE DIVISION USING found**

PROCEDURE DIVISION USING has been found. This is not normal for an XCICS program: a warning is issued.

**no linkage section found: forcing**

source code does not contains LINKAGE section. Xprecob forces it.

**unsupported/wrong command**

xprecob does not support/recognize the specified EXEC CICS command.

**unsupported/wrong option(s)**

xprecob does not support/recognize the specified EXEC CICS option.

**command ignored**

xprecob is ignoring the specified command: no code is generated for the specified EXEC CICS.

**parameter ignored**

specified parameter has no meaning for xprecob.

**option required**

specified option is required for the command.
syntax error
  xprecob does not recognize the format of specified EXEC CICS command.

Requires
  specified command require specified options

no host defined
  A literal is supplied as parameter in EXEC CICS while a working storage field is required.

requires user data
  specified options requires user data
3.2.3.2 xcob - COBOL compiler script

The compiling of an xframe-targeted COBOL source (TP or batch) follows normally the same steps like on the original mainframe environment, in dependence with the program's typology and language. It comprehends:

- source normalization
- SQL precompiling
- XCICS precompiling
- COBOL compiling

In order to give an user-friendly interface to all compiling processes, the "xcob" script is supplied.

SYNOPSIS

xcob [options] <source>

USE

xcob is used to compile both TP and batch COBOL programs. The filename specified as source has to be suffixed with "pre", and all files generated have the base name of the source.

<source> Is the COBOL source to be compiled.

Options

-b handles programs as batch. By default xcob handles programs as TP.
-x handles programs as TP. By default xcob handles programs as TP.
-m handles programs as an IMS online program.
-i generates .INT code
-u generates .GNT code
-c generates a link-editable object (.o)
-z generates a .o object to build a shared library
-a generate debug information
-o<path> output to specified directory
-U<cmd> specifies a user exit filter
-k activates BLL handling
-E<ext> handles input files is with specified extention
-s preprocess for E/SQL
-R oracle|sybase|db2|odbc specifies RDBMS: oracle, sybase, db2 or odbc
-I<path> alternative path for SQL include
-C<path> alternative path for COBOL copybooks
-E xprecob nosevere mode: precompiler doesn't stops on error.
-L let the COBOL compiler generating a list file.
-x let the COBOL compiler to generate a cross-reference file.
-v produces verbose output.
-h produces command line help
-l (lowercase L) activity is logged to '*.cob.log'
-A forces ACUCOBOL compiler invocation
-M forces Microfocus compiler invocation
-D debug: all intermediate files are left in $PWD
-O<options> specifies one or more of the following options:
HOST2UNIX process source with 'host2unix'
MFCOPY uses MF copy/include preprocessor
ACUCOPY use ACU copy/include preprocessor
NOMFCOPY uses xframe internal copy/include preprocessors
NOACUCOPY uses xframe internal copy/include preprocessors
MFCOBOL enables MF COBOL dialect
OSVS enables OSVS dialect (default)
VSC2 enables VSC2 dialect
COBSQL uses COBSQL oracle interface
NOCOBOL uses SQL precompiler directly
NOCICS compile a TP program without EXEC CICS (CALLs)
ANIM same as -a
NOANIM do not generate debug information
COBOL1 source file contains COBOL 1 limitations
SQL same as -s
LOG same as -l
ORACLE same as -R oracle
DB2 same as -R db2
ODBC same as -R odbc
SYBASE same as -R sybase
SQLCA includes SQLCA
CVDA enables precompiler to handle CVDA values (DFHRESP)
POOLDIR includes XCICS POOLDIR addressing
NOLINK no CALL to LINK conversion performed
TRACE enables script trace
NOERROR manage COBOL errors (non severe) as warnings
DL1PRE processs source with d1 precompiler 'dl1pre.pl'
INTGNT same as -u -a
AUTOCA enables precompiler automatic COMMAREA definition
NORETURN inhibits RETURN command
MAKESYN forces COBSQL convert all COMP host variables to COMP-5 host variables (default on Linux)

NOMAKESYN forces COBSQL to do no conversion of variables or host variables from COMP to COMP-5

ALTSPLIT enables alternative parsing routine in precompiler

xcob produces the following files:

<sourcename>.errors List of errors caught during (pre)compiling phases
<sourcename>.lst List file if required (-L)

Moreover it generates debugable and non debugable code in fixed directories receiving this kind of coding.

RETURN CODES

A return code of 0 (zero) is returned if compiling succeeded. Non zero if something went wrong.

CONFIGURATION

xcob may be configured both using environment variables and a configuration file.

The environment

Some environment variables should be previously set to let xcob work properly:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCOBCONF</td>
<td>specifies an alternate configuration file (default is $HOME/etc/xcob.conf)</td>
</tr>
<tr>
<td>XCOB_INT_DEST</td>
<td>specifies output destination for executable code with debug symbols (-i option)</td>
</tr>
<tr>
<td>XCOB_GNT_DEST</td>
<td>specifies output destination for executable code without debug symbols (-u option)</td>
</tr>
<tr>
<td>XCOB_OBJC_DEST</td>
<td>specifies output destination for .o files (-c option)</td>
</tr>
<tr>
<td>XCOB_TP_COBOPT[1-9]</td>
<td>XCOB_TP_COBOPT1 up to XCOB_TP_COBOPT9 are used to pass compiler directives (COBOPTs) to the Microfocus COBOL compiler, while compiling TP programs. Please refer to your MicroFocus COBOL documentation.</td>
</tr>
<tr>
<td>XCOB_BATCH_COBOPT[1-9]</td>
<td>XCOB_BATCH_COBOPT1 up to XCOB_BATCH_COBOPT9 are used to pass compiler directives (COBOPTs) to the MicroFocus COBOL compiler, while compiling batch programs. Please refer to your MicroFocus COBOL documentation.</td>
</tr>
<tr>
<td>XCOB_TP_CBLFLAGS</td>
<td>ACUCOBOL compiler options for TP compiling</td>
</tr>
<tr>
<td>XCOB_BATCH_CBLFLAGS</td>
<td>ACUCOBOL compiler options for BATCH compiling</td>
</tr>
<tr>
<td>XCOB_CBLFLAGS</td>
<td>common ACUCOBOL compiler option for both TP and batch compiling</td>
</tr>
<tr>
<td>XCOB_COPYCOB</td>
<td>path for COBOL copybooks</td>
</tr>
<tr>
<td>XCOB_RDBMS</td>
<td>rdbms type (oracle, db2 or odbc; oracle is the default)</td>
</tr>
</tbody>
</table>
The configuration file

The environment variable listed above may be contained in a file named `xcob.conf` which is loaded at XCOB startup. In this phase, xcob looks for the `xcob.conf` in present working directory and then in `$HOME/etc`. An alternative file may be specified setting the `XCOBCONF` environment with the full path to the desired configuration file.

Sample configuration file 1

```bash
# xcob configuration file
# compile with Microfocus COBOL and Oracle
export COBOL_PATH=$HOME/cpy
export XCOB_RDBMS=oracle
export XCOB_SQLINCLUDE=$HOME/cpy
export XCOB_OPTIONS="MFCOBOL"
```
Sample configuration file 2

```
# xcob configuration file
# compile with ACUCOBOL and IBM DB2 UDB
#
# export COBCPY=/HOME/cpy; $XADSMHOME/src; $XCISHOME/cobcpy
# export XCOB_INIT_DEST=$HOME/objs/int
# export XCOB_GNT_DEST=$HOME/objs/gnt
export XCOB_USREXIT=""
export XCOB_PGMFILE="db2"
export XCOB_SQLINCLUDE=$HOME/cpy
export XCOB_ERROR="user1/mypass@HTDB2"
export XCOB_CBLFLAGS="-Dz -Dv=0 -D7 -Rw ERROR -Rw OVERFLOW -Rw POS"
```

**COBOL Compilers options**

To get best results while migrating from IBM COBOL, we suggest to use the following options for the COBOL compilers:

**Microfocus COBOL**

- IBMCOMP
- NOTRUNC
- DEFAULTBYTE=0

**ACUCOBOL**

- -Dz
- -Dv=0

Please refer to the compiler specific docs for further information.

**Working with DB**

**Oracle DB**

When Oracle precompiling is required, xcob invokes procob. With Microfocus compiler procob call may be embedded in the compiling process (default for MF) or called directly. With ACUCOBOL procob is always called directly.

In the first case options are passed through the file "cobsq1.dir" that must be placed in the directory or the source being compiled. To get further information on cobsq1.dir please refer to Microfocus Server Express docs.

Sample cobsq1.dir

```
csqlt=ora8sqldebug
dbdebug
eendc
dechibit=yes
declare_section=no
userid=xbm/xbm@htora1
include=/usr2/hite/fabrizio/cpy
ixaclen=132
sqlcheck=syntax
maxliteral=160
```
When procob is invoked directly options are passed through the environment XCOB_SQLOPT and Oracle userid is specified through XCOB_DBCID.

I.e.:

```
export XCOB_DBCIID=pli2cob/pli2cob@dbh1
export XCOB_SQLOPT="ireclen=132 sqlcheck=syntax"
```

**IBM DB2 DB**

When DB2 precompiling is required, xcob invokes db2 preparation.

With Microfocus compiler, DB2 precompiler is invoked only through the compiler. With ACUCOBOL DB2 precompiler is invoked directly.

For Microfocus compiler, options and userid are passed through the environment XCOB_SQLOPT in the microfocus format. To get further information on DB2 precompiling through cob command please refer to Microfocus Server Express docs.

For ACUCOBOL compiler the options to DB2 precompiler are always passed through the environment XCOB_SQLOPT, but with the format specified on the IBM DB2 documentation. Userid and password must be specified using XCOB_DBCID.

I.e.:

```
# for Microfocus
export XCOB_SQLOPT="db==SAMPLE pass==db2inst2.db2inst2 bindfile"

# for ACUCOBOL
export XCOB_DBCIID=user1/mypass@udbdb2
export XCOB_SQLOPT=""
```

**ODBC Data Sources**

When ODBC precompiling is required, xcob instructs the compiler to process SQL through ODBC.

To get further information on ODBC precompiling through cob command please refer to Microfocus Server Express E/SQL or ACUCOBOL ACU/SQL docs.

Additional options may be passed through the environment XCOB_SQLOPT.

I.e.:

```
export XCOB_SQLOPT="dbman=odbc autocommit odbctrace=always"
```

**Example**

```
# xcob -s PGM010
xcob PGM010: XPRE precompiler started
xcob PGM010: Precompiled successfully.
xcob PGM010: Oracle E/SQL precompilation started
xcob PGM010: Oracle precompilation successfully
xcob PGM010: COBOL compiler started.
xcob PGM010: Information issued.
```

The compiling was successful and intermediate (.int, .idy, .cbl) files have been generated and located in the correct path.
3.2.3.3 xpli - PL/I compiler script

The compiling of an xframe-targeted PL/I source (TP or batch) follows normally the same steps like on the original mainframe environment, in dependence with the program's typology and language. It comprehends:

- source normalization
- SQL coding conversion
- SQL precompiling
- XCICS precompiling
- PL/I compiling

In order to give an user-friendly interface to all compiling processes, the "xpli" script is supplied.

SYNOPSIS

```
xpli [options] <source>
```

USE

```
xpli is used to compile both TP and batch PL/I programs. The filename specified as source has to be suffixed with ".pre", and all files generated have the base name of the source.
```

Options

```
-b           precompiles for batch use
-s           precompiles SQL
-c           generates object file (.o)
-x           generates executable file
-l           generates shared library (.so|sl) <default>
-o<name>     specifies an output name
-d           compiles for debug
-L           generates listing
-I<path>     specifies an alternate include path
-f           removes target option before generation
```

xpli produces the following files:

```
<sourcename>.errors List of errors caught during (pre)compiling phases
<sourcename>.lst List file if required (-L)
```

Return codes

A return code of 0 (zero) is returned if compiling succeeded. Non zero if something went wrong.

Configuration

```
xpli may be configured both using environment variables and a configuration file.
```

© 2007 H. T. W. C. Sri
The environment

Some environment variables should be previously set to let xpli work properly:

- **XPLI_OBJECT_PATH**: default directory for generated objects
- **XPLI_OUTPUT_PATH**: default directory for generated executables/libraries
- **XPLI_STB_PATH**: default directory for generated stb files (for debugging)
- **XPLI_PROPLI_CONF**: path to pccpli.conf (Liant precompiler conf.)
- **XPLI_RDBMS_TYPE**: database type: db2 or oracle
- **XPLI_INCLUDES_PATH**: path for PLI includes
- **XPLI_PREPROCESSOR_OPTIONS**: options to be passed to the Liant PLI preprocessor
- **XPLI_COMPILER_OPTIONS**: options to be passed to the Liant PLI compiler

The configuration file

The environment variable listed above may be contained in a file named `xpli.conf` which is loaded at xpli startup. In this phase, xpli looks for its configuration in present working directory and then in `$HOME/etc`. An alternative file may be specified setting the `XPLICONF` environment with the full path to the desired configuration file.

Sample configuration file

```
# configuration file for xpli
XPLI_INCLUDES_PATH=$HOME/include
XPLI_OUTPUT_PATH=$HOME/lib
XPLI_PROPLI_CONF=$HOME/etc/pccpli.conf
XPLI_PREPROCESSOR_OPTIONS="-margins 2,7"
XPLI_COMPILER_OPTIONS="-noopt -setstack"
XPLI_STB_PATH=$HOME/objs/stb
```

Example

```bash
# xpli -d PL02.plx
xpli: loading configuration from /home/fabrizio/regtest/src/mixed/xpli.conf
xpli: include PL02.plx PL02.pp
xpli: precompiling CGI calls
xpli: compiling PL/I source PL02.pli
xpli: compiler output is /home/fabrizio/regtest/src/mixed/PL02.o
pli: linking library /home/fabrizio/lib/PL02.so
```

The compiling was successful and a shared library has been generated and located in the correct path.
xbms2mod is the BMS Macro Compiler from XFRAME. It allows to import BMS source originally written on and for mainframe. xbms2mod converts BMS code into a module file, to be used by XCICS.

The name of the map set and maps to import is obtained from the declarations made in the source file. The new generated module file is named according to the XSDF convention.

**Configuration**

In order to configure xbms2mod, following environments must be set:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDF_MODULES_PATH</td>
<td>location of XSDF map modules</td>
</tr>
<tr>
<td>XSDF_INCLUDES_PATH</td>
<td>location of XSDF generated &quot;C&quot; includes</td>
</tr>
<tr>
<td>XSDF_COPIES_PATH</td>
<td>location of XSDF generated COBOL copies</td>
</tr>
<tr>
<td>XSDF_DUMMYSECT_PATH</td>
<td>location of XSDF generated ASM dummies</td>
</tr>
<tr>
<td>XSDF_BMSSRC_PATH</td>
<td>location of the BMS sources generated by XSDF</td>
</tr>
<tr>
<td>XMAPDEFLANG</td>
<td>(L</td>
</tr>
<tr>
<td></td>
<td>when not specified in the XSDF module</td>
</tr>
</tbody>
</table>

The values passed to these environment variables should be customized to adapt it to the specific user’s environment.

**XSDF_MODULES_PATH** is the so-called modules path and it indicates the directory containing the module files. A module file is the file resulting from the import and compilation of the original BMS code. This path is very important and should be set in accordance with the used directory’s structure.

**XSDF_INCLUDES_PATH** is the includes path. It indicates the directory which contains the include files used as header files by programs written in C language.

**XSDF_COPIES_PATH** is the COBOL copy path and it indicates the directory containing the copybooks used by the COBOL programs.

**XSDF_DUMMYSECT_PATH** is the Assembler /370 dsect path. It indicates the directory containing the structures (DSECTs) for Assembler programs.

**XSDF_BMSSRC_PATH** is the BMS sources path and it specifies where the original BMS macros, that are to be imported and compiled by XSDF, are located. This directory is also used by XSDF during the generation of the BMS macro source code, every time one or more maps are to be exported back to the mainframe environment.

Normally it is not necessary to set up all these variables for generating include, copy or dummy section elements: if, for example, only COBOL programs are used then the path variables for C or Assembler structure files need not to be defined.

If xbms2mod cannot locate the configuration environments, or some environment variables are not set, it will assume the following default values:

```
modules_path = $HOME/sdf
includes_path = $HOME/include
copies_path = $HOME/cobcpy
dummysect_path = $HOME/asm
BMS_sources_path = $HOME/bms
```

**SYNTAX**
The syntax of XBMS2MOD is:

```
xbms2mod [options] file
```

**Options**

When no argument is specified, XBMS2MOD displays its syntax and also its options. This possibility to call XBMS2MOD without options can be used as a help display. xbms2mod offers the following options:

- **-v**
  turns verbose mode on. With verbose mode on, XBMS2MOD prints out its settings.

- **-d**
  do nothing (for compatibility with older version)

- **-s**
  shifts map one byte right (default behavior)

- **-l**
  shifts map one byte left

- **-r**
  shifts map one byte right

- **-c**
  enables copybook generation. By default no copybook is generated. The specification of the -c option produces the creation of copybooks. The programming language in which the copybook is generated depends on the LANG statement coded in the definition of the map using BMS macros. If the LANG statement is omitted, it will be assumed that an Assembler copy must be generated.

- **-j**
  enables XJWEB customizable JSPs generation

- **-M<path>**
  specifies an alternate path for modules

- **-I<path>**
  specifies an alternate path for C and PL/I includes

- **-C<path>**
  specifies an alternate path for copybooks

- **-A<path>**
  specifies an alternate path for Assembler DSECT

- **-J<path>**
  specifies a path for JSPs

- **-e**
  assumes ASSEMBLER as default language (default is COBOL)

- **-a**
  forces STORAGE=AUTO mode

- **-k**
  keeps slack field (LENGTH=0) in module

- **-i**
  ignores syntax errors

- **-g**
  generates alternate copybook structure

- **-L<lang>**
  overrides language specification for user structures generation.

  `<lang>` can assume the following values:

  - cobol generates COBOL copybook
  - pli generates PL/I include file
  - clang generates C include file
  - asm generates 370 assembler DSECT

- **-t<type>**
  overrides system specific settings for COBOL copybooks generation.

  `<type>` can assume the following values:

  - legacy generates COMP length fields
  - native generates COMP-5 length fields (default)
overrides settings for structure packing in C includes generation.

<mode> can assume the following values:

default generates OS specific code

gcc generates with GCC compliant code

attribute generates with __attribute__((packed))

pragma generates with #pragma pack

hpppragma generates with old HP-UX 11.11 style pragmas

none generates with no pragma/attribute

Example

```
# xbms2mod -vc -I/home/cfiles/include mapsetA
```

The instruction above will compile mapsetA with turning verbose mode on, generating Copybooks in the following directory:

/home/cfiles/include

Once xbms2mod has finished compiling a map set, it displays the following messages:

```
xbms2mod - BMS compiler version 2.0
Current configuration:
Copybook generation : enabled
First char cutting : disabled
C language header files path : /home/cfiles/include
COBOL copy files path : /users/xsdf/v2/copybook/cobol
Assembler dummy section files path : /users/xsdf/v2/copybook/assembler
```
3.2.4 Debugging guide

XCICS/TS offers some facilities to easily debug user programs.

The easiest facility is the log tracking: when transaction logging is active, all XCICS instructions and parameters are logged in the terminal log file and the program output on stdout and stderr (that means DISPLAYS, PUT and printf commands) is redirected on the same log file too.

This means that user can track its programs, by reading the log of performed instructions and, eventually, of the messages he sent from the program.

Please refer to XCICS configuration books to know more about XCICS logs.

Of course this method is very useful to track unexpected situations that may occur, but to do application development the best solution is the on-line animation system.

This facility enables the debug of user application through interactive debuggers. Every language has its own debugger.
3.2.4.1 Debugging COBOL

COBOL code may be easily debugged using the native compiler features. These feature vary depending on the COBOL compiler used.

XCICS/TS supports the following COBOL compilers:

- Microfocus ServerExpress
- ACUCOBOL

**Debugging with MicroFocus Server Express**

**Cross-session method**

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

**Debug of programs running on current terminal**

The following steps must be followed to start the debug of a the current terminal:

1. connect XCICS using the terminal emulator
2. activate the X/Server
3. start the transaction CEDB with these parameters

```
CEDB ON, DISP=<display address>
```

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the Microfocus animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

**Debug of a specific transaction code**

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

1. connect XCICS using the terminal emulator
2. activate the X/Server
3. start the transaction CEDB with these parameters

```
CEDB ON, DISP=<display address>, TRAN=<transid>
```

Whenever the specified transid is started in the XCICS region, the Microfocus animator will start an XTERM session on the target display.

**Debug of programs running on a specific terminal**

Whenever the programmer wants to debug programs running on background terminal (ie. on a printer terminal), the TERM parameter must be provided, according to this procedure:
- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

CEDB ON, DISP=<display address>, TERM=<termid>

Whenever a transaction is started on the requested terminal identifier, the Microfocus animator will start an XTERM session on the target display.

**Unsolicited Dynamic Attachment Method**

This method allows the normal execution of an XCICS program, and only when the user wants to start to debug, attaches the debugger to the running process.

The following steps must be followed to start the animation of a TP program:

- connect XCICS using the terminal emulator
- enter the transaction:

  ```
  CEDB ON
  ```

- wait for terminal to switch in debugging status
- take note of the PID shown on the screen (the PID is always shown in the X4J status bar)
- open a telnet session and enter:

  ```
  anim <PID>
  ```

Note: when connecting XCICS with a TN3270 emulator (i.e. IBM Personal Communication), obviously no PID number is shown anywhere. In order to easily get the debug status of the terminal and the PID of the working process, simply enter on the screen

```
CEDB
```  

**Deprecated procedure**

The following steps must be followed to start the animation of a TP program:

- connect XCICS using X4J
- start the transaction XANM
- disconnect XCICS
- reconnect XCICS
- send ENTER to XCICS and read the PID number on the status line of X4J

Note: when connecting XCICS with a TN3270 emulator (i.e. IBM Personal Communication), obviously no PID number is shown anywhere. In order to easily get the PID of the working engine, simply enter on the screen

```
XANM PID
```  

before to start the transaction.
Debugging with ACUCOBOL Extend

**XTerm method**

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

**Debug of programs running on current terminal**

The following steps must be followed to start the debug of a the current terminal:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```plaintext
CEDB ON, DISPLAY=<display address>
```

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the ACUCOBOL debugger animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

**Debug of a specific transaction code**

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```plaintext
CEDB ON, DISPLAY=<display address>, TRAN=<transid>
```

Whenever the specified transid is started in the XCICS region, the ACUCOBOL debugger will start an XTERM session on the target display.

**Debug of programs running on a specific terminal**

Whenever the programmer wants to debug programs running on background terminal (i.e. on a printer terminal), the TERM parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```plaintext
CEDB ON, DISPLAY=<display address>, TERM=<termid>
```

Whenever a transaction is started on the requested terminal identifier, the ACUCOBOL debugger will start an XTERM
session on the target display.

**Terminal method**

If an X/Server is not available, ACUCOBOL may connect to a telnet session too. The procedures are the same as described above, except for the presence of the parameter TTY in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a unix session (telnet, ssh or serial) and enter:

  ```
  runcbl --wait
  ```

- start the transaction CEDB with these parameters

  ```
  CEDB ON,TTY=<device>
  ```

The device string is the one shown in field "Terminal" by the runcbl command.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.

**Example**

On the UNIX terminal:

```
# runcbl --wait
Named pipes created
Terminal: pts/14
Waiting for application runtime to open pipe
```

On the XCICS terminal:

```
CEDB ON,TTY=pts/14,TRAN=ACCT
```

**Thin client method**

If XCICS has been configured with the option "allow_acu_thin=yes", programmers have to debug on the ACUCOBOL thin client. To do that, the ACU thin client must be installed on the programmer workstation.

The procedures are the same as described above, except for the presence of the parameter THIN in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a DOS shell on the local PC, and run

  ```
  acuthin --wait --port <port number> --restart
  ```

- start the transaction CEDB with these parameters

  ```
  CEDB ON,THIN=<thin client address>
  ```

The thin client address is the address of the workstation, followed by ":" and the port number.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.
Example
On the workstation at IP 192.168.1.32, open the DOS shell and type:

```
C:\Acucorp\Acucbl700\AcuGT\bin> acuthin --wait --port 8000 --restart
```

On the XCICS terminal:

```
CEDB ON,THIN=192.168.1.32:8000,TRAN=ACCT
```
3.2.4.2 Debugging PL/I

PL/I on-line programs can be debugged using Liant Codewatch(tm).

The following steps must be followed to start the interactive debug of an XCICS program:

- connect XCICS using X4J
- issue "CEDB ON"
- wait for CEDB to complete
- issue "XANM PLISTOP=<MODULE>" to defined to PL/I entry to break on
- read the PID number on the status line of X4J

Once PID is obtained, from a command line session, issue the command

```
xcw <PID> <MODULE>
```

This script is facility provided with XFRAME, which starts codewatch and instructs it to attach the specified process, to load defined module and start debugging.

Please refer to your Liant manual for further information about Codewatch(tm).
3.2.4.3 Debugging C

C language programs may be debugged using every symbolic debugger, which is able to remotely attach a process. In any case XCICS process to attach must be created with this procedure:

- connect XCICS using X4J
- enter CEDB ON
- wait for debug to be activated
- use the PID displayed to connect with your debugger

**Using GDB**

To debug interactively C programs with GDB (GNU Debugger), this procedure should be followed:

1. compile programs with debug symbols (-g option)
2. define an empty entry point (function) to be compiled & linked into a shared library to preload with "load library" command in xcics.conf. I.e.

   ```c
   int gdb_entry_point() { return 0; }
   ```

3. put a call to this function in the module to debug
4. startup XCICS
5. connect XCICS and issue CEDB ON and wait for its messages
6. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
7. from command-line, change PWD to source directory and issue:

   ```bash
   gdb $XFRAMEHOME/bin/xcicsd <PID>
   ```

8. set a breakpoint on the dummy entry point and continue execution. I.e

   ```bash
   gdb> break gdb_entry_point
   gdb> continue
   ```

9. continue using the transaction
10. when the program execution reaches the dummy entry point, debugger will stop
11. at this point GDB have to reload symbols from user library using the sharedlibrary command. I.e.

   ```bash
   gdb> sharedlibrary
   ```

12. GDB is now ready to debug the application

   Remember that user programs is always reloaded from the shared library every time the transaction restart.

   Please refer to GDB documentation for further information about it.

**Using DBX on AIX**

To debug interactively C programs with DBX, this procedure should be followed:
1. compile programs with debug symbols (-g option)
2. startup XCICS
3. connect XCICS and issue CEDB ON
4. for the CEDB messages on terminal.
5. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
6. from command-line, change PWD to source directory and issue:

   dbx -a <PID>

7. set a breakpoint on the module or function to debug and continue execution. I.e

   (dbx) stop in MYPROG
   (dbx) c

8. continue using the transaction
9. when the program execution reaches the dummy entry point, debugger will stop.
10. now it is possible to use dbx commands to debug the program

Remember that user programs is always reloaded from the shared library every time the transaction restart.
Please refer to DBX documentation for further information about it.
3.2.5 Internet guide

XCICS/TS supports provides a Web Support Interface to emulate, as closely as possible, the IBM implementation of CWI (CICS Web Interface) of CICS TS 3.1 as described in the document "IBM CICS Internet Guide" (SC34-6450-04).

This guide explains how to set up and manage XCICS® Web support to enable XCICS regions to act as HTTP servers and HTTP clients, and how to write XCICS application programs that interact with Web clients and servers.

This guide assumes that you are familiar with CICS, either as a system administrator or as a system or application programmer.
3.2.5.1 Concepts and Structure

XCICS Web support is a collection of XCICS services that enable a XCICS region to act both as an HTTP server, and as an HTTP client.

**XCICS as an HTTP server**

When XCICS is an HTTP server, a Web client can send an HTTP request to XCICS and receive a response. The response can be a static response created by XCICS from a document template or static file, or an application-generated response created dynamically by a user application program.

The actions of XCICS as an HTTP server are controlled by:

1. System initialization parameters and resource definitions, including TCPIPSERVICE definitions and URIMAP definitions, which are used to configure XCICS Web support and instruct XCICS how to process requests and responses.
2. XCICS utility programs, which can be used to analyze and process the HTTP requests and responses.
3. User-written application programs, which are used to receive the HTTP requests and provide material for HTTP responses. These can be Web-aware application programs designed for use with XCICS Web support, or non-Web-aware XCICS application programs that were not originally designed for use with XCICS Web support.

The behavior of XCICS Web support as an HTTP server is conditionally compliant with the HTTP/1.1 specification, as described in RFC 2616 specifications.

**XCICS as an HTTP client**

When XCICS is an HTTP client, a user application program in XCICS can initiate a request to an HTTP server, and receive a response from it.

The actions of XCICS as an HTTP client are controlled by user-written application programs. The EXEC XCICS WEB application programming interface includes commands that an application program can use to construct and initiate HTTP requests from XCICS, and to receive responses sent by servers. URIMAP resource definitions can be used to provide information such as a URL or a client certificate label.

**XCICS Web support and non-HTTP messages**

XCICS Web support also supports non-HTTP requests from clients. You can use many of the components of XCICS Web support, including TCPIPSERVICE definitions, XCICS utility programs, and user-written application programs, to provide request handling for any request format that you have defined. Non-HTTP messages that are handled by XCICS Web support use a special protocol (the USER protocol) on the TCPIPSERVICE resource definition, so that they are not subjected to the checks that XCICS carries out for HTTP messages.

In XCICS Transaction Server for z/OS, Version 3 Release 1, this facility is primarily intended to provide support for requests from user-written clients that use nonstandard request formats. The processing that takes place for requests is defined by the user. The facility does not provide specific support for any formally defined protocols which are used for client-server communication.

The support that XCICS Web support provides for non-HTTP messages is not the same thing as the TCP/IP Sockets interface for XCICS. The IP XCICS Sockets interface supplied with z/OS Communications Server has an application programming interface which allows clients to communicate directly with XCICS application programs over TCP/IP. XCICS Web support is not involved with this process.

3.2.5.1.1 Components

XCICS Web support includes some base components that are used for all XCICS Web support tasks, and some task-specific components which you select and configure for individual XCICS Web support tasks.

**Base components**

These are the basic components of the XCICS Web Support interface:
TCP/IP support in XCICS is provided by the Unix/Linux operating systems.

Secure Sockets Layer (SSL) support is used to provide security for the XCICS Web support implementation, and it is provided by OpenSSL packages (www.openssl.org). XCICS supports the Secure Sockets Layer (SSL) 3.0 protocol, and the Transport Layer Security (TLS) 1.0 protocol.

The Sockets listener task (CSOL) detects inbound TCP/IP connection requests, and invokes XCICS Web support by attaching the Web attach task.

Web attach tasks (CWXN, CWXU or an alias) receive data from the Web client and deal with initial processing of requests, including URIMAP matching, code page conversion of the HTTP headers, analysis of the request, and code page conversion of the message body. The tasks also pre-process chunked and pipelined messages received from a Web client. If a static response is delivered (using a URIMAP definition), the Web attach task handles this processing as well.

Resource definitions

These are the resources required for the XCICS Web Support interface execution:

- **TCPIPSERVICE** resource definitions are used to define each port that you use for XCICS as an HTTP server, including security options for connections on that port, and timeout and maximum size limits for inbound requests. They are not used for XCICS as an HTTP client.

- **URIMAP** resource definitions match the URLs of requests from Web clients, or requests to an HTTP server, and provide XCICS with information on how to process the requests.

- **TRANSACTION** resource definitions are used to define alias transactions for HTTP request processing. XCICS supplies a resource definition for a default alias transaction, CWBA. When the Web attach task has completed initial processing for the request, if an application-generated response is to be produced, an alias transaction handles the remaining stages of processing. These include receiving the request, executing the application's business logic, construction of the HTTP response and code page conversion of the HTTP response.

User application programs

The following type of user application programs may be used with the XCICS Web Support interface:

- **Web-aware application** programs can be designed for XCICS Web support, using the EXEC CICS WEB and EXEC CICS DOCUMENT application programming interfaces. For XCICS as an HTTP server, these programs can receive and analyze HTTP requests and provide application-generated responses to the Web client. For XCICS as an HTTP client, a user application program in XCICS can initiate an HTTP request to a server, and receive a response from it.

- **COMMAREA applications**, programs which are designed to be linked to from another program using a COMMAREA interface, can be accessed using XCICS Web support with a converter program to convert their output into HTML for transmission to a Web client. Alternatively, you can write a Web-aware application program that links to a COMMAREA application and uses its output to provide HTTP responses.

Programming interfaces

The XCICS Web Support interface is normally programmed by means of:

- The EXEC XCICS WEB application programming interface interprets and constructs HTTP requests and HTTP responses. Some commands are used for XCICS as an HTTP server, some for XCICS as an HTTP client, and some are for both forms of XCICS Web support.

- The EXEC XCICS DOCUMENT application programming interface constructs XCICS documents to provide the body of a response or request that is sent out from XCICS.

XCICS Web support utility programs

- **Analyzer programs** are associated with TCPIPSERVICE definitions. They are used to interpret an HTTP request if a
URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present. XCICS supplies a default analyzer program DFHWBAAX, which provides basic error handling, and a sample analyzer program DFHWBADX, which supports requests using the URL format that XCICS Web support used before CICS TS 3.1. Either of these analyzers can be used as a basis for your own analyzer program.

- Converter programs can be used to decode an HTTP request and construct input to a user application program. Web-aware application programs do not normally require converter programs, but they might be needed by non-Web-aware applications that were not designed for XCICS Web support. XCICS does not supply a converter program. You can write a number of converter programs and select any converter program in your XCICS region to process a request.

**Document construction facilities**

Response documents may be constructed by means of these facilities:

- UNIX/Linux files can be served as the body of a response to an HTTP request from a Web client.
- Document template support enables message bodies to be built from fragments of HTML which are prepared offline.

### 3.2.5.1.2 Tasks structure

When XCICS Web support is active in a XCICS region, for XCICS as an HTTP server, separate tasks are used to listen for inbound connection requests; to receive data from the socket and perform initial processing; and to cover work carried out by application programs in connection with a request. For XCICS as an HTTP client, only one task applies, which is the task for the application program making the HTTP requests.

**The Sockets listener task (CSOL)**

This is a long running XCICS task. There is one instance of the Sockets listener task in a XCICS system.

The task detects inbound TCP/IP connection requests on all ports defined to XCICS, and invokes the XCICS service associated with the port. When the port is intended for XCICS Web support (that is, HTTP or USER is specified as the protocol), the Web attach task is defined as the transaction in the TCPIPSERVICE resource definition for the port, so the listener attaches that task.

**Web attach tasks (CWXN, CWXU or an alias)**

When the TCPIPSERVICE definition for a port has the protocol HTTP, the default transaction ID for the Web attach task is CWXN. When the protocol is USER, the default is CWXU. An alias can be used instead, but the transaction always executes program DFHWBAX.

When a Web attach task is invoked by the Sockets listener task, the first thing it does is to issue a SOCKET RECEIVE request to receive data from the Web client. When some data has been received, the Web attach task deals with initial processing of the Web client's request.

- For an HTTP request (on the HTTP protocol), the initial processing includes URIMAP matching, code page conversion of the HTTP headers, analysis of the request, and code page conversion of the message body. The task also pre-processes chunked messages received from a Web client. If an analyzer program is used, it is covered by this transaction.

- For a non-HTTP request (on the USER protocol), no initial processing takes place.

If a static response is delivered to an HTTP request (using a URIMAP definition), the Web attach task handles this processing as well. If an application-generated response is required, the Web attach task attaches an alias transaction.

There is an instance of the Web attach task for each individual request from a Web client which is in the initial stages of processing.

Even of persistent connections, the CWXN transaction terminates after a request from the Web client has been passed to the alias transaction, or after the static response has been delivered. The Sockets listener task monitors the socket, and initiates a new instance of CWXN for each request on the persistent connection.
Alias transactions for application-generated responses

When a Web attach task has completed initial processing for a request, if an application-generated response is to be produced, the Web attach task attaches the alias transaction which is specified for the remaining processing stages of that request. XCICS supplies a resource definition for a default alias transaction, CWBA. Alias transactions are not used where a static response is provided.

An alias transaction handles the processing stages for an application-generated response, which include receiving the request, executing the application's business logic, constructing the HTTP response and code page conversion of the HTTP response. If a converter program is used to process the request, it is also handled by the alias transaction. There is an instance of an alias transaction for each HTTP request which is in these stages of processing.

XCICS as an HTTP client

For XCICS as an HTTP client, all activity caused by an application program that makes HTTP client requests is covered by a single task. This includes the application program's actions, the actions of XCICS in sending requests and receiving responses, and socket activity. If the application program links to other programs using the EXEC CICS LINK command, these are also covered by the task. The task has the transaction ID that triggers the application program.

The task remains in the system from the beginning to the end of the application program's activity. The task may involve more than one request and response, and the application program may open and maintain more than one connection to a server. When the task ends, any open connections are automatically closed.

3.2.5.1.3 HTTP requests and response processing (as server)

HTTP requests for XCICS as an HTTP server are initiated by a Web client that makes a request to XCICS. XCICS provides the Web client with responses to the requests it makes. The responses can be created from a static document identified by a URIMAP resource definition, or they can be created dynamically by a user application program.

Processing for XCICS as an HTTP server takes place as follows:

1. XCICS receives a TCP/IP connection request. The XCICS Sockets domain uses the TCPIPSERVICE resource definition for the port to determine that the request should be processed by XCICS Web support. The TCPIPSERVICE definition specifies security attributes to be applied to the request, specifies the timeout setting for receiving the request message, and limits the maximum amount of data that can be received for a single request.

2. XCICS matches the URL for the request to a URIMAP definition, if available. XCICS tries to match the URL specified in the HTTP request to any URIMAP resource definitions that are related to the TCPIPSERVICE definition and apply to XCICS as an HTTP server. If a successful match is made, the URIMAP definition tells XCICS how to process the request. If no match is found, XCICS continues with the default process, which begins at processing stage 5 with the analyzer program.

3. If the URIMAP definition specifies redirection, XCICS redirects the Web client to the specified URL. XCICS composes the redirection message and transmits it to the Web client. This completes the processing for that HTTP request.

4. If the URIMAP definition specifies a static response, XCICS forms and supplies the response. XCICS uses a document template or a UNIX/Linux file, together with appropriate HTTP headers, to form an HTTP response. The response undergoes appropriate code page conversion, and XCICS then transmits the response to the Web client. This completes the processing for that HTTP request.

5. An analyzer program may be run, if the URIMAP definition specifies its use, or if no matching URIMAP definition is found. The analyzer program can interpret the request dynamically, or it can be used for monitoring or audit purposes.

The analyzer program for the TCPIPSERVICE definition must be used in the request processing path if no URIMAP definition has been set up for the request. It might also be needed if you are using a non-Web-aware application program that has special requirements, for code page conversion or for pre-XCICS TS Version 3 compatibility processing. ("Analyzer programs" in topic 2.7 explains these situations.) Otherwise, the use of an analyzer program is optional, but note that the analyzer program is called to process the request if the URIMAP definition is not found.

If an analyzer program is being used, the HTTP request and the HTTP headers are passed to the analyzer program. The analyzer program can interpret the request to determine:

- Which XCICS resources are to be used to service the request.
3.2.5.1.4 HTTP requests and response processing (as client)

For CICS as an HTTP client, CICS is the Web client, and it communicates with an HTTP server. A user-written application program sends requests through XCICS to the HTTP server, and receives the responses from it. XCICS maintains a persistent connection with the server. A session token is used on the commands issued by the application program to identify the connection.

An application program that makes an HTTP request and receives a response must use the EXEC CICS WEB API commands to explicitly direct the interaction with the server. A Web-aware application program could be used to make an HTTP request, and then process the results to provide information to an application that is not Web-aware.

The application program that initiates the HTTP request should be designed to process whatever XCICS receives from the server.
server in response to that request, which might include error responses, redirection to another URL, embedded hypertext links, HTML forms, image source, or other items that request an action from the application program. XCICS can perform code page conversion for requests and responses, if required.

Processing for XCICS as an HTTP client takes place as follows:

1. The application program initiates a connection with the HTTP server through XCICS. The application program does this by issuing the EXEC CICS WEB OPEN command. A URIMAP resource definition that you have created can be referenced to specify the scheme and host name for the connection, or the application program can provide this information. An application may have more than one connection open at a time.

2. XCICS establishes the connection with the server. Using the information provided by the application program, XCICS opens a TCP/IP connection on a socket and contacts the server. During this process, XCICS queries the HTTP version of the server when the connection is opened, and uses this information for subsequent communications. XCICS also returns the HTTP version information to the application program, to be checked if you plan to write HTTP headers or send chunked messages. When the TCP/IP connection is established, XCICS returns a session token to the application program to uniquely identify the connection. This session token is used on all the remaining commands issued by the application program concerning that connection.

3. The application program may write HTTP headers for its request. User-written HTTP headers can be built using the WEB WRITE HTTPHEADER command and stored ready for sending.

4. The application program specifies components of the request line. The request method, path information and query string are specified using the WEB SEND or WEB CONVERSE command. The HTTP version for the request is supplied by XCICS.

5. The application program may produce an entity body for its request. The content of the request is specified on the WEB SEND or WEB CONVERSE command. It can be formed from a XCICS document (using the DOCUMENT interface), or from the contents of a buffer. If the server is at HTTP/1.1, chunked transfer-encoding may be used for a request body formed from the contents of a buffer (but not for a XCICS document).

6. The application program initiates transmission of the request. When the application program issues the WEB SEND or WEB CONVERSE command, the request is passed to XCICS for sending across the connection specified by the session token.

7. XCICS generates some required HTTP headers and adds them to the request, then sends the request to the server. The values for some of the headers are generated directly by XCICS (such as the Date header), and the values of others are based on information provided by the application program (using the WEB SEND command) or by a URIMAP definition.

8. The server receives and processes the request, and provides a response. XCICS passes the response to the application program.

9. The application program examines the response. The WEB READ HTTPHEADER command, or the HTTP header browsing commands, can be used to examine the headers of the response. The WEB RECEIVE or WEB CONVERSE command receives the body of the response (if there is one), which can be processed by the application program, and the response's status code and status text.

10. The application program may initiate further requests and responses. If the server supports persistent connections, the connection identified by the session token remains open for further requests.

11. The application program initiates closing of the connection to the server. When all the requests and responses are completed, the application program issues a WEB CLOSE command, and XCICS closes its end of the TCP/IP connection. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

### 3.2.5.1.5 Session tokens

A session token is an 8-byte binary value that uniquely identifies a connection between XCICS as an HTTP client, and an HTTP server. The use of a session token for each connection means that XCICS Web support can manage multiple connections to servers by different tasks, and also means that an application program can control more than one connection.

A connection begins in response to a WEB OPEN command issued by a user application program. The session token is returned on successful completion of the WEB OPEN command, and used on all the EXEC CICS WEB commands issued by the application program concerning that connection.

Using the connection, the user application program can make HTTP client requests to the server, and receive responses from it. The connection can persist for more than one exchange of a request and a response, until either the application...
If the server terminates the connection, the application program cannot send any further requests using that connection, but it can read the response that the server sent before it terminated the connection. The session token remains valid for use on commands to access that data, until the application issues the WEB CLOSE command. After the WEB CLOSE command is issued, the session token that applies to the connection is no longer valid. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

The maximum number of open client connections, each represented by a session token, that can be present simultaneously in a XCICS region is 32768.

3.2.5.1.6 URLs for XCICS Web Support

In a request URL for a resource that is provided by XCICS Web support, the path component of the URL is up to you. In XCICS Web support, the URIMAP definition or the analyzer program creates the linkage between the request URL and the resource provided by XCICS, so the URL does not need to have any direct relationship to the XCICS resource. However, you can design the URL to provide information for processing or administrative purposes.

URLs for application-generated responses

In XCICS Transaction Server for z/OS, Version 3 Release 1, information in a request URL can be used by analyzer programs and by user-written application programs.

Where an analyzer program is used in the processing path for the request, you can design a URL that tells the analyzer program which programs and transaction to specify for further processing. The XCICS-supplied sample analyzer program DFWBADX analyzes URLs with a path component in the format /converter/alias/program/other path information, where converter names the converter program (if any), alias names the alias transaction, program names the user application program, and other path information gives additional information that is not used by the analyzer.

A Web-aware application program which is providing a response can also use information from the path component of the URL. The path component can be extracted by the application using the WEB EXTRACT command, and analyzed to determine the appropriate action. For example, the path component can be used to specify a particular function provided by the application. Alternatively, if the Web-aware application is providing a front end for more than one other application, the path component of the URL can identify the application to which the request applies.

For application-generated responses that are managed using URIMAP definitions, the path components of URLs can be designed to map multiple request URLs to the same application. You can do this by making the path components of the URLs begin in the same way, and creating a single URIMAP definition with a wildcard to map all the request URLs to a single resource. For example, all requests whose path begins with /staffapps/ordering/ could be mapped to a particular XCICS application, by creating a URIMAP definition that specifies the path /staffapps/ordering/* and specifies the relevant application. The application can then extract and analyze information in the remainder of the URL to determine the appropriate action for each request.

URLs for static responses

In XCICS Web support, the URL does not need to have any direct relationship to the XCICS resource. For static responses, this means that the URL does not have to contain the full path to the file that provides the response. Instead, the URIMAP definition matches the request URL to the appropriate file.

However, where flat files are used as the static responses, you could decide to design the path components of the request URLs so that they match the directories used on the file system. If all the files provided by XCICS Web support are located in subdirectories of the same directory, such as the HOME directory of the XCICS region userid, you might want to omit this directory and make the request URLs match the remainder of the paths to the files. For example, if your HOME directory is /var/www/html, and you want to provide the following HFS files as static responses:

```
/var/www/html/FAQs/ordering.html
/var/www/html/help/directory/viewing.html
```

you could use request URLs such as:

```
http://www.example.com/FAQs/ordering.html
http://www.example.com/help/directory/viewing.html
```
Remember that the path components of URLs are case-sensitive, and so are file names. URLs are normally specified in lower case. Take care to use the correct case when specifying each item in the URIMAP definition, especially if the file name is in mixed case and the URL is in lower case.

You might want to make your request URLs match your HFS directory structure:

- To make administration of resources more straightforward.
- To follow standard practice for Web servers.
- To reduce the number of URIMAP definitions that you need to create.

You can create a single URIMAP definition with wildcards, to deliver multiple static responses using the path matching mechanism. This is possible where the path component of the URL for all those static responses begins in the same way, and where the files for the static responses are stored in the same directory. Wildcards are used at the end of the path component of the URL, and also at the end of the file path for the file. In the example above, all the HTML documents stored in the FAQs directory could be provided by a single URIMAP definition that specifies the path /faqs/* and specifies the file attribute as /var/www/html/FAQs/*.

**Query strings**

A query string in a request URL can be used to select alternative URIMAP definitions. To use a query string for URIMAP matching, the complete and exact query string must be specified in the path attribute of the URIMAP definition, together with the path itself.

For application-generated responses, the application can extract and analyze information from a query string, using the WEB EXTRACT command or the WEB READ FORMFIELD command. This can be done whether or not the query string has been used for URIMAP matching.

If you are providing a static response with a document template, XCICS automatically passes the content of the query string into the named XCICS document template as a symbol list. If you want to use the content of the query string in the document template, you can include appropriate variables in your document template to be substituted for the content of the query string. This happens only if the query string has not already been used for URIMAP matching.

### 3.2.5.1.7 Chunked transfer-coding

Messages using chunked transfer-coding can be sent and received by XCICS.

XCICS as an HTTP server can receive a chunked message as a request, or send one as a response. XCICS as an HTTP client can send a chunked message as a request, or receive one as a response. XCICS Web support handles these different cases as follows:

- **When XCICS as an HTTP server receives a chunked message as an HTTP request,** XCICS Web support recognizes the chunked encoding. It waits until all the chunks are received (indicated by the receipt of a chunk with zero length), and assembles the chunks to form a complete message. The assembled message body can be received by a user application program using the WEB RECEIVE command.

- **You can limit the total amount of data that XCICS accepts for a single chunked message,** using the MAXDATALEN option on the TCPIPSERVICE resource definition that relates to the port on which the request arrives.

- **Trailing headers from the chunked message can be read using the HTTP header commands.** The Trailer header identifies the names of the headers that were present as trailing headers. If you are using an analyzer program in the processing path for the request, note that trailing headers are not passed to the analyzer program along with the main request headers.

- **When XCICS as an HTTP client receives a chunked message as a response to an application program's request,** the chunks are also assembled before being passed to the application program as an entity body, and any trailing headers can be read using the HTTP header commands. You can specify how long the application will wait to receive the response, using the RTIMOUT attribute of the transaction profile definition for the transaction ID that relates to the application program.

- **When XCICS sends a chunked message,** either as an HTTP server (response) or as an HTTP client (request), the application program can specify chunked transfer-coding by using the CHUNKING(CHUNKYES) option on the WEB SEND command for each chunk of the message. The message can be divided up in whatever way is most convenient for the application program. XCICS sends each chunk of the message, adding appropriate
HTTP headers to indicate to the recipient that chunked transfer-coding is being used. The application program issues WEB SEND with CHUNKING(CHUNKEND), to indicate the end of the message. XCICS then sends an empty chunk (containing a blank line) to end the chunked message, along with any trailing headers that are wanted.

3.2.5.1.8 Persistent connections

Persistent connections are the default behaviour for XCICS Web support.

Before XCICS TS 3.1, the connection behavior was that XCICS would normally close the connection when data had been received from the Web client, unless the Web client sent a Connection: Keep-Alive header.

Now, when a connection is made between a Web client and XCICS as an HTTP server, or between XCICS as an HTTP client and a server, XCICS attempts to keep the connection open by default.

When XCICS is the HTTP server, the persistent connection is closed if:

- The user-written application that is handling the request closes the connection (by specifying the CLOSESTATUS(CLOSE) option on the WEB SEND command).
- The Web client closes the connection (notified by a Connection: close header).
- The Web client is an HTTP/1.0 client that does not send a Connection: Keep-Alive header.
- The timeout period is reached (indicating that the connection has failed, or that the Web client has deliberately exited the connection).

Otherwise, XCICS leaves the persistent connection open for the Web client to send further requests. If there is a persistent connection with the client, XCICS does not keep the connection open after an error response is sent through a Web error program.

When XCICS is the HTTP client, the persistent connection is closed if:

- The server closes the connection (notified by an HTTP/1.1 server sending a Connection: close header, or an HTTP/1.0 server failing to send a Connection: Keep-Alive header).
- The user application program closes the connection (by specifying the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command, or by issuing a WEB CLOSE command).
- End of task is reached and the connection has not yet been closed.

If the application program needs to test whether the server has requested termination of the connection, the READ HTTPHEADER command can be used to look for the Connection: close header in the last message from the server. If the server requests closure of the connection, but the application program has not yet issued a WEB CLOSE command, XCICS closes the connection but maintains the data relating to the connection (including the last response received from the server and its HTTP headers). The application program can continue to use that data until it issues a WEB CLOSE command or end of task is reached.

The WEB CLOSE command for XCICS as an HTTP client does not cause XCICS to notify the server that the persistent connection should be terminated. It only makes XCICS close the connection. It is good behavior to include a Connection: close header on the final request that you make to the server. Using this header means that the server can close its persistent connection immediately after sending the final response, rather than waiting to see if XCICS sends further requests and then timing out. To include this header, specify the CLOSESTATUS(CLOSE) option on the WEB SEND or WEB CONVERSE command.

If XCICS as an HTTP client is communicating with an HTTP/1.0 server, XCICS automatically sends Connection: Keep-Alive headers on HTTP messages. The application program that requested the connection does not need to provide these. Keep-Alive informs the server that a persistent connection is desired.

3.2.5.1.9 XCICS HTTP/1.1 compliance

XCICS Web support complies on your behalf with many of the requirements in the HTTP/1.1 specification.

Most of the behavior described here applies whether you are using URIMAP definitions or an analyzer program to handle HTTP requests for XCICS as an HTTP server, but there are a few exceptions. If you are not using URIMAP definitions, you should note that some changes might be required in the behavior of your analyzer program to ensure HTTP/1.1 compliance.
XCICS checks inbound messages for compliance with HTTP/1.1, and handles or rejects non-compliant messages.

The checks are made immediately on receipt of the request, before a URIMAP definition or analyzer program is involved. A variety of basic acceptance checks are made, and if the message is unacceptable and it is not appropriate for XCICS to handle the problem itself, an error response is returned to the Web client where possible. These basic acceptance checks are not carried out for HTTP/1.0 messages, nor are they carried out if the USER protocol (instead of the HTTP protocol) is specified on the TCPIPSERVICE definition.

Note: XCICS requires the Content-Length header on all inbound HTTP/1.1 messages that have a message body. If a message body is present but the header is not provided, or its value is inaccurate, the socket receive for the faulty message or for a subsequent message can produce unpredictable results. For HTTP/1.0 messages that have a message body, the Content-Length header is optional.

XCICS follows the HTTP/1.1 rules for comparison of URLs.

Scheme names and host names are compared case-insensitively, but paths are compared case-sensitively. All components are unescaped before comparison. XCICS also handles the absolute URI form in requests (where the host name is specified in the request line). Note that when you use an analyzer program instead of a URIMAP definition to handle an inbound HTTP request, if you need to achieve compliance in this respect, you must code your analyzer program to perform URL comparison according to the rules stated in the HTTP/1.1 specification.

XCICS provides a suitable HTTP version number in the start line of outbound messages.

XCICS normally identifies the message as HTTP/1.1, unless it knows that the Web client or server is at HTTP/1.0 level. In that case, XCICS identifies the message as HTTP/1.0. Requests by a Web client to upgrade from one HTTP version to another, or to a different security protocol, are not supported by XCICS.

On outbound HTTP/1.1 messages, XCICS supplies the HTTP headers that should normally be present for the message to be compliant with HTTP/1.1.

Some headers are also produced by XCICS which are not required for compliance, but support actions that the application program has requested (such as the Expect header). The headers for compliance are supplied for messages sent by both Web-aware applications and non-Web-aware applications. They are not supplied if the USER protocol (instead of the HTTP protocol) is specified for the TCPIPSERVICE definition.

XCICS handles OPTIONS requests from Web clients and makes an appropriate response.

OPTIONS * (an enquiry on the whole server, rather than a specific resource) is the only format accepted. The response contains basic information about XCICS as an HTTP server (the HTTP version and server software description). No user application program is involved. When XCICS is an HTTP client, it uses an OPTIONS request to check the HTTP version of the server, and returns this information on the WEB OPEN command.

XCICS accepts inbound messages with chunked transfer-coding and assembles them for you, and supports your use of chunked transfer-coding to send outbound messages.

Trailing headers for chunked messages can be manipulated through the HTTP header read, write and browse commands. This means that applications can receive and handle chunked messages as they would normal messages. XCICS also supports sending chunked messages out from a user application, but you must ensure that you follow the correct procedure to make the chunked message acceptable to the recipient.

XCICS supports virtual hosting (multiple host names at the same IP address).

Support for virtual hosts is based on your URIMAP definitions.

Connections are persistent by default.
This is the case for both XCICS as an HTTP server and XCICS as an HTTP client. If XCICS is communicating with a Web client or server at HTTP/1.1 level, it keeps connections open unless the Web client, the server, or the user application in XCICS, specifically requests closure, or the task ends. If XCICS is communicating with a Web client or server at HTTP/1.0 level, it sends Connection: Keep-Alive headers when a persistent connection is supported.

**HTTP functions not supported by XCICS Web support**

The HTTP/1.1 specification defines various roles for the parties that make use of the HTTP protocol. XCICS Web support carries out many of the functions that are appropriate for an origin server, for a client, and for a user agent (although a human user might not be involved for every HTTP client request). The HTTP/1.1 specification also includes requirements that relate to proxies, gateways, tunnels and caches, and these requirements are not relevant to XCICS Web support and can be ignored.

**XCICS does not act as a proxy.**

You can ignore all the requirements in the HTTP/1.1 specification that relate to the behaviour of proxies.

**XCICS does not act as a gateway (an intermediary for another server) or a tunnel (a relay between HTTP connections).**

You can ignore all the requirements in the HTTP/1.1 specification that relate to the behaviour of gateways and tunnels.

**XCICS does not provide caching facilities, or provide support for user-written caching facilities.**

You can ignore all the requirements in the HTTP/1.1 specification that relate to the behaviour of caches. Although you may store any information you receive from a server, you should be careful that you do not deliver the stored information to a user who is making a request in the expectation of receiving current information from the server.

**XCICS is not designed for use as a Web browser.**

Through XCICS as an HTTP client, user application programs can make requests for individual, known resources that are available from a server, but they would not be expected to browse the Internet generally. XCICS does not provide history lists, lists of favorites or other features of a Web browser, so any requirements relating to these can be ignored.
3.2.5.2 Planning the architecture for XCICS as HTTP server

The XCICS Web support architecture for XCICS as an HTTP server varies depending on the tasks you want it to perform. Some configurable components of XCICS Web Support are required for all tasks, such as the TCPIPSERVICE definitions for the ports that receive inbound requests. Other configurable components are only required for specific tasks.

- You can serve dynamic, application-generated responses to HTTP requests, using a specially designed Web-aware XCICS application program. *Providing dynamic HTTP responses with Web-aware application programs* tells you how to do this.

- You can serve static responses to HTTP requests, using documents or files that are available to XCICS. *Providing static HTTP responses with a XCICS document template or flat file* tells you how to do this.

- You can enable a Web client using HTTP to access an existing COMMAREA application in XCICS. *Giving Web clients access to COMMAREA applications* tells you how to do this.

- You can receive non-HTTP requests from a client and provide an application-generated response. *XCICS Web support and non-HTTP requests* tells you how to do this.

### 3.2.5.2.1 Providing dynamic HTTP responses with Web-aware application programs

You can use Web-aware application programs to provide application-generated responses to HTTP requests from a Web client.

The following task-specific components of XCICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- Web-aware application programs, using the EXEC XCICS WEB programming interface
- Alias transactions for the application programs
- Analyzer program
- Security facilities
- Web error programs

*HTTP requests and response processing (as server)* explains how the process elements work together.

1. Design and code one or more Web-aware application programs to provide a response to each request by the Web client, using the information in *Writing Web-aware application programs for XCICS as an HTTP server*. Web-aware application programs can use EXEC CICS WEB and EXEC CICS DOCUMENT commands to receive and analyze an HTTP request, and to write and send a response to the request. Each program handles a single request and response.

   **Note:** Web-aware application programs that use the EXEC XCICS WEB commands must run in the XCICS region where the Web client’s request is received. However, they may link to application programs in other XCICS regions, for example to perform business logic.

2. Consider security issues for this XCICS Web support task. For application-generated responses, XCICS can implement HTTP basic authentication for a connection, where the user must supply an ID and password. (Basic authentication does not operate for items delivered as a static response). You can use the user ID to control access to individual resources. If you need to protect information passed over the Internet (including the user IDs and passwords used for basic authentication), consider using Secure Sockets Layer (SSL). For more information, see *Security for XCICS Web support*.

3. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string.

4. Decide on the port that will be used for the requests. For HTTP, the default port number is 80, and for HTTPS (with SSL), the default port number is 443. Port numbers that are not the default for a scheme need to be specified explicitly in the URL of requests. If you prefer, you can allow a request to use any port that is associated with XCICS Web support.

5. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure in *Creating TCPIPSERVICE definitions* to set up a TCPIPSERVICE definition. The name of the relevant TCPIPSERVICE definition is specified in the URIMAP definition for the request. Specifying no TCPIPSERVICE...
definition means that requests matched by the URIMAP definition can use any port for which a TCPIPSERVICE
definition exists.

6. Select an alias transaction ID for the user application programs that are involved with this task. The default alias
transaction is CWBA. You can create your own alias transaction following the instructions in "Creating TRANSACTION
resource definitions". You can use the URIMAP definition or an analyzer program to specify an alias transaction for
each HTTP request.

7. Decide how the analyzer program associated with the TCPIPSERVICE definition should be used, and select an
appropriate program. "Analyzer programs" has more information about what you can do using an analyzer program.
For Web-aware applications, you can choose between the following strategies:

   a. Use the XCICS-supplied sample analyzer program DFHWEBAHX to provide basic support for both requests using
      URIMAP definitions, and requests following the same process that XCICS Web support used before XCICS TS
      3.1. DFHWEBADX takes no action if a matching URIMAP definition is found. If no match is found, it analyzes
      URLs in the format that was required before CICS TS 3.1. If the analysis fails, DFHWEBADX gives control to the
      Web error program to produce an error response.

   b. Use your own analyzer program to provide customized support. This might include:
      - Making dynamic changes to processing for requests using URIMAP definitions.
      - Providing monitoring or audit actions during processing for requests.
      - Supporting requests following the same process that XCICS Web support used before IBM CICS TS 3.1.
      A customized analyzer program can be specified using the analyzer=eyes attribute in a URIMAP definition, and is
      then involved in the processing path for requests. As supplied, DFHWEBAHX take no action if they are invoked
      from a URIMAP definition.

8. Set up a URIMAP definition to handle each request. Follow the procedures in "Starting a URIMAP resource definition
for any requests for XCICS as an HTTP server" and "Completing a URIMAP definition for an application response
to an HTTP request for XCICS as an HTTP server". It is possible to pass HTTP requests directly to an analyzer
program without using URIMAP definitions, following the same process that XCICS Web support used before IBM CICS
TS 3.1. However, using URIMAP definitions can make it easier to manage HTTP requests. Without URIMAP
definitions, if you want to change the way in which XCICS responds to a particular HTTP request, you need to change
the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system
management task.

3.2.5.2.2 Providing static HTTP responses with a XCICS document template or flat file

You can use a XCICS flat files to provide a static response to an HTTP request from a Web client.

The following task-specific components of XCICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- Files in HFS
- XCICS document template support
- Security facilities
- Web error programs

"HTTP requests and response processing (as server)" explains how the process elements work together.

1. Consider security issues for this XCICS Web support task. For application-generated responses, XCICS can implement
HTTP basic authentication for a connection, where the user must supply an ID and password. (Basic authentication
does not operate for items delivered as a static response). You can use the user ID to control access to individual
resources. If you need to protect information passed over the Internet (including the user IDs and passwords used for
basic authentication), consider using Secure Sockets Layer (SSL). For more information, see "Security for XCICS Web
support".

2. If you want to use a UNIX/Linux flat file to provide a response, create the file and place it in an appropriate location of
your choice in the file system. When this response is identified by a URIMAP definition that matches the Web client's
request, XCICS retrieves the file send it as response.

   a. Do not include any HTTP headers or status line information in the file. XCICS generates the required information
      when the response is sent. The file only provides the body of the response.

   b. The location of the file is significant if you want to use path matching, as described later in this topic. If you do not
want to use path matching, the location of the file does not need to have any relationship to the URL of the request.

c. The XCICS region must have permissions to access to access the directory containing the file, and the file itself.

3. Identify the media type (type of data content) that is provided by the file. You may let XCICS to recognize automatically the media type depending on the file extension.

Note that when you use a URIMAP definition to send a static response, the use of quality factors (the "q" parameter) is not supported. Quality factors can be used to choose among a client's list of acceptable media types or character sets, as specified in Accept headers. If you want to carry out this type of analysis, an application-generated response can be used instead.

4. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string.

5. Decide whether you want to use path matching in the URIMAP definition. If so, plan your request URLs, and arrange the names of your XCICS your files to support this. In path matching, a wildcard character is used in the path component of the URIMAP definition, and also in the name of the file that is specified by the URIMAP definition. The portion of the path that is covered by the wild card character is used to select the file to provide the response.

For files, the portion of the path that is covered by the wild card character is substituted as the last part of the file name. You can store a number of files in the same directory, and access them using request URLs whose paths begin in the same way, through a single URIMAP definition. Bear in mind that because a URIMAP definition must specify a type of data content (the MEDIATYPE attribute), a single URIMAP definition can only handle a group of files that produce the same type of data content.

6. Decide on the port that will be used for the requests.

7. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure in "Creating TCPIPSERVICE definitions for COMMAREA applications".

### 3.2.5.2.3 Giving Web clients access to COMMAREA applications

You can use XCICS Web support to enable Web clients to interact with XCICS applications that use a COMMAREA interface to communicate with other programs. You can write a Web-aware application program that links to the application and uses its output to provide HTTP responses. Alternatively, you can use a converter program to convert the input from the Web client into a suitable COMMAREA, and convert the output from the application into HTML to provide the response.

The following task-specific components of XCICS Web support are used for this task:

- TCPIPSERVICE resource definitions
- URIMAP resource definitions
- COMMAREA application programs
- Either: Web-aware application programs, using the EXEC XCICS WEB programming interface, that link to the COMMAREA application programs and use their output
  - Or: Converter programs that can provide suitable COMMAREA input and convert the output from the application programs into HTML
- Alias transactions to cover the user application programs involved in this process
- Analyzer program
- Security facilities

"HTTP requests and response processing (as server)" explains how the process elements work together.

1. Decide whether you want to write a Web-aware application program that handles the HTTP request and links to the COMMAREA application program; or whether you want to write a converter program to convert the input from the Web client into a suitable COMMAREA, and convert the output from the application into HTML to provide the response.

**Note:** For COMMAREA applications that use a converter program, XCICS cannot provide the fullest possible assistance with structuring responses and with HTTP protocol compliance. To take advantage of all the available XCICS Web support facilities, it is recommended that you use Web-aware applications to handle requests where possible.
a. If you want to use a Web-aware application program, follow the steps in "Providing dynamic HTTP responses with Web-aware application programs". Code your Web-aware application program to link to the COMMAREA application and use its output. The only task that cannot be performed by a Web-aware application program, but can be performed by a converter program, is receiving information that an analyzer program has created to pass to the next processing stage (in a user token or shared work area). This is unlikely to be a consideration when you are developing a new XCICS Web support application.

b. If you want to use a converter program, follow the steps in this topic.

2. Decide how the analyzer program associated with the TCPIPSERVICE definition should be used, and select an appropriate program. "Analyzer programs" has more information about what you can do using an analyzer program. URIMAP definitions or analyzer programs can be used to map requests from Web clients to appropriate converter programs and user-written application programs. For non-Web-aware applications, even if you have URIMAP definitions, you need to use a customized analyzer program in the processing path for the request in the following circumstances:

   a. If you require nonstandard code page conversion.

   b. If you need to communicate any information to a converter program in addition to the standard input. A user token is provided, which the analyzer and converter programs can use to exchange either a small amount of information, or the address of a shared work area containing more information.

   c. If you require monitoring or audit actions, which can be carried out by an analyzer program.

   d. If you need to make dynamic changes to elements of the process such as the converter program that is used, the application program that handles the request, or the alias transaction and user ID used for the request.

3. Use the information in "Converter programs" to create a suitable converter program. XCICS passes the Web client's request to the converter program in a 32K block of storage. (If the request does not fit, it is truncated.) The converter program is called twice, first for the decode function, which receives the request and passes it to the application program, and next for the encode function, which receives the application's output and creates an HTTP response. The tasks carried out by a converter program are:

   **Decode function**
   - Receive the HTTP request in a storage area, along with any additional information supplied by the URIMAP definition or analyzer program in a COMMAREA.
   - Process the HTTP request and convert relevant elements into suitable COMMAREA input for the application program (in the same storage area or a new one).

   **Encode function**
   - Receive the output from the application program.
   - Construct the HTTP response in a COMMAREA.
   - Pass the HTTP response to XCICS for sending to the Web client.

4. Decide on the URL that the Web client will use for each request, including the scheme, host and path components, and any query string.

5. Decide on the port that will be used for the requests.

6. Select an alias transaction ID for the user application programs that are involved with this task. The default alias transaction is CWBA. You can create your own alias transaction following the instructions "Creating TRANSACTION resource definitions". You can use the URIMAP definition or an analyzer program to specify an alias transaction for each HTTP request.

7. Consider security issues for this XCICS Web support task. For application-generated responses, XCICS can implement HTTP basic authentication for a connection, where the user must supply an ID and password. (Basic authentication does not operate for items delivered as a static response.) You can use the user ID to control access to individual resources. If you need to protect information passed over the Internet (including the user IDs and passwords used for basic authentication), consider using Secure Sockets Layer (SSL). For more information, see "Security for XCICS Web support".

8. Set up a URIMAP definition to handle each request. Follow the procedures in "Starting a URIMAP resource definition for any requests for XCICS as an HTTP server" in topic 2.4.3 and "Completing a URIMAP definition for an application response to an HTTP request for XCICS as an HTTP server". It is possible to pass HTTP requests directly to an analyzer program without using URIMAP definitions, following the same process that XCICS Web support used before IBM CICS TS 3.1. However, using URIMAP definitions can make it easier to manage HTTP requests. Without URIMAP definitions, if you want to change the way in which XCICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task.

9. If you do not yet have a TCPIPSERVICE definition for the port on which the requests are received, follow the procedure.
in "Creating TCPIPSERVICE resource definitions" to set up a TCPIPSERVICE definition. The name of the relevant TCPIPSERVICE definition is specified in the URIMAP definition for the request. Specifying no TCPIPSERVICE definition means that requests matched by the URIMAP definition can use any port for which a TCPIPSERVICE definition exists.
3.2.5.3 Writing Web-aware application programs for XCICS as an HTTP server

In XCICS, Web-aware application programs are programs that use EXEC CICS WEB commands to interact with a Web client or a server through XCICS. For XCICS as an HTTP server, these programs can receive and analyze HTTP requests and provide application-generated responses to the Web client.

Before you start to code Web-aware application programs for XCICS as an HTTP server, read the topic HTTP requests and response processing (as server) so that you are aware of the processing stages that can be involved.

For each HTTP request that requires an application-generated response, XCICS calls the Web-aware application program that is specified on the URIMAP definition for the request, or by the analyzer program, if an analyzer is used. If you use a URIMAP definition to specify the application program, you can select a single application program to service all requests using a particular URL. If you are using an analyzer program either instead of, or in addition to, the URIMAP definition, it can carry out analysis on the request and decide on an alternative application program.

Remember: Web-aware application programs that use the EXEC CICS WEB commands must run in the XCICS region where the Web client’s request is received. However, they may link to application programs in other XCICS regions, for example to perform business logic.

For XCICS as an HTTP server, when an application program has sent a response to a request and returned control to XCICS, it does not wait for further requests from the Web client. If you need to share information between different programs (or new instances of the same program) across a series of requests and responses, you can do this using XCICS-managed resources, or using elements of the requests sent by the Web client.

You can code each of your Web-aware application programs to perform some or all of the following actions for processing an HTTP request:

1. Retrieve any information that your application program needs from the request line (including the request URL), using the WEB EXTRACT command. *Examining the request line for an HTTP request* tells you how to do this. The request line includes the HTTP method, which indicates the action that the application program should take. You can also design the path component of a request URL to provide processing information to the application program. If there is a query string in the request URL, the application program can extract it as a whole for processing.

2. Read or browse the HTTP headers for the request, using the HTTP header commands. *Examining the HTTP headers for a message* tells you how to do this. The information in the HTTP headers might be useful to the application program for processing and responding to the request.

3. Retrieve any technical information about the request that your application program needs to use. You can use EXEC CICS commands to access information about the TCP/IP environment and security options. *Retrieving technical and security information about an HTTP request* tells you how to do this.

4. If the request contains form data that you want to extract, read or browse the data using the form field commands. *Examining form data in an HTTP request* in topic 2.3.4 tells you how to do this. The data can be in the body of the request or as a query string in the URL, and XCICS can extract the data from either of these locations.

5. If the request has a message body that you need to use, receive it into a buffer using the RECEIVE command. *Receiving the entity body of an HTTP request* in topic 2.3.5 tells you how to do this. XCICS does not require you to receive a message body if one is present, and some requests do not have a message body.

6. Execute the business logic for the request processing, using the information you have gathered. You might want to involve other application programs to perform processing. A Web-aware application program can produce a response to the HTTP request based on information that it receives from non-Web-aware programs. It is advisable to separate the business logic from the presentation logic. In a Web-aware application, presentation logic controls the interaction with the Web client.

7. Write HTTP headers for the response, using the WRITE HTTPHEADER command. *Writing HTTP headers for a response* tells you how to do this. XCICS automatically provides some required headers, such as the Date header. You can provide additional headers for other purposes.

8. Produce an entity body, or message body, which is the content of the HTTP response. *Producing an entity body for an HTTP message* tells you how to do this. The entity body can be formed from a XCICS document (which is created using the EXEC CICS DOCUMENT application programming interface) or from a buffer of data supplied by the application program.

9. Send the response to the Web client using the SEND command. *Sending an HTTP response from XCICS as an HTTP server* in topic 2.3.8 tells you how to do this. You need to select a suitable status code and reason phrase, and specify the entity body. XCICS assembles the response using these items and the HTTP headers. If you want to use chunked transfer-coding, you also need to follow the special instructions in "Using chunked transfer-coding to send an HTTP request or response".
10. If you expect to exchange further requests and responses with this Web client, and you need to share data across the request sequence, use the suggestions in "Managing application state across an HTTP request sequence" to achieve this.

11. When EXEC CICS WEB commands are used for XCICS as an HTTP server, they do not have the SESSTOKEN option. The SESSTOKEN option indicates that a command is being used for XCICS as an HTTP client.

### 3.2.5.3.1 Examining the request line for an HTTP request

XCICS stores the request line used for each HTTP request, for the application program to access if needed. An application program can use the WEB EXTRACT command to extract components of the request URL (including the path, host name, port number and query string), the method used for the request, or the HTTP version of the request. Non-HTTP requests can also be identified in this way.

The request URL is a major element of the HTTP request line. Your application program might need to examine any of the items in the request line in order to process the request and provide an appropriate response. Some common reasons for extracting information from a request line are:

- Because the same application program is called to handle a number of different requests, perhaps as part of a logical request sequence, or as different requests that relate to the same resource.
- To see what action is being requested from the application by the HTTP method.
- To use the path component of the URL. This identifies the resource to which the request applies. As well as being used to map the request to the handling application, the path component of the URL can be designed to provide processing information to the application. For example, the path component can be used to specify a particular function provided by the application. Or if the Web-aware application is providing a front end for more than one other application, the path component of the URL can identify the application to which the request applies.
- To obtain a query string for processing by the application.
- To identify the HTTP version for the Web client, so that the application can provide an appropriate response. The HTTP version used by the Web client can affect the HTTP headers, status code, and message content for the response. HTTP/1.0 clients might not understand the more advanced features described in the HTTP/1.1 specification.
- To identify a non-HTTP request. "XCICS Web support and non-HTTP requests" has more information about handling non-HTTP requests.

The XCICS Command Reference has full reference information and descriptions of the options available on the WEB EXTRACT command. The WEB EXTRACT command lets you obtain the following items:

- Use the HOST option to obtain the host component of the request URL, as specified either in the Host header field for the request, or in the request line (if the absolute URI form was used for the request).
- Use the HTTPMETHOD option to obtain the HTTP method for the request (for example, GET or PUT).
- Use the HTTPVERSION option to identify the HTTP version, HTTP/1.1 or HTTP/1.0.
- Use the PATH option to obtain the path component of the URL.
- Use the PORTNUMBER option to obtain the port number that applies to the URL. Well-known port numbers for a service are normally omitted from the URL. If the port number is not present in the URL, the WEB EXTRACT command identifies and returns it based on the scheme. For HTTP, the well-known port number is 80, and for HTTPS, the well-known port number is 443.
- Use the QUERYSTRING option to obtain the whole of the query string. The query string is returned in its escaped form, with %xx sequences to represent certain characters that could hinder correct parsing. Alternatively, if the query string includes form data as name and value pairs (for example, account=40138025 ), you can use the WEB READ FORMFIELD command to obtain this data in an unescaped form. "Examining form data in an HTTP request" in topic 2.3.4 tells you how to do this.
- Use the REQUESTTYPE option to identify a non-HTTP request.
3.2.5.3.2 Examining the HTTP headers for a message

Each HTTP header for a request or response message consists of a header name and header value. XCICS stores this information for the application to access if required. An application can receive the value of a specified header, or browse through the names and values of all the headers for a request or response. You can also convert an architected date and time stamp string taken from a header into the ABSTIME format.

Your application might need to examine information in the headers in order to process a request or response, and construct subsequent messages. For example:

- The TE header tells the application whether or not trailing headers are permitted in a chunked response message.
- Conditional headers can provide instructions to the application, such as to reply only if the response document has changed.

Bear in mind that unless you know the exact format of the HTTP request or response, your application should not rely on the presence of any particular header, as Web clients and servers can be inconsistent in the headers they send.

Some HTTP headers contain date and time stamps. XCICS provides the CONVERTTIME command to convert common formats for architected date and time stamp strings into the ABSTIME format, for use by the application.

The standard HTTP headers are described in the HTTP/1.1 specification (RFC 2616) and the HTTP/1.0 specification (RFC 1945). Some HTTP headers are ignored by XCICS, and it is up to the user application to take appropriate action in response to these. Check the HTTP specification to which you are working for detailed guidance and requirements about the meaning and correct use of each HTTP header.

If the message includes any trailing headers, you can read these using the EXEC CICS WEB commands in the same way as for standard headers. The Trailer header on the message specifies the names of all the HTTP headers that were sent as trailing headers.

- To examine the contents of a particular HTTP header, use the WEB READ HTTPHEADER command. Your application program needs to provide a buffer that receives the contents of the header. XCICS returns a NOTFND condition if the header is not present in the request.

- To browse all the headers in a request or response:
  1. Use the WEB STARTBROWSE HTTPHEADER command to begin browsing the header lines.
  2. Use the WEB READNEXT HTTPHEADER command to retrieve the header name and header value for each line. Your application program needs to provide two buffers: one receives the name of the header, and one receives its contents. XCICS returns an ENDFILE condition when all headers have been read.
  3. Use the WEB ENDBROWSE HTTPHEADER command to end the browse when your program has retrieved all the header information of interest.

- To convert an architected date and time stamp string that is provided in a HTTP header, receive it into a buffer using the WEB READ HTTPHEADER command, and then process it using the CONVERTTIME command. You do not need to identify the format of the date and time stamp; the CONVERTTIME command recognizes and converts three different date and time stamp formats which are commonly used on the Internet. These are RFC 1123 format (the Web standard), RFC 850 format (an older format), and Asctime format (output from C function). The ABSTIME can be converted to other formats by the application, using the FORMATTIME command.

3.2.5.3.3 Retrieving technical and security information about an HTTP request

An application can obtain information about the TCP/IP environment for an HTTP request, including the security options that are in use, and about a client certificate that has been provided by a Web client. XCICS manages the TCP/IP connection between a Web client and server, applies appropriate security measures, and manages the process of authenticating a Web client’s identity. The actions taken by XCICS for each connection are determined by the options you set in the TCPIPSERVICE definition for the port on which the Web client’s request is received. A user-written application can examine information obtained by this process, if this is useful for determining how to process the request. For example, you can obtain the host name and IP address of the Web client that sent the HTTP request, or check the level of security and encryption for the connection.

The EXTRACT TCP command provides information about the TCP/IP connection, and about security options specified in the TCPIPSERVICE definition. The EXTRACT CERTIFICATE command provides information taken from any X.509 client certificate that was received from the Web client during a Secure Sockets Layer (SSL) handshake.
3.2.5.3.4 Examining form data in an HTTP request

Form data is information provided by the user through interaction with an element in a HTML form (such as a text input box, button, or check box). The information is transmitted as a series of name and value pairs. XCICS can scan an HTTP request to pick out the form fields, so an application can obtain the data using XCICS commands, without needing to receive and analyze the entire body of the request.

An application can receive the value of a specified form field, or browse through the names and values of all the form fields contained in a request. You can specify code page conversion options if you need to convert the data into a different code page for use by your application.

The Web client sends form data in a query string when the GET method is used, and in the message body when the POST method is used. XCICS can extract the data from either of these locations, so you do not need to specify which method was used. As an alternative, if the form data is sent in the query string, you can retrieve the entire query string using the WEB EXTRACT command. "Examining the request line for an HTTP request" tells you how to do that.

XCICS only reads form data when XCICS is the HTTP server. The facility is not available when XCICS is an HTTP client.

- To obtain the value of a particular field of an HTML form, use the WEB READ FORMFIELD command. Your application program can provide a buffer which will receive the value, or alternatively, you can provide a pointer which XCICS sets to the address of the value. XCICS returns a NOTFND condition if the form data does not contain a field with the specified name. The form data is unescaped by XCICS before it is returned, with the %xx sequences converted back to the original characters.

- To browse all the fields in the form data:
  1. Use the WEB START BROWSE FORMFIELD command to begin browsing the fields.
  2. Use the WEB READ NEXT FORMFIELD command to retrieve the name and value of each field in turn. Your application program needs to provide two buffers: one receives the name of the field, and one receives its contents. XCICS returns an ENDFILE condition when all fields have been read.
  3. Use the WEB END BROWSE FORMFIELD command to end the browse when your program has retrieved all the fields of interest.

3.2.5.3.5 Receiving the entity body of an HTTP request

An application can issue the WEB RECEIVE command to receive the entity body of an HTTP request. You can receive only the first part of the entity body, or use a series of WEB RECEIVE commands to receive the whole body in smaller sections.

The WEB RECEIVE command does not set a timeout value. The user application is only called when the complete request has been successfully received from the Web client, and is being held by XCICS. For XCICS as an HTTP server, the close_timeout attribute in the TCPIP SERVICE definition for the port determines how long the Web client has to complete its request send. When this period expires, XCICS returns a 408 (Request Timeout) response to the Web client.

If a request message is sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to the application. If a series of pipelined requests is sent, XCICS treats each request as a separate transaction, and requires a response from the user application before making the next request available to the next user application for processing.

The XCICS Command Reference has full reference information and descriptions of the options available on the WEB RECEIVE command. When you issue the WEB RECEIVE command:

1. Identify whether or not you need to receive an entity body for this request.
   - For certain request methods (such as the GET method), an entity body is not appropriate, and your application is allowed to ignore any entity body that is present. If an inappropriate entity body is present, you may still receive it if you want. "Examining the request line for an HTTP request" in topic 2.3.1 tells you how to identify the request method.
   - For an HTTP/1.1 request, the presence of an entity body is indicated by a non-zero Content-Length header on the request (or a Transfer-Encoding header, if the message is chunked). If the value of the Content-Length header is zero, or if neither the Transfer-Encoding header nor the Content-Length header is supplied, there is no entity body. "Examining the HTTP headers for a message" tells you how to read the HTTP headers for the message.
   - HTTP/1.0 requests are not required to specify a Content-Length header, but they might do so. If you find a
non-zero Content-Length header on the request, this indicates the presence of an entity body. If there is no Content-Length header, but the request method (in particular, the POST method) indicates that an entity body is appropriate, it is likely that an entity body is present.

2. Receive the entity body by specifying either the INTO option (for a data buffer), or the SET option (for a pointer reference), and the LENGTH option. On return, the LENGTH option is set to the length of data received.

3. If you want to limit the amount of data received from the entity body, specify the MAXLENGTH option.
   - If you want to receive only the first part of the entity body, and discard any data that exceeds this length, omit the NOTRUNCATE option. This is the default.
   - If you want to retain, rather than discarding, any data that exceeds this length, specify the NOTRUNCATE option. Any remaining data can be obtained using further WEB RECEIVE commands.
   - If the data has been sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the entity body for the chunked message, rather than to each individual chunk. The total amount of data that XCICS accepts for a single message is limited by the MAXDATALEN attribute of the TCPIPSERVICE definition.

4. If you specified MAXLENGTH and NOTRUNCATE, and you have more data to receive, issue further WEB RECEIVE commands. A single RECEIVE command using the SET option and without the MAXLENGTH option receives all the remaining data, whatever its length. Alternatively, you can use a series of RECEIVE commands with the NOTRUNCATE option to receive the remaining data in appropriate chunks. Keep issuing the RECEIVE command until you are no longer getting a LENGERR response. Bear in mind that if you receive less than the length requested on the MAXLENGTH option, this does not necessarily indicate the end of the data; this could happen if XCICS needs to avoid returning a partial character at the end of the data.

### 3.2.5.3.6 Writing HTTP headers for a response

For dynamic responses created by application programs, XCICS automatically provides the HTTP headers that are required for basic messages, depending on the HTTP protocol version used for the message. You might need to add further HTTP headers to your response.

Some HTTP headers are created automatically by XCICS if the message requires them. Your application does not need to write these headers. The full list of headers created by XCICS is:

- Connection
- Content-Length
- Content-Type
- Date
- Host
- Server
- Transfer-Encoding
- User-Agent
- WWW-Authenticate

The headers that XCICS provides when a response is sent are the ones that should normally be written for a basic message to be compliant with the appropriate HTTP protocol specification. You might want to add further HTTP headers to the response for purposes such as:

- Control of caching and document expiry (for example, Cache-Control, Expires, Last-Modified)
- Content negotiation (for example, Accept-Ranges, Vary)
- Information for the Web client (for example, Title, Warning, further Content headers)

If your application program is performing complex actions, or if you select certain status codes for your response, the HTTP specification to which you are working is likely to require the use of particular HTTP headers for your message. When you add any HTTP headers to a response, check the HTTP specification to which you are working for any important requirements that apply to those headers.
Write additional HTTP headers for a message before you issue the WEB SEND command to send the message. The exception to this rule is if you are writing headers to be sent as trailing headers on a chunked message, in which case the special process mentioned below applies.

When writing HTTP headers for a response:

- Use the WEB WRITE HTTPHEADER command for each header that you want to add to the message. Make sure that you specify the name and value for each header in the format described by the HTTP specification to which you are working (XCICS does not validate the content of HTTP headers). The command adds a single header, and you can repeat the command to add further headers. If you write a header that you have already written, XCICS overwrites the previous header to the request or response in addition to the existing header.

- If you want to produce a date and time stamp for use in one of your HTTP headers (for example, the Last-Modified header), you can use the FORMATTIME command. The command converts the current date and time (in ABSTIME format), or a date and time produced by the application program, into the RFC 1123 format. The RFC 1123 format is the only date and time stamp format that XCICS creates for use on the Web. Other date and time stamp formats might not be accepted by some Web clients or servers with which XCICS is communicating.

- If you are using chunked transfer-coding to send an HTTP request or response, and you want to include trailing headers at the end of the chunked message, follow the special instructions in “Using chunked transfer-coding to send an HTTP request or response.” You need to write a Trailer header before sending the first chunk of the message. All the HTTP headers written after the WEB SEND command for the first chunk are treated as trailing headers.

- Make sure that your application program carries out any actions that are implied by your user-written headers. For example, if you have written content-negotiation headers, your application program needs to provide different versions of the resource.

3.2.5.3.7 Producing an entity body for an HTTP message

Web-aware application programs can produce an entity body formed from a XCICS document, or from a buffer of data. XCICS documents can be used as the entity body of an HTTP message. They are created using the EXEC CICS DOCUMENT commands. They can be populated by data specified directly by the application program, and by document templates, which are portions of documents defined as XCICS resources or created by another XCICS program. Documents and document templates can be stored for reuse.

Alternatively, you can specify a buffer of data created by the application program. You might find this option more convenient for short or simple entity bodies, and it is the only option that enables you to use chunked transfer-coding for the message. However, data created in this way cannot be stored for reuse so easily.

1. To create a XCICS document, follow the instructions in the XCICS Document commands guide. The document is created using the EXEC CICS DOCUMENT application programming interface (EXEC CICS DOCUMENT CREATE, INSERT, and SET commands). The DOCTOKEN option on the WEB SEND command is used to specify the document token for the finished document. XCICS retrieves the document and performs appropriate code page conversion, depending on the options you specify on the WEB SEND command. The body of a chunked message cannot be formed from XCICS documents.

2. Alternatively, assemble a message body within your application program. The FROM option on the WEB SEND command is used to specify the buffer of data.

3.2.5.3.8 Sending an HTTP response from CICS as an HTTP server

Use the WEB SEND command to make XCICS assemble the HTTP headers, entity body, status code and reason phrase for an HTTP response, carry out code page conversion, and transmit the response to the Web client.

Write any additional HTTP headers for the response using the WEB WRITE HTTPHEADER command before issuing the WEB SEND command, as described in “Writing HTTP headers for a response.” Also produce any entity body that is needed for the message, as described in “Producing an entity body for an HTTP message.”

You need to specify a status code and reason phrase on the WEB SEND command.

If wanted, the response can be sent in chunks (chunked transfer-coding).

When you issue the WEB SEND command:
1. Specify the STATUSCODE option to select an appropriate status code for the response, depending on the situation, and the STATUSTEXT and STATUSLEN options to provide the reason phrase. XCICS does not validate your choice of status code, and it is the user application's responsibility to ensure that the value is valid and conforms to the rules for HTTP status codes.

2. Identify the source of any entity body for the response by specifying either the DOCTOKEN option, for a XCICS document that you have created, or the FROM option, for a body of data that you have assembled. If you are using the FROM option, specify the FROMLENGTH option to give the length of the entity body, or of the chunk if chunked transfer-coding is in use. For chunked transfer-coding, the DOCTOKEN option cannot be used.

3. Specify the media type for the body of the response, using the MEDIATYPE option. XCICS does not check the validity of the specification against the data content. There is no default. If you do not specify this, XCICS does not build a Content-Type header for the response.

4. If you want the message to be sent immediately, rather than at the end of the task (which is the default), specify IMMEDIATE for the ACTION option. If you are using chunked transfer-coding, IMMEDIATE is the default, so there is no need to make this choice.

Note: Only one response can be sent during a task. This can be a standard response using one WEB SEND command, or a chunked response using a sequence of WEB SEND commands.

5. If you want to close the connection after sending the response, specify CLOSE for the CONNECTION option. XCICS writes a Connection: close header on the response, which notifies the Web client that the connection is closed and no more requests should be sent. (For a Web client at HTTP/1.0 level, XCICS achieves the same effect by omitting the Connection: Keep-Alive header.)

6. If you are using chunked transfer-coding (or chunking), in addition to the basic instructions in this topic, you need to follow the special instructions in "Using chunked transfer-coding to send an HTTP request or response". You need to ensure that the procedure described in that topic is followed correctly, so that the chunked message is acceptable to the recipient. Chunked messages are sent using several instances of the WEB SEND command, with particular options.

3.2.5.3.9 Using chunked transfer-coding to send an HTTP request or response

Before setting up chunked transfer-coding, you need to plan the following attributes of the item that you want to send:

1. The HTTP headers that should be used at the beginning of the message. XCICS supplies its usual message headers and, for a chunked message, the proper headers for chunked transfer-coding, including the Transfer-Encoding: chunked header. If any additional headers are required at the beginning of the message, the application can write them before the first WEB SEND command.

2. Any headers that should be sent in the trailer at the end of the message. These headers are known as trailing headers.

3. How the message should be divided up. This can be done in whatever way is most convenient for the application program. For example, the output from a number of other application programs could be sent as it is produced, or data from each row of a table could be read and sent individually.

4. The length of each chunk of data that will be sent. Do not include the length of any trailing headers.

The procedure described in this topic enables you to create a correctly constructed chunked message, as defined in the HTTP/1.1 specification. If the chunked message is not correctly constructed, the recipient may discard it.

The body of a chunked message cannot be formed directly from XCICS documents (so the DOCTOKEN option cannot be used). The FROM option must be used to specify data to form the body of a chunked message.

When you have begun sending the parts of a chunked message, you cannot send any different messages or receive any items, until the final empty chunk is sent and the chunked message is complete.

1. Before beginning a chunked message, verify that the Web client or server is at HTTP/1.1 version. All HTTP/1.1 applications are required to understand chunked transfer-coding. A chunked message cannot be sent to an HTTP/1.0 recipient.

- For responses sent by XCICS as an HTTP server, use the WEB EXTRACT command to check the HTTP version specified for the Web client's request.

- For requests sent by XCICS as an HTTP client, the HTTP version of the server is returned on the WEB
OPEN command for the connection.

2. Use the WRITE HTTPHEADER command as many times as necessary to write any HTTP headers that should be sent before the body of the message. Do not write the headers for chunked transfer-coding; XCICS writes these itself, using the chunk length information supplied by the application program.

3. If you want to include trailing headers (headers sent out after the body of the message) with the chunked message, use the WRITE HTTPHEADER command to write a Trailer header. Specify the names of all the HTTP headers you plan to send in the trailer, as the value of the Trailer header. You may send any headers as trailing headers, except the Transfer-Encoding, Trailer and Content-Length headers.

4. Use the WEB SEND command to send the first chunk of the message.
   - Specify CHUNKING(CHUNKYES) to tell XCICS that this is a chunk of a message.
   - Use the FROM option to specify the first chunk of data from the body of the message.
   - Use the FROMLENGTH option to specify the length of the chunk.
   - For requests by XCICS as an HTTP client, an appropriate method must be specified on the METHOD option. Chunked transfer-coding is not relevant for requests with no message body, so it is not relevant for the GET, HEAD, DELETE, OPTIONS, and TRACE methods, but it can be used for the POST and PUT methods.
   - Specify any other options that apply to both chunked and non-chunked messages, as given in your main set of instructions. For example, if this chunked message is the final message that you want to send to this server or Web client, specify the CLOSESTATUS(CLOSE) option.

5. Use the WEB SEND command as many times as necessary to send each of the remaining chunks of the message. On each WEB SEND command, just specify the following items:
   - CHUNKING(CHUNKYES).
   - The FROM option, giving the chunk of data.
   - The FROMLENGTH option, giving the length of the chunk.

6. When you have sent the last chunk of the data, specify a further WEB SEND command with CHUNKING(CHUNKEND) and no FROM or FROMLENGTH option. XCICS then generates and sends an empty chunk to the recipient to end the chunked message. The empty chunk is sent along with the trailer containing any trailing headers that you wrote.

7. For XCICS as an HTTP server, errors are handled as follows:
   - If one of the WEB SEND commands fails during the sequence, an error response is returned, and subsequent sends will also fail. The application should handle this situation appropriately.
   - If all of the chunks are sent successfully but the application does not issue the final WEB SEND command with CHUNKING(CHUNKEND), XCICS completes the request issuing the ending chunk.

8. For XCICS as an HTTP client, errors are handled as follows:
   - If your application program is informed of an error at any point in the chunked transfer-coding process, use the WEB CLOSE command to stop the process and close the connection. The server will not receive the final empty chunk, and so should ignore and discard the data that you have sent so far. You can decide whether or not to retry the request.
   - If you do not send the final empty chunk or issue the WEB CLOSE command, XCICS will do it.

3.2.5.3.10 Managing application state across an HTTP request sequence

XCICS initiates a new alias transaction and a new program for each request made by a Web client. You need to consider how the application's state will be managed between requests. If you need to share data across the request sequence, between different programs or instances of the same program, you can do this using XCICS-managed resources, or using elements of the requests sent by the Web client.
When more than one exchange of a request and response between a Web client and XCICS is needed to complete a task successfully, each new step in the sequence is initiated by the Web client. You can design the response sent by XCICS to guide the Web client, and any human user of the Web client, to the next step. For example, the entity body can contain controls (such as links or buttons) that the end user can use to compose the next request. However, you cannot easily enforce the correct sequence of requests. In particular, the planned sequence can be disrupted if:

- The client is a Web browser, and the end user types a known URL to initiate a particular request, rather than selecting a control in an HTML page provided by a previous response.
- The end user abandons the activity altogether, by shutting down the Web client or by changing to some alternative activity with the Web client.

The end user might also delay initiation of any request in the sequence.

You should design your application programs so that they can cope with delays or disruptions in the request sequence. For example, if you are sharing data across the request sequence, you should ensure the data is cleaned up if the request sequence does not complete or is delayed excessively. If your application programs update protected resources, you should ensure that updates that must be committed or backed out together are made in the same transaction. (This means that a single request from the Web client should be designed to complete the update.)

The ideal situation for an application is that each exchange of a request and response is self-contained and completes an independent element of the task. However, this design is not always possible, especially when the task is complex, or when a Web client has sent a pipelined sequence of requests. A pseudo-conversational model may be required, where the application's state must be managed between requests. This can be arranged using the following techniques:

- You can design the requests sent by the Web client so that application state, or shared data, is incorporated in the request, for example as part of a request URL that is used when the Web client submits an HTML form. The next program can examine the request URL to obtain the shared data.
- You can store small quantities of application state using hidden fields in an HTML form that is returned to the Web client as a response. When the user performs the next action in the planned sequence, the request that they send to XCICS can include the hidden fields, which can be located and read by the next application program.
- For larger quantities of state, and state with an extended lifetime, you can create a XCICS-managed resource to maintain the application's state, and pass a token that represents the resource. This token can be conveyed from program to program in a pseudo-conversation as a hidden field in an HTML form, or from interaction to interaction as a query string in a URL. This technique can be used for preserving information throughout a pseudo-conversation, and also for preserving information throughout an extended interaction between an end user and various XCICS application programs, perhaps over several pseudo-conversations.
3.2.5.4 Resource definition for XCICS Web support

You need to create some additional resource definitions for each XCICS Web support task that you want to perform. "Planning the architecture for XCICS as HTTP server" has planning guidance that specifies the resource definitions which you need for each task. The resource definitions that you might need to set up are as follows:

- Create a TCPIPSERVICE resource definition to define each port that you use to receive inbound HTTP requests for XCICS Web support. This resource definition is the place where you specify the security measures that are applied for each port, along with technical information on the operation of the port. "Creating TCPIPSERVICE resource definitions" tells you how to do this.

- Optional: Create TRANSACTION resource definitions for any alias transactions that you want to use for inbound HTTP request processing. "Creating TRANSACTION resource definitions" tells you how to do this.

- Create a URIMAP resource definition to provide processing information for each HTTP request for XCICS as an HTTP server, and for each HTTP request made from XCICS as an HTTP client.

  1. For all HTTP requests to XCICS as an HTTP server, start the definition following the steps in "Starting a URIMAP resource definition for any requests for XCICS as an HTTP server".

  2. For HTTP requests where an application-generated response is provided, follow the steps in "Completing a URIMAP definition for an application response to an HTTP request for XCICS as an HTTP server".

  1. For HTTP requests where a static response is provided, follow the steps in "Completing a URIMAP definition for a static response to an HTTP request for XCICS as an HTTP server".

  1. For HTTP requests made from XCICS as an HTTP client, follow the steps in "Creating a URIMAP definition for an HTTP request by XCICS as an HTTP client".

- If you need to create URIMAP resource definitions for use by XCICS as an HTTP client, follow the steps in "Creating a URIMAP definition for an HTTP request by XCICS as an HTTP client".

3.2.5.4.1 Creating TCPIPSERVICE resource definitions

TCPIPSERVICE resource definitions are used to define the association between ports and XCICS services, including XCICS Web support. Define and install a TCPIPSERVICE resource definition for each port that you use for XCICS Web support.

Each TCPIPSERVICE definition that is active in a XCICS system must specify a unique port number. XCICS uses the TCPIPSERVICE definition for a port to determine what XCICS service should be invoked when it receives an inbound TCP/IP connection request on that port. The PROTOCOL attribute is used to identify the service. HTTP is specified for standard XCICS Web support, and USER is specified for non-HTTP requests that are handled using XCICS Web support.

For XCICS Web support, you normally need to create TCPIPSERVICE definitions for the default, or well-known, port numbers that are used for Internet services. For HTTP, the default port number is 80, and for HTTPS, the default port number is 443. You can also use non-standard port numbers.

Each TCPIPSERVICE definition can only specify one analyzer program, and one transaction definition for the Web attach task. If you need to use more than one of these items, you will need to use different TCPIPSERVICE definitions, and therefore different ports.

Note: The TCPIPSERVICE resource definition is the place where you specify the security measures that are applied for each port. You can choose whether or not to use SSL, and if you do use SSL, you need to choose the exact security measures that are applied (for example, the authentication method, the sending of certificates by client and server, and encryption of messages). Read "Security for XCICS Web support" in topic 2.9 for more information about the security features that you can use to keep your XCICS Web support facility safe.

The XCICS System Management Guide contains information about TCPIPSERVICE resource definition attributes that you will use during this process.

  1. Identify a TCP/IP port that you want to use for XCICS Web support. You are recommended to reserve the port number for use by XCICS Web support.

  2. Begin TCPIPSERVICE definition with the statement "add tcpipservice" in xcics.conf

  3. Specify the "port" attribute as the number of the TCP/IP port that is covered by this definition.

  4. Use the "protocol" attribute to specify that XCICS Web support handles requests on this port.
• Specify "http" for normal HTTP requests. This option covers both HTTP with SSL and HTTP without SSL. The "ssl" attribute specifies whether or not SSL is involved.

• Specify "user" for non-HTTP requests that are handled using XCICS Web support. When "user" is specified, XCICS Web support facilities are used for handling the request, but no acceptance checks are carried out for messages sent and received using this protocol. The requests are flagged as non-HTTP and passed straight to the analyzer program. URIMAP definitions are not used for these requests.

5. Specify the "urn" attribute as the name of the analyzer program that is associated with this TCPIPSERVICE definition. For a non-HTTP (USER protocol) request, the analyzer program is always used. For an HTTP request, the analyzer program is used to interpret the request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present. An analyzer program must be specified. Only one analyzer program can be selected for each TCPIPSERVICE definition, but you can code it to handle any requests.

6. Use the "close_socket" and the "close_timeout" attributes to specify if and how long XCICS should wait before closing the socket, after issuing a receive for incoming data on that socket. The "close_socket" attribute set to "no" means that persistent connection will be used. Persistent connection are closed when the timeout specified with the "close_timeout" attribute is reached, or when the Web client closes it.

7. Use the "ssl" attribute to specify whether or not the secure sockets layer (SSL) is used for this port. YES means that SSL is used, and XCICS sends a server certificate to the Web client.

8. If you have specified ssl=yes, make sure that the general settings server_ssl_certificate and server_ssl_key are correctly defined for the region.

9. Use the "auth" attribute to specify the level of authentication that is used for Web clients making requests on this port.

• Specify "none" if the Web client is not required to send authentication or identification information.

• Specify "basic" to make XCICS attempt HTTP basic authentication, where XCICS requests a user ID and password from the Web client.

• Specify "certificate" to use SSL client certificate authentication. The Web client must send a valid certificate which is already registered to the security manager, and associated with a user ID. If a valid certificate is not received, or the certificate is not associated with a user ID, the connection is rejected.

• Specify "automatic" to either use SSL client certificate authentication, or if no certificate is sent, to use HTTP basic authentication (as for the "basic" option).

### 32.5.4.2 Creating TRANSACTION resource definitions

TRANSACTION resource definitions are used to define alias transactions for XCICS Web support. An alias transaction handles the later stages of processing for an HTTP request, including receiving the request, executing the application’s business logic, construction of the HTTP response and code page conversion of the HTTP response. Alias transactions can also be used for processing non-HTTP requests.

XCICS supplies a resource definition for a default alias transaction, CWBA. You may want to use alternative alias transaction names for the purposes of:

• Auditing, monitoring or accounting

• Resource and command checking for security

• Allocating initiation priorities

• Allocating DB2® resources

• Assigning different runaway values to different XCICS application programs

• Transaction class limitation

You can set up as many alias transaction definitions as you want. You can use the URIMAP definition or an analyzer program to specify the alias transaction that is required for a particular request.
Your alias transaction definition must use the XCICS-supplied alias program "dfhweb". The alias program calls the user application program that you have specified to process the request.

Your alias transaction definition must be a local transaction.

**Example**

```
define transaction name=MWEB, program=dfhweb, twa=512, protection=1;
```

### 3.2.5.4.3 Starting a URIMAP resource definition for any requests for XCICS as HTTP server

URIMAP resource definitions are used to define how HTTP requests are processed. For any HTTP request for XCICS as an HTTP server, start the URIMAP definition by specifying the components of the expected URL for the Web client's request (scheme, host and path) and other basic information.

The XCICS System Management Guide contains information about URIMAP resource definition attributes that you will use during this process.

1. Identify the URL that you plan to receive as an HTTP request from a Web client. The URL represents a resource that you plan to make available to a Web client through XCICS.

2. Divide the URL for the HTTP request into its scheme, host and path components. If you want the URIMAP definition to match more than one path, you can use an asterisk as a wildcard character at the end of the path. For example, specifying the path "xcics/web/cics/*" would make the URIMAP definition match all requests whose path begins with the string "xcics/web/cics/". If more than one wildcarded URIMAP definition matches an HTTP request, the most specific match is taken.

3. If a query component is present in the URL, and you want to match the URIMAP definition to that specific query alone, you can include this as part of the PATH specification. A query string can be used after a path that includes an asterisk as a wildcard, but the query string cannot itself include an asterisk as a wildcard. The complete and exact query string must be specified. For a static response with a XCICS document template, a query string can either be used to select the URIMAP definition, or it can be substituted into the document template, but not both. If you do not include a query string in the URIMAP definition, matching takes place only on the path, and any query string that is present in the request is automatically ignored for matching purposes.

4. Begin a URIMAP definition with a name.

5. Use the "enabled" attribute to specify whether the URIMAP definition should be in an enabled or disabled state.

6. Specify a "usage" attribute of "server" (XCICS as an HTTP server).

7. Specify the "scheme" attribute as the scheme component of the URL for the HTTP request: "http" or "https".

8. Specify the "tcpipservice" attribute as the name of the TCPIPIService definition that defines the inbound port to which this URIMAP definition relates. If this attribute is not specified, the URIMAP definition applies to a matching HTTP request on any inbound port.

9. If you need to distinguish between URLs containing different host names, specify the "host" attribute as the host component of the URL for the HTTP request. Do not include a port number. An IPv4 address can be used as a host name, but IPv6 addresses are not supported. If you specify a single asterisk as the host attribute, the URIMAP definition matches any host name on incoming URLs. Use this option if you are not using multiple host names, or if you do not want to distinguish them.

10. Specify the "path" attribute as the path component of the URL for the HTTP request, including an asterisk as a wildcard character if required. You can either include or omit the delimiter / (forward slash) at the beginning of the path component; if you omit it, XCICS automatically provides it. If a query component is present, and you want to apply the URIMAP definition to that specific query alone, include this as part of the path component (including the question mark at the beginning of the string).

   **Note:** Specifying the path attribute as / makes the URIMAP definition match all requests directed to the host named in the host attribute, unless more specific URIMAP definitions are also installed, which case the most specific match is taken.

11. Now complete your definition:

   - For an application-generated response, follow the instructions in "Completing a URIMAP definition for an application response to an HTTP request for XCICS as an HTTP server".
For a static response, follow the instructions in "Completing a URIMAP definition for a static response to an HTTP request for XCICS as an HTTP server".

### 3.2.5.4.4 Completing a URIMAP definition for an application response to an HTTP request for XCICS as an HTTP server

If you are providing an application-generated response to an HTTP request, when you have started the URIMAP definition by specifying the components of the expected URL (scheme, host and path) and other basic information, complete the definition by providing information about the application or applications that should process the request and supply an HTTP response.

When you have planned your application-generated response and started your URIMAP definition, complete the definition following the instructions in this topic. The XCICS System Management Guide contains information about URIMAP resource definition attributes that you will use during this process.

1. Specify a "type" attribute of "application".

2. If the analyzer program associated with the TCPIPSERVICE definition (or definitions) to which this URIMAP definition relates is to be involved in the processing path for this request, specify "yes" for the "analyzer" attribute to activate it. If an analyzer program is used, you can still specify the "converter", "transaction", "userid" and "program" attributes. The values that you specify for these attributes are used as input to the analyzer program, but they can be overridden by it. Alternatively, you can leave these attributes blank and let the analyzer program specify them.

3. If you are using a converter program, specify the "converter" attribute as the name of the program. The program can be any converter program that is available in XCICS; there is no association between the converter program and the TCPIPSERVICE definition, as there is for the analyzer program. If a converter program is used, you can still specify the "program" attribute. The value that you specify for this attribute is used as input to the converter program. The converter program can change the "program" attribute to specify a different application program to process the request.

4. Specify the "transaction" attribute as the name of an alias transaction that is available in XCICS, which XCICS can use to run the application program that provides the response. The default alias transaction is CWBA. The transaction name can also be changed or provided by an analyzer program, if "analyzer=yes" is specified.

5. Specify the "userid" attribute as the user ID under which the alias transaction is attached. A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained from the Web client (specified by the AUTHENTICATE attribute of the TCPIPSERVICE definition for the connection). If "analyzer=yes" is specified, the analyzer program can change either of these user IDs, or provide one. If no user ID is specified by any of these means, the default user ID is the XCICS default user.

6. Specify the "program" attribute as the name of the application program that provides the response to the request. If no analyzer or converter program is specified in the URIMAP definition, the HTTP request is passed directly to this application program. If an analyzer or converter program is specified, you can leave this attribute blank and let the analyzer or converter program specify it.

### Example

```plaintext
define urimap name=URIA,
   usage=server,
   type=application,
   scheme=http,
   analyzer=no,
   tcpipservice=HTTPD,
   host="*",
   program=PMX1YURI,
   path="/prova/ax1";

define urimap name=URIB,
   usage=server,
   type=application,
   analyzer=yes,
   tcpipservice=HTTPD,
   host="*",
   program=PMX1YURI,
   path="/prova/ad";
```
3.2.5.4.5 Completing a URIMAP definition for a static response to an HTTP request for XCICS as an HTTP server

For a static response, when you have started the URIMAP definition by specifying the components of the expected URL (scheme, host and path) and other basic information, complete the definition by providing the information that XCICS needs to construct the static response to the request, using a resource file. You can use a wildcard for path matching, where XCICS takes the portion of each HTTP request's path that is covered by the wildcard character, and substitutes this as the last part of the file path.

When you have planned your static response and started your URIMAP definition, complete the definition following the instructions in this topic. The XCICS System Management Guide contains information about URIMAP resource definition attributes that you will use during this process.

**Note**: When you deliver items as a static response, basic authentication does not operate. Resource level security cannot be applied to these items. If you need to apply access controls based on a user ID to an item delivered in this way, you need to deliver the material as an application-generated response instead.

1. Specify a "type" attribute of "static".
2. Specify the "media_type" attribute as the data content of the static response that XCICS provides. A media_type attribute of "*" (asterisk) means the XCICS will try to determine the media type according to the file extension.
3. Specify the "file" attribute as the fully qualified (absolute) name of the file. The file can be specified as an absolute, or fully qualified, path that begins with a slash, or as a relative path that does not begin with a slash. The XCICS region must have permission to access the directory containing the file, and the file itself. If you want to use path matching, include an asterisk as a wildcard character at the end of the path for the file, and also at the end of the path specified by the "path" attribute. XCICS takes the portion of each HTTP request's path that is covered by the wildcard character, and substitutes this as the last part of the file path.

**Example**

```define urimap name=URL, type=static, usage=server, host="*", path="/home/*", file="$HOME/regtest/web/*";```
3.2.5.5 Analyzer programs

Analyzer programs are associated with TCPIPSERVICE definitions. Their primary role is to interpret an HTTP request if a URIMAP definition specifies the use of an analyzer program, or if no URIMAP definition is present.

Relationship between analyzer programs and URIMAP definitions

URIMAPs definitions are the strategic facility to control the processing of HTTP requests. They replace key functions of the analyzer program in matching the URLs of requests to the application program that processes them, and specifying the use of a converter program and an alias transaction.

URIMAP definitions may, however, invoke an analyzer program for selected HTTP requests, to take over some of the processing stages, and to perform other actions such as monitoring or audit actions. The attributes of the URIMAP definition that reproduce analyzer functions, namely "converter" (converter program name), "transaction" (alias transaction), "userid" (user ID for alias transaction), and "program" (name of application program to process request), can be passed to the analyzer program, and the analyzer program can choose to override these.

You can choose to pass an HTTP request directly to an analyzer program without using a URIMAP definition, following the same process used by IBM CICS Transaction Server Version 2. However, without URIMAP definitions, if you want to change the way in which XCICS responds to a particular HTTP request, you need to change the logic in the analyzer program. With URIMAP definitions, you can perform these changes dynamically as a system management task.

Note: As supplied, the XCICS-supplied default analyzer program DFHWBAHX does not perform any analysis of a request when a matching URIMAP definition has been found for the request, even if the URIMAP specifies "analyzer=yes".

Use of analyzer programs for error handling

Although an analyzer program is not now required in the processing path for every HTTP request, an analyzer program must still be specified for each TCPIPSERVICE resource definition that is used for XCICS Web support.

The name of the analyzer program is specified in the URM attribute of the resource definition. You can specify a different analyzer in each TCPIPSERVICE definition, or you can specify the same analyzer in more than one TCPIPSERVICE definition. If you are invoking an analyzer program from a URIMAP definition, you cannot choose between different analyzer programs; you can only select whether or not to use the analyzer program specified for the TCPIPSERVICE definition.

The analyzer program specified for a TCPIPSERVICE definition is invoked to handle an HTTP request if XCICS does not find a matching URIMAP definition for the request. This could be caused by a user error in typing a request URL, or because the appropriate URIMAP definition is not installed.

The XCICS-supplied default analyzer program DFHWBAHX is the default.

Use of analyzer programs for some non-Web-aware applications, and for non-HTTP messages

Non-Web-aware applications might function correctly when they are invoked directly from a URIMAP definition. However, because these applications do not use the EXEC CICS WEB API commands, some might be dependent on facilities that can only be provided for them by an analyzer program.

For non-HTTP requests, which use the user-defined (USER) protocol on the TCPIPSERVICE definition, an analyzer program is always required to process the requests, and URIMAP definitions cannot be used.

Use of analyzer programs for additional processing

In situations where the use of an analyzer program in the processing path is optional, you might choose to use an analyzer program for reasons such as the following:

- You want to make dynamic changes to elements of the processing path, based on the content of the request. Each URL for a HTTP request is matched by a single URIMAP definition, which defines a single processing path. An analyzer program can interpret the content of the request and change elements such as the application program that handles the request, the involvement of a converter program, or the alias transaction and user ID used for the request.

- You want to introduce monitoring or audit actions into the process. An analyzer program is an appropriate location in which to do this.
3.2.5.5.1 Writing an analyzer program

You can write an analyzer program in C, COBOL, or PL/I.

Input and output parameters for an analyzer program are passed in a COMMAREA. Language-dependent header files, include files, and copy books which map the COMMAREA are described in "Reference information for analyzer programs".

The full range of functions which an analyzer program can perform is as follows:

- Determine whether processing should continue for the request, or whether XCICS should return an error response to the Web client.
- Analyze the content of the request, and any parameters that have been passed to the converter program from a URIMAP definition, to determine which of the subsequent processing stages are required, and which XCICS resources are needed to carry out each stage. (The EXEC XCICS WEB API commands may be used during this analysis.)
- Specify the name of a converter program to process the request before it is passed to an application program. Converter programs are normally used with application programs that are not Web-aware. A user token is provided for the analyzer program to communicate with the converter program, if required. The request is passed to the converter program in a 32K block of storage indicated by a pointer in the COMMAREA. "Converter programs" explains the functions of a converter program.
- Specify the name of the user-written application program that is to process the request and provide the response.
- Specify the transaction ID of the alias transaction that handles the remaining stages of processing.
- Specify a user ID that is to be associated with the alias transaction.
- Modify the request body. Any changes made are visible in the data passed to the converter program in the block of storage, but not to the EXEC XCICS WEB API commands.

XCICS supplies the default analyzer program DFHWBAHX, which is described in "XCICS-supplied default analyzer program DFHWBAHX". If this analyzer does not meet your requirements, you might be able to use it as an example.

Input to an analyzer program

Input parameters are passed to the analyzer program in a COMMAREA, giving information about the nature and content of the request, and any input supplied by a URIMAP definition. The analyzer program can choose to accept these values and pass them on as output parameters, or it can dynamically override them based on its analysis of the content of the request.

The input parameters include the following items, or a pointer to them:

- An eye-catcher for an analyzer parameter list.
- The dotted decimal IP address of the client and of the server (XCICS as an HTTP server).
- An indicator of whether the request is an HTTP request.
- An indicator of whether a matching URIMAP definition was found for the request. If this indicator is positive, the URIMAP definition might have passed additional input parameters to the analyzer program.
- The HTTP version.
- The request method.
- The host name specified for the request, taken from the Host header or (for an absolute URI) from the request URL. For HTTP/1.1 requests, a host name is required, so this parameter is always passed to the analyzer. For HTTP/1.0 requests, a host name might not be supplied.
- The path component of the URL.
- Any query string that was specified for the request.
- The HTTP headers for the request.
- The request body, or as much of the request body as will fit into a 32K block of storage.
For HTTP requests received on a connection using SSL client authentication, the following parameter is also passed:

- The user ID obtained from the client certificate.

If a matching URIMAP definition was found for the request and has invoked the analyzer program, the following parameters from the URIMAP definition are passed to the analyzer program, if they were present in the URIMAP definition:

- The name of the recommended converter program to process the request before it is passed to an application program (CONVERTER attribute in the URIMAP definition).
- The name of the recommended user-written application program to process the request and provide the response (PROGRAM attribute in the URIMAP definition).
- The transaction ID of the recommended alias transaction to cover the remaining stages of processing (TRANSACTION attribute in the URIMAP definition).
- The recommended user ID that is to be associated with the alias transaction (USERID attribute in the URIMAP definition).

The wbra_urimap input parameter can be used to test whether or not a URIMAP definition was used in the processing path for the request.

You can also use the EXEC CICS WEB API commands to examine the HTTP request, if preferred. Using the EXEC CICS WEB commands can increase the accuracy and completeness of your analysis of the request, particularly when examining the HTTP headers, which are subject to wide variation in content and usage. The EXEC CICS WEB commands also simplify the process of locating and extracting query string or formfield information from a request, which can be a key determinant of subsequent processing.

You can use the EXTRACT TCPIP command to obtain the following information about the client request which is being processed:

- The Web client's IP address
- The Web client's host name, as known by the DNS server
- The number of the port on which the Web client sent its connection request
- The server's (that is, XCICS as an HTTP server's) IP address
- The type of authentication in use
- The level of SSL support in use
- The TCPIPSERVICE definition associated with the request

**Output from an analyzer program**

An analyzer program provides output in a COMMAREA. The output includes a response code, and a range of optional output parameters that can be used to specify further processing stages and to share information with a converter program.

The analyzer program must provide the following output in its COMMAREA:

- A response code.
  - If your analyzer program returns a response code of URP_OK, processing continues with the next step.
  - If your analyzer program returns any other value, XCICS returns an error response to the Web client.

The analyzer program may also provide the following outputs:

- The name of a converter program that is to be used to process the request before it is passed to a user-written application program.
  - If a converter program name was input from a URIMAP definition, you can accept or override this.
  - If the analyzer indicates that a converter program is not required, the first 32K bytes of the request is passed to the
user-written application program in a block of storage. A Web-aware application can ignore this and use the EXEC CICS WEB API commands to read the request.

- The name of an application program that is to process the request and provide the response.
  - If a program name was input from a URIMAP definition, you can accept or override this.
  - If you are using a converter program, the converter program can specify or override the program name. A converter can be used in this way to involve more than one program in processing the request.
- The transaction ID of the alias transaction that is to cover the remaining stages of processing. If a transaction ID was input from a URIMAP definition, you can accept or override this.
- The user ID that is to be associated with the alias transaction. If a user ID was input from a URIMAP definition, you can accept or override this. This is how XCICS determines the user ID if you do not specify one:
  - If a user ID was input from a URIMAP definition, that is used.
  - If the HTTP request uses SSL with client authentication, the user ID is obtained from the client certificate.
  - In other cases, the XCICS default user ID is used.
- An eight byte user token, used to share information between the analyzer and converter programs.
- A modified value for the request body length.

The analyzer can modify the contents of the request:

- The modified data can be shorter than, or of the same length as the original data. The request body cannot be lengthened.
- Any changes made are visible in the data passed to the converter program, but not to the EXEC CICS WEB API commands.

### 3.2.5.5.2 Sharing data between analyzer and converter programs

XCICS passes three parameters between the analyzer and the converter programs that enable data to be shared by these processing stages:

**user_data_pointer**

This parameter contains the address of a 32K block of storage that is passed from stage to stage. On entry to the analyzer, the pointer points to a block of storage containing the HTTP request. On completion of the encode function of the converter, XCICS Web support uses it to locate the block of storage containing the HTTP response.

You must not change the value of the pointer in the analyzer, although you can modify the contents of the block of storage addressed by the pointer.

Between the converter and the XCICS application program, you can pass the pointer unchanged from one stage to another, or you can issue a GETMAIN command in one program and pass the address of the newly acquired storage in the pointer.

**user_data_length**

This parameter is the length of the block of storage addressed by the user_data pointer.

**user_token**

The user token is an 8-byte field which is shared by the analyzer and the converter. It can contain any information you wish:

- You can pass small quantities of shared information directly in the user token.
- To pass larger quantities, you can issue a GETMAIN command in one program, to acquire storage for a shared work area.
Use the user token to pass the address of the shared storage.

You can change the contents of the user token in each program: for example, the user token can have one meaning when passed from the analyzer to the decode stage of the converter, and a different meaning when passed to the encode stage.

**Note:** The analyzer and converter programs execute under different XCICS tasks. Therefore, if you issue a GETMAIN command in the analyzer program, you must code the SHARED option if the storage is to be visible in the converter program. In general, storage acquired with the SHARED option is not freed automatically by XCICS, so you must issue a GETMAIN command when your programs no longer need the storage. However, XCICS will free the storage addressed by the user_data pointer after the HTTP response has been sent to the Web client.

### 3.2.5.5.3 XCICS-supplied default analyzer program DFHWBAHX

XCICS supplies a default analyzer program, DFHWBAHX. If you need to provide request handling through your analyzer program, as well as or instead of through URIMAP definitions, you can use DFHWBAHX itself as a starting point for writing your own analyzer program. It is C language program and it is located in the $XFRAMEHOME/xcics/src directory.

As supplied, DFHWBAHX does not perform any analysis of a request when a matching URIMAP definition has been found for the request, even if the URIMAP specifies "analyzer=yes". This means that the settings specified in the URIMAP definition for the alias transaction, converter program and application program are automatically accepted and used to determine subsequent processing stages.

DFHWBAHX uses the wbra_uriamp input parameter to test for the presence of a URIMAP definition, and if the result is positive, returns without performing any analysis on the request URL.

Otherwise it interprets the HTTP requests in which the path component of the URL has the following syntax:

```
/_converter/_alias/_program_________ _________ _                   |_/_ignored_|_?_token_|
```

All fields processed by the analyzer program are translated to upper case.

After translation:

- **converter:** Specifies the name of the converter program to be used for the request. It can be up to eight characters in length. As a special case, the four character value 'CICS' denotes that no converter program is used.

- **alias:** Specifies the transaction ID of the alias transaction for subsequent request processing. It can be up to four characters in length.

- **program:** Specifies the name of the XCICS application program that is to be used to service the request. It can be up to eight characters in length.

- **ignored:** This part of the path is ignored by DFHWBAHX (but may be used by the converter program or the application program).

- **token:** The first eight bytes specify the user token that is passed to the converter program. Data following the first eight bytes of the token is ignored by DFHWBAHX (but may be used by the converter program or the application program).

### Responses from DFHWBAHX

The meanings of the responses produced by DFHWBAHX are as follows:

- **URP_OK:** The analyzer found that the request conformed to the default HTTP request format, and generated the appropriate outputs for the alias.

- **URP_EXCEPTION:** The analyzer found that the request did not conform to the default format.

- **URP_INVALID:** The eye-catcher was invalid. This indicates an internal error.
3.2.5.5.4 Reference information for analyzer programs

The names of the parameters and constants for analyzer programs, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of XCICS.

<table>
<thead>
<tr>
<th>Language</th>
<th>Parameters file</th>
<th>Constants file</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>dfhwbtdh.h</td>
<td>dfhwbuch.h</td>
<td>$XFRAMEHOME/xcics/cpy</td>
</tr>
<tr>
<td>COBOL</td>
<td>DFHWBTD0.cpy</td>
<td>DFHWBUCO.cpy</td>
<td>$XFRAMEHOME/xcics/cobcpy</td>
</tr>
</tbody>
</table>

These files give language-specific information about the data types of the fields in the COMMAREA.

Parameters description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wbra_alias_tranid</td>
<td>Input/Output</td>
<td>A string of length 4. The transaction ID of the alias transaction that is to cover the remainder of processing for this request. If a URIMAP definition is involved, this contains the value of the TRANSACTION attribute. If you do not set this field, or if you set it to blanks, CWBA is used.</td>
</tr>
<tr>
<td>wbra_alias_termid</td>
<td>Output</td>
<td>A string of length 4. The terminal ID to be used on the START request for the alias transaction that is to cover the remainder of processing for this request.</td>
</tr>
<tr>
<td>wbra_characterset</td>
<td>Output</td>
<td>The name of the IANA character set that the client used for the entity body of the request. This information is used for code page conversion of the entity body of the request and the response. If the request is not an HTTP request, this character set is used to translate the entire request and response. wbra_hostcodepage must also be supplied.</td>
</tr>
<tr>
<td>wbra_client_ip_address</td>
<td>Input</td>
<td>The 32-bit IP address of the client.</td>
</tr>
<tr>
<td>wbra_commarea</td>
<td>Output</td>
<td>The flag to indicate that pre-IBM CICS TS Version 3 compatibility processing is required for a response that uses a non-Web-aware application and a converter program.</td>
</tr>
<tr>
<td>wbra_content_length</td>
<td>Input</td>
<td>A 32-bit binary representation of the entity body length as specified by the Content-Length HTTP header in the received data.</td>
</tr>
<tr>
<td>wbra_converter_program</td>
<td>Input/Output</td>
<td>A string of length 8. The name of the converter program that is used to process the request. If a URIMAP definition is involved, this contains the value of the CONVERTER attribute. If this field is not set on output, no converter program is called.</td>
</tr>
<tr>
<td>wbra_dfhcnv_key</td>
<td>Output</td>
<td>A string of length 8. The name of a conversion template in the DFHCNV table for code page conversion of the entity body for the request and the response. If the request is not an HTTP request, this template is used to translate the entire request and response. XCICS initializes this field to high values. If you use this field to specify a conversion template, the name you choose must be defined in the DFHCNV table, as described in &quot;Migrating entries in the code page conversion table (DFHCNV)&quot; in topic 2.1.3. As an alternative, you can set the wbra_hostcodepage and wbra_characterset fields to specify the pair of code pages to use for code page conversion. If you set wbra_dfhcnv_key to nulls or blanks and do not set wbra_hostcodepage and wbra_characterset, code page conversion is suppressed.</td>
</tr>
<tr>
<td>wbra_eyecatcher</td>
<td>Input</td>
<td>A string of length 8. Its value is &quot;analyze&quot;.</td>
</tr>
<tr>
<td>wbra_function</td>
<td>Input</td>
<td>A code indicating that an analyzer program is being called. The value is 1.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Mode</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wbra_hostcodepage</td>
<td>Output</td>
<td>The name of a host code page suitable for the application program that is handling the request.</td>
</tr>
<tr>
<td>wbra_hostname_length</td>
<td>Input</td>
<td>The length in bytes of the host name specified on the HTTP request. If no host name was specified, the value is undefined.</td>
</tr>
<tr>
<td>wbra_hostname_ptr</td>
<td>Input</td>
<td>A pointer to the host name specified on the HTTP request sent by the client. If an absolute URI was used for the request, the host name is taken from the URL. Otherwise the host name is as specified in the Host header for the request. For HTTP/1.1 requests, a host name is required, so this parameter is always passed to the analyzer. For HTTP/1.0 requests, a host name might not be supplied, in which case the value is undefined.</td>
</tr>
<tr>
<td>wbra_http_version_length</td>
<td>Input</td>
<td>For an HTTP request, the length in bytes of the string identifying the HTTP version of the client's request. If the request is not an HTTP request, the value is zero.</td>
</tr>
<tr>
<td>wbra_http_version_ptr</td>
<td>Input</td>
<td>For an HTTP request, a pointer to the string identifying the HTTP version of the client's request. If the request is not an HTTP request, the value is undefined.</td>
</tr>
<tr>
<td>wbra_method_length</td>
<td>Input</td>
<td>For an HTTP request, the length in bytes of the string identifying the method specified in the HTTP request. If the request is not an HTTP request, the value is zero.</td>
</tr>
<tr>
<td>wbra_method_ptr</td>
<td>Input</td>
<td>For an HTTP request, a pointer to the method specified in the HTTP request. If the request is not an HTTP request, the value is undefined.</td>
</tr>
<tr>
<td>wbra_querystring_length</td>
<td>Input</td>
<td>The length in bytes of the query string specified on the HTTP request. If no query string was sent, the value is undefined.</td>
</tr>
<tr>
<td>wbra_querystring_ptr</td>
<td>Input</td>
<td>A pointer to the query string specified on the HTTP request sent by the client. If no query string was sent, the value is undefined.</td>
</tr>
<tr>
<td>wbra_reason</td>
<td>Output</td>
<td>The reason code returned by the analyzer program. See “Response and reason codes”.</td>
</tr>
<tr>
<td>wbra_request_header_length</td>
<td>Input</td>
<td>For an HTTP request, the length of the first HTTP header in the HTTP request. If the request is not an HTTP request, the value is zero.</td>
</tr>
<tr>
<td>wbra_request_header_ptr</td>
<td>Input</td>
<td>For an HTTP request, a pointer to the first HTTP header in the HTTP request. The other HTTP headers follow this one in the request buffer. If the request is not an HTTP request, the value is undefined.</td>
</tr>
<tr>
<td>wbra_request_type</td>
<td>Input</td>
<td>If this is an HTTP request, the value is WBRA_REQUEST_HTTP. If this is not an HTTP request, the value is WBRA_REQUEST_NON_HTTP.</td>
</tr>
<tr>
<td>wbra_resource_escaped_ptr</td>
<td>Input</td>
<td>For an HTTP request, a pointer to a copy of the HTTP headers for the request which have not been unescaped (that is, are still in their escaped form).</td>
</tr>
<tr>
<td>wbra_resource_length</td>
<td>Input</td>
<td>For an HTTP request, the length in bytes of the path component of the URL. If the request is not an HTTP request, the value is zero.</td>
</tr>
<tr>
<td>wbra_resource_ptr</td>
<td>Input</td>
<td>For an HTTP request, a pointer to the path component of the URL. If a URIMAP definition is involved, this contains the value of the PATH attribute. If the request is not an HTTP request, the value is undefined.</td>
</tr>
<tr>
<td>wbra_response</td>
<td>Output</td>
<td>The response code returned by the analyzer program. See “Response and reason codes”.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Mode</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wbra_server_ip_address</td>
<td>Input</td>
<td>The 32-bit IP address that is being used by XCICS as an HTTP server.</td>
</tr>
<tr>
<td>wbra_server_program</td>
<td>Input/output</td>
<td>A string of length 8. The name of a XCICS application program that is to process the request. If a URIMAP definition is involved, this contains the value of the PROGRAM attribute. The program name is passed to any converter program specified in wbra_converter_program. If you do not set this field, the value passed is nulls. The program name must be set here or by the converter program, or no XCICS application program will be called.</td>
</tr>
</tbody>
</table>
| wbra_unescape                 | Output       | • To specify that data is to be passed to the XCICS application program in its unescaped form, set this parameter to WBRA_UNESCAPE_REQUIRED.  
• To specify that data is be passed to the application in its escaped form, set this parameter to WBRA_UNESCAPE_NOT_REQUIRED. This is the default value.  

You should also set the parameter to WBRA_UNESCAPE_NOT_REQUIRED if your analyzer has converted the data to its escaped form. |
| wbra_urimap                   | Input        | The name of any matching URIMAP definition that is involved in the processing path for the request. If this field is non-blank, the XCICS-supplied default analyzer DFHWBADX returns without processing the path component of the URL. |
| wbra_user_data_length         | Input/output | A 15-bit integer, representing the length of the entity body in the HTTP request. If the request is non-HTTP, this value is the length of the request. The length passed to the analyzer includes any trailing carriage return and line feed (CRLF) characters that may delimit the end of the entity body. If the analyzer reduces the length of the entity body, the newly redundant bytes are replaced by null characters, X'00'. The modified value is passed on to the XCICS business logic interface in field wbbl_user_data_length, and to the converter program in field decode_user_data_length. |
| wbra_user_data_ptr            | Input        | For an HTTP request, a pointer to the entity body in the HTTP request. If the request is not an HTTP request, this is a pointer to the request. |
| wbra_user_token               | Output       | A 64-bit token that is passed to the converter program as decode_user_token. If you do not set this field, the value passed is null. If there is no converter program for this request, the value is ignored. |
| wbra_userid                   | Input/output | A string of length 8. On input, this is a user ID supplied by the client (using basic authentication or client certificate authentication), or if a URIMAP definition is involved, the value of the USERID attribute (if specified). On output, it is the user ID that is used for the alias transaction, which can be the supplied user ID or a user ID chosen by the analyzer program. If this field is blank or null on output, the XCICS default user ID is used. |

### Response and reason codes

An analyzer program must return one of the following values in wbra_response:

- URP_OK                      0
- URP_EXCEPTION               4
- URP_INVALID                 8
- URP_DISASTER                12
URP_OK_LOOP 16

In case of a response code not equal to URP_OK, a reason code may be supplied in wbra_reason:

URP_RESOURCE_TOO_SHORT 1
URP_FIRST_SLASH_MISSING 2
URP_CONV_NAME_INVALID 4
URP_TRAN_NAME_INVALID 5
URP_SERV_NAME_INVALID 6
URP_USER_TOKEN_INVALID 7
URP_SERVER_NAME_MISSING 8
Converter programs

Converter programs are primarily for use with application programs that were not originally coded for use with the Web. They can also be used to combine output from several application programs into a single HTTP message.

Converter programs are not used when XCICS is an HTTP client, or for Web service processing; they can only be invoked when XCICS is an HTTP server. The role of converter programs in the XCICS Web support process for XCICS as an HTTP server is described in "HTTP request and response processing for XCICS as an HTTP server".

A URIMAP definition can invoke a converter program to carry out relevant processing for HTTP requests. If an analyzer program is used in XCICS Web Support processing, the analyzer program can also invoke a converter program. A converter program can be useful in the following circumstances:

- When application programs that were not originally coded for use with the Web need to receive input in the form of a COMMAREA, or need their output to be converted into an HTTP response. Web-aware application programs, which are coded using the EXEC CICS WEB and EXEC CICS DOCUMENT application programming interfaces, should not require this conversion to take place. You can use a converter program to perform this conversion or other processing on the content of the request.

- When you want to make more than one application program work on the same request data in sequence, and return a single HTTP response to the Web client.

If a converter program is invoked directly from a URIMAP definition, the "program" attribute of the URIMAP definition (which specifies the name of the application program to process the request) can be passed to the converter program, and the converter program can choose to override it.

A converter program is not associated with a TCPIP SERVICE definition in the same way as an analyzer program. You can use any converter program that is local to the XCICS system, to process any HTTP request. For a given request, the same converter program is called for both the decode and encode functions.

The converter program has two functions:

**Decode**

The HTTP request is passed to the decode function of the converter program in a 32K block of storage, which is indicated by the user_data pointer in the COMMAREA. It has already been divided into separate elements, such as the method, request headers and body. (Note that if the request is too long to fit into the block of storage, the remainder of the data is not passed to the converter program.) The name of the application program that should handle the request is also supplied, either taken from the URIMAP definition for the request, or set by the analyzer program. If an analyzer program is used, it can provide additional information in a user token. Using this information, the decode function:

- Determines whether processing should continue for the request, or whether XCICS should return an error response to the Web client.

- Specifies the name of the user-written application program that is to process the request and provide the response. If the name has already been input from a URIMAP definition or by an analyzer program, the converter program can accept or change this.

- Constructs the COMMAREA that is passed to the user-written application program. The COMMAREA includes data from the Web client's request which has been converted into an acceptable input format for the application program. The storage containing the HTTP request can be reused, or a new COMMAREA can be specified.

**Encode**

Using the COMMAREA returned by the user-written application program, the encode function:

- If more than one application program is to supply data, sets the loop response, to call the decode function again. The decode function changes the name of the application program and supplies appropriate input in a COMMAREA. The output is returned to the encode function again.

- Constructs an HTTP response in a COMMAREA, which is to be sent to the Web client.
3.2.5.6.1 Writing a converter program

You can write your analyzer in C, COBOL, or PL/I. Language-dependent header files, include files, and copy books are described in "Reference information for converter programs".

The converter must run in the same XCICS region as the TCPIPSERVICE which receives the request.

The size and contents of the COMMAREA that is passed from the converter program to the user application program are governed by the design of the XCICS application program that is invoked for the HTTP request. When you use a converter program with XCICS Web support, you can:

- Issue a GETMAIN command to obtain storage for the COMMAREA.
- Use storage acquired in an earlier stage of processing (such as the analyzer program) for the COMMAREA.
- Use the storage which contains the HTTP request for the COMMAREA.

Input parameters for the decode function

Input parameters are passed to the decode function in a COMMAREA. The parameters include:

- The IP address of the Web client
- A pointer to the HTTP version
- A pointer to the method
- A pointer to the path component of the URL
- A pointer to the request headers
- A pointer to the entity body
- The name of the XCICS application program that will service the request (as set by the analyzer)
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs".
- An iteration counter that records the number of times the decode function has been entered for each HTTP request. The counter is set to 1 before the decode function is called for the first time, and is incremented before it is called on each subsequent occasion.
- An indication of whether the address of the entity body can be the target of a FREEMAIN command.

Output parameters for the decode function

The decode function must provide the following outputs in its COMMAREA:

- A response code (optionally qualified by a reason code).
  - If the decode function returns a response code of URP_OK, processing continues with the next step.
  - If the decode function returns any other value, the HTTP request is rejected.
- The address and length of the COMMAREA passed to the XCICS application program. If no application program is called, the COMMAREA is passed unchanged to the encode function. It may also provide the following outputs:
  - The name of the XCICS application program that is to service the request. If the analyzer supplied a name, the converter can change it, or specify that no application program should be called.
  - An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs".

Input parameters for the encode function

Input parameters are passed to the encode function in a COMMAREA. The parameters include:

- The address and length of the COMMAREA returned by the XCICS application program. If no application...
program was called, the COMMAREA is passed unchanged from the decode function.

- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs".
- An iteration counter that records the number of times the encode function has been entered for each HTTP request. The counter is set to 1 before the encode function is called for the first time, and is incremented before it is called on each subsequent occasion.

Output parameters for the encode function

The encode function can provide the following outputs:

- A response code (optionally qualified by a reason):
  - If the encode function returns a response code of URP_OK, an HTTP response is sent to the Web client.
  - If the encode function returns any other value, the HTTP request is rejected.
- If the response code is URP_OK, the address of a buffer containing the HTTP response to be sent to the Web client, and the length of the HTTP response. The first word of the buffer must contain the length of the data (that is, the length of the HTTP response plus 4).
- An eight byte user token, used to share information between the analyzer and converter programs. See "Sharing data between analyzer and converter programs".

3.2.5.6.2 Converter programs: Constructing an HTTP response

The size and contents of the HTTP response that you construct in the converter program are governed by:

- The number and length of the HTTP headers that you want to include in the response.
- The length and contents of the HTML data that you want to send in your response.

In turn, these will be governed by the design of the user-written application program and the information that it returned in its COMMAREA.

The response must contain an HTTP version, status code, status text, any HTTP headers that are required, and the message body. The format of the response should be compliant with the HTTP protocol specification to which you are working (HTTP/1.0 or HTTP/1.1).

XCICS inserts automatically some HTTP headers suitable for an HTTP/1.0 or HTTP/1.1 response.

When you use a converter program with XCICS Web support, you can:

- Issue a GETMAIN command to obtain storage for the HTTP response.
- Use storage acquired in an earlier stage of processing (such as the analyzer program) for the COMMAREA.
- Construct the response in the COMMAREA returned by the application program.

The first word of the area used for the response must contain the length of the area (that is, the length of the HTTP response plus 4).

3.2.5.6.3 Reference information for converter programs

The names of the parameters and constants for converter programs, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of XCICS.

<table>
<thead>
<tr>
<th>Language</th>
<th>Parameters file</th>
<th>Constants file</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>dfhwbcdh.h</td>
<td>dfhwbuch.h</td>
<td>EXFRAMEHOME/xcics/cpy</td>
</tr>
<tr>
<td>COBOL</td>
<td>DFHWBCDO.cpy</td>
<td>DFHWBUCO.cpy</td>
<td>EXFRAMEHOME/xcics/cobcpy</td>
</tr>
</tbody>
</table>
These files give language-specific information about the data types of the fields in the COMMAREA.

## Parameters description for the DECODE function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decode_client_address</td>
<td>Input</td>
<td>The 32-bit IP address of the client.</td>
</tr>
<tr>
<td>decode_client_address_string</td>
<td>Input</td>
<td>The IP address of the client in dotted decimal format.</td>
</tr>
<tr>
<td>decode_data_ptr</td>
<td>Input/Output</td>
<td>On input, a pointer to the request from the client (as modified by the analyzer) or, if this call is a loop back from the Encode converter function, a pointer to the response data of encode_data_ptr. On output, pointer to the COMMAREA to be passed to the XCICS application program. Do not modify this parameter when decode_volatile has a value of 0.</td>
</tr>
<tr>
<td>decode_entry_count</td>
<td>Input</td>
<td>A count to say how many times the Decode converter has been entered for the current Web request.</td>
</tr>
<tr>
<td>decode_eyecatcher</td>
<td>Input</td>
<td>A string of length 8. Its value for Decode is &quot;&quot;decode_&quot;.&quot;</td>
</tr>
<tr>
<td>decode_function</td>
<td>Input</td>
<td>A halfword code set to the constant value URP_DECODE, indicating that Decode is being called.</td>
</tr>
<tr>
<td>decode_http_version_length</td>
<td>Input</td>
<td>The length in bytes of the string identifying the HTTP version supported by the client. If the request is not an HTTP request, or decode_entry_count is greater than 1, the value is zero.</td>
</tr>
<tr>
<td>decode_http_version_ptr</td>
<td>Input</td>
<td>A pointer to the string identifying the HTTP version supported by the client. If the analyzer modified this part of the request, the changes are visible here. If decode_http_version_length is zero, the value is undefined.</td>
</tr>
<tr>
<td>decode_input_data_len</td>
<td>Input/Output</td>
<td>On input, this is the length in bytes of the request data pointed to by decode_data_ptr. The value to be used for the DATALENGTH option of a LINK command for the XCICS application program. The default value if this output is not set is 32K.</td>
</tr>
<tr>
<td>decode_method_length</td>
<td>Input</td>
<td>The length in bytes of the method specified in the HTTP request. If the request is not an HTTP request, or decode_entry_count is greater than 1, the value is zero.</td>
</tr>
<tr>
<td>decode_method_ptr</td>
<td>Input</td>
<td>A pointer to the method specified in the HTTP request. If the analyzer modified this part of the request, the changes are visible here. If decode_method_length is zero, the value is undefined.</td>
</tr>
<tr>
<td>decode_output_data_len</td>
<td>Output</td>
<td>The value to be used for the LENGTH option of the LINK command for the XCICS application program. The default value if this output is not set is 32K.</td>
</tr>
<tr>
<td>decode_reason</td>
<td>Output</td>
<td>A reason code. See &quot;Responses and reason codes&quot;.</td>
</tr>
<tr>
<td>decode_request_header_length</td>
<td>Input</td>
<td>The length of the first HTTP header in the HTTP request. If the request is not an HTTP request, or decode_entry_count is greater than 1, the value is zero.</td>
</tr>
<tr>
<td>decode_request_header_ptr</td>
<td>Input</td>
<td>A pointer to the first HTTP header in the HTTP request. If the analyzer modified this part of the request, the changes are visible here. If decode_request_header_length is zero, the value is undefined.</td>
</tr>
</tbody>
</table>
### Parameters description for the ENCODE function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
</table>
| encode_data_ptr | Input/Output| On input, a pointer to the COMMAREA returned by the XCICS application program. If no application program was called, it is a pointer to the COMMAREA created by Decode.  
On output, a pointer to the buffer containing the response to be sent to the client. You must ensure that the pointer points to a valid location, or results can be unpredictable. The buffer must be doubleword aligned. The first four bytes must be a 32-bit unsigned number specifying the length of the buffer. (In COBOL, specify this as PIC 9(8) COMP.) The rest of the buffer is the response. |
<p>| encode_entry_count | Input       | A count to say how many times the Encode converter has been entered for the current Web request.                                             |</p>
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>encode_eyecatcher</td>
<td>Input</td>
<td>A string of length 8. Its value for Encode is &quot;&gt;encode .&quot;</td>
</tr>
<tr>
<td>encode_function</td>
<td>Input</td>
<td>A halfword code set to the constant value URP_ENCODE, indicating that Encode is being called.</td>
</tr>
<tr>
<td>encode_input_data_len</td>
<td>Input</td>
<td>The length of the COMMAREA as specified by Decode in decode_output_data_len.</td>
</tr>
<tr>
<td>encode_reason</td>
<td>Output</td>
<td>A reason code. See &quot;Responses and reason codes&quot;.</td>
</tr>
<tr>
<td>encode_response</td>
<td>Output</td>
<td>A response code. See &quot;Responses and reason codes&quot;.</td>
</tr>
<tr>
<td>encode_user_token</td>
<td>Input</td>
<td>The 64-bit token output by Decode as decode_user_token.</td>
</tr>
<tr>
<td>encode_version</td>
<td>Input</td>
<td>A single-character parameter list version identifier: always set to character zero ('0'), indicating an IBM CICS TS 1.3 version parameter list.</td>
</tr>
<tr>
<td>encode_volatile</td>
<td>Input</td>
<td>A single-character code indicating whether the data area pointed to by encode_data_ptr can be replaced. Possible values are: 0 - The area is part of another COMMAREA and cannot be replaced. 1 - The storage pointed to by encode_data_ptr can be freed and replaced by a different size work area.</td>
</tr>
</tbody>
</table>

**Response and reason codes**

An analyzer program must return one of the following values in wbra_response:

- URP_OK 0
- URP_EXCEPTION 4
- URP_INVALID 8
- URP_DISASTER 12
- URP_OK_LOOP 16

In case of a response code not equal to URP_OK, a reason code may be supplied in wbra_reason:

- URPRESOURCE_TOO_SHORT 1
- URP_FIRST_SLASH_MISSING 2
- URP_CONV_NAME_INVALID 4
- URP_TRAN_NAME_INVALID 5
- URP_SERV_NAME_INVALID 6
- URP_USER_TOKEN_INVALID 7
- URP_SERVER_NAME_MISSING 8
3.2.5.7 Security for XCICS Web support

When XCICS is connected to the Internet, security measures are essential to prevent unauthorized access to XCICS applications and data, and also to prevent third parties obtaining private information that is sent over the Internet.

You should consider security throughout the development process for your XCICS Web support architecture, as part of the design of your XCICS Web support applications and utility programs, as well as when creating resource definitions for the relevant XCICS facilities. This section summarizes the measures that can be used to enhance the security of your XCICS Web support implementation.

The basic systems to protect your application against unauthorized accesses are:

- HTTP authentication
- SSL
- firewalls & proxies

HTTP authentication

Authentication and identification of clients enables a server to protect its resources from access by unauthorized users.

For XCICS as an HTTP server, authentication schemes are specified by the "auth" attribute of the TCPIPSERVICE definition. Identification is obtained in connection with the authentication process, or can be supplied by XCICS if authentication is not needed.

Obtaining authentication and identification from Web clients is a key step in protecting your XCICS system from access by unauthorized users.

Two authentication schemes are supported by XCICS for use with the HTTP protocol:

basic

The basic authentication is an HTTP facility that enables a client to both authenticate and identify itself to a server by providing a user ID and password. This information is encoded using base-64 encoding, which is simple to decode. Because of this, using basic authentication as the sole means of authentication is only appropriate when there is no possibility of a password being intercepted. In most environments, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

certificate

The SSL client certificate authentication is a more secure method of authenticating a client, using a client certificate which is issued by a trusted third party (or Certificate Authority), and sent using SSL encryption.

The TCPIPSERVICE must be defined with "ssl=yes" and the web client must provide a certificate when connected. XCICS validates the user and assigns the USERID, matching the common name (CN) supplied in the certificate with the "username" attribute of the users definition in the SNT (Signon Table).

When you use basic authentication or client certificate authentication, XCICS handles the process of requesting authentication from the user, decoding the authentication information if necessary, checking the supplied authentication against the security manager’s database, and rejecting the request if the authentication is not acceptable. An analyzer program or user-written application program is only called after the authentication has been verified and accepted.

SSL with XCICS Web support

The Secure Sockets Layer (SSL) can be used with HTTP to enable encryption, message authentication, and client and server authentication using certificates. The HTTPS scheme is HTTP with SSL. When you have configured XCICS to use SSL, its facilities are available for both XCICS as an HTTP server, and XCICS as an HTTP client.

When XCICS is an HTTP server, you can use SSL to protect an interaction with a Web client. To do this, specify appropriate security options on the TCPIPSERVICE definition for the port on which XCICS receives the client's requests.

When XCICS is an HTTP client, a server might require the use of SSL for some connections. If that is the case, you need to do some or all of the following:

- Use HTTPS as the scheme for the connection.
Supply a list of cipher suites that you want to use for the connection. You can specify these in the URIMAP definition that you use on the WEB OPEN command for the connection.

Supply a client certificate. Client certificates are not a requirement for all SSL transactions, but a server might require one for particular transactions. If a server does request a client certificate, you can specify the label of a suitable certificate in the URIMAP definition that you use on the WEB OPEN command for the connection.

XCICS uses the OpenSSL Secure Socket Layer implementation (www.openssl.org).

Setting the server certificate

A server certificate must be installed for XCICS as an HTTP server. XCICS manages certificate and keys file in PEM format. Both key and certificate must be installed on the system running the region and their path must be defined in the XCICS configuration file, with the "set server_ssl_certificate" and "set server_ssl_key" directives.

```
set server_ssl_certificate=$HOME/etc/certificate.pem
set server_ssl_key=$HOME/etc/key.pem;
```

The certificate should be emitted by a trusted authority. By the way, it is also possible to use auto-signed certificates for testing purposes:

```
# openssl genrsa -des3 -rand file1:file2:file3:file4:file5 -out demo.key 1024
0 semi-random bytes loaded
Generating RSA private key, 1024 bit long modulus
............,+++++
............,+++++
e is 65537 (0x10001)

Enter pass phrase for demo.key: ****
Verifying - Enter pass phrase for demo.key: ****

# openssl rsa -in demo.key -out demo.pem
Enter pass phrase for demo.key: ****
writing RSA key

# openssl req -new -key demo.key -out demo.csr
Enter pass phrase for demo.key: ****
You are about to be asked to enter information that will be incorporated into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter ".", the field will be left blank.

Country Name (2 letter code) [GB]:IT
State or Province Name (full name) [Berkshire]:Italy
Locality Name (eg, city) [Newbury]:Rome
Organization Name (eg, company) [My Company Ltd]:HTWC
Organizational Unit Name (eg, section) []:R&D
Common Name (eg, your name or your server’s hostname) []:www.htwc.com
Email Address [ ]:master@htwc.com

Please enter the following 'extra' attributes to be sent with your certificate request
A challenge password []:
An optional company name []:
Signature ok

# openssl x509 -req -days 60 -in demo.csr -signkey demo.key -out demo.crt
Getting Private key
Enter pass phrase for demo.key: ****
Setting the client certificates

Programs using the XCICS Web Support as Web Clients may require to connect an HTTPS server, which may request or require to provide a client certificate.

The client certificate used in a Web Client connection may be defined in the WEB OPEN command either with the CERTIFICATE parameter or by the URIMAP definition. In both cases, a certificate "label" is supplied to XCICS.

This label is the name (without extension) of both certificate and key files. These files must in PEM format, and must be located in the path defined with the "set client_ssl_certificate_path" and "set client_ssl_key_path" in the XCICS configuration file. Their name must conform to

<LABEL>.pem

For example, in the configuration file:

```plaintext
set client_ssl_certificate_path=$HOME/etc/sslclient/certificate;
set client_ssl_key_path=$HOME/etc/sslclient/key;
```

In the file system:

```plaintext
# ls -l $HOME/etc/sslclient/certificate
total 4
-rw-r--r-- 1 fabrizio devel 887 Dec 20 16:01 CLICERT.pem

# ls -l $HOME/etc/sslclient/key
total 4
-rw-r--r-- 1 fabrizio devel 887 Dec 20 16:01 CLICERT.pem
```

In the program:

```plaintext
EXEC CICS WEB OPEN SESSTOKEN(ST)
HTTPS
HOST( HOST-NAME )
HOSTLENGTH( HOST-LENGTH )
PORTNUMBER( PORT )
CODEPAGE('037')
HTTPVNUM( VNUM ) HTTPRNUM( VREL )
CIPHERS( CYP ) NUMCIPHERS( CLEN )
CERTIFICATE( 'CLICERT' )
END-EXEC
```
3.2.5.8 HTTP client requests from a CICS application

XCICS can act as an HTTP client, and communicate with an HTTP server on the Internet. A user-written application program sends requests through XCICS to the HTTP server, and receives the responses from it.

Note that the HTTP client facility of XCICS Web support is not designed for use as a browser. User application programs can make requests for individual, known resources that are available from a server, but they would not be expected to browse the Internet generally. The range of responses that you might receive from a server, and the actions that you need to take to handle them, should relate only to your preselected resources, plus the error responses that might be associated with those resources and with the type of requests you are making.

“HTTP request and response processing for XCICS as an HTTP client” explains the processing structure for XCICS as an HTTP client. Before writing an application program that makes an HTTP client request, make sure you understand the processing stages for these requests, because most of the stages are initiated by the application program itself.

Several XCICS Web support facilities are used when XCICS is an HTTP client:

- You can use EXEC CICS WEB API commands in XCICS application programs to open an HTTP connection to a server, make requests, and receive responses for processing by the application program.
- URIMAP definitions may be used to avoid specifying information such as a URL or a certificate label in your application program.
- TCPIPSERVICE resource definitions, which are used for XCICS as an HTTP server, do not apply to XCICS as an HTTP client, and you do not need to create these to make HTTP client requests.

3.2.5.8.1 Making HTTP requests through CICS as an HTTP client

HTTP client requests made from XCICS to a server on the Internet are initiated by a user-written application program. This topic tells you how to write an application program that makes an HTTP client request.

For XCICS as an HTTP client, the application program makes requests to a server, and waits for the responses. An application program can control more than one connection, using a session token to differentiate between them.

To make HTTP requests and receive responses, write your application program to follow this process:

1. Initiate a connection to the server, using the WEB OPEN command.
2. Write HTTP headers for the request, using the WEB WRITE HTTPHEADER command.
3. If required, produce an entity body, or message body, which is the content of the HTTP request. For many request methods, an entity body is not used, but for the POST and PUT methods (which are used to supply data to the server) it is required. The entity body is formed from an XCICS document (which is created using the EXEC CICS DOCUMENT application programming interface) or from a buffer of data supplied by the application program.
4. Send the request to the Web client using the WEB SEND or WEB CONVERSE command.
5. Wait for and receive the request.
6. Receive the message body of the response using the WEB RECEIVE command (or the WEB CONVERSE command that you issued earlier).
7. Read the headers for the response using the WEB READ HTTPHEADER command, or the HTTP header browsing commands.
8. Process the server’s response, depending on the status code, and execute the application’s business logic.
9. If further requests and responses are wanted, repeat the process. If the server supports persistent connections, and does not close the connection, there is no need to open a new connection. You can continue using the same session token. If the server closes the connection, you need to issue the WEB OPEN command again if you want to make further requests.
10. When you have finished working with the server, close the connection.
Opening a connection to an HTTP server

When making an HTTP client request in XCICS Web support, you must open a connection to the server before sending the first request. XCICS returns a session token that represents the connection.

Initiate a connection with the server by issuing a WEB OPEN command as follows:

1. Specify the host name of the server, the length of the host name, and the scheme that is to be used (HTTP or HTTPS). Also specify the port number for the host if this is other than the default for the specified scheme. You can specify the URMAP option on the WEB OPEN command to use this information directly from an existing URMAP definition. Alternatively, you can supply the information using the SCHEME, HOST, HOSTLENGTH and PORTNUMBER options. You can extract these details from a known URL using the WEB PARSE URL command, or from an existing URMAP definition using the WEB EXTRACT URMAP command.

2. If you are using the HTTPS scheme, specify appropriate security options:
   - If you need to supply an SSL client certificate, specify the CERTIFICATE option to do this. If you specify the URMAP option on the WEB OPEN command, you can use this information directly from an existing URMAP definition.
   - Use the CIPHERS and NUMCIPHERS options to specify a list of cipher suite codes to be used for the connection. If you specify the URMAP option on the WEB OPEN command, you can either accept the setting from the URMAP definition, or specify your own list of cipher suite codes to override the URMAP specification.

3. XCICS opens the connection with the server, and returns a session token to the application program. XCICS also returns information on the HTTP version of the server.

4. Save the session token and use it on all subsequent commands that relate to this connection.

Writing HTTP headers for a request

For client HTTP requests, XCICS automatically provides the HTTP headers that are required for basic messages, depending on the HTTP protocol version used for the message. You might need to add further HTTP headers to your request.

Write additional HTTP headers for a message before you issue the WEB SEND command to send the message. The exception to this rule is if you are writing headers to be sent as trailing headers on a chunked message, in which case the special process mentioned below applies. When writing HTTP headers for a request:

- For all commands, specify the session token for this connection, using the SESSTOKEN option.
- Use the WEB WRITE HTTPHEADER command for each header that you want to add to the message.
- If you want to produce a date and time stamp for use in one of your HTTP headers (for example, the If-Modified-Since header), you can use the FORMATTIME command.
- Make sure that your application program carries out any actions that are implied by your user-written headers.

Writing an HTTP request

For XCICS as an HTTP client, the WEB SEND command or the WEB CONVERSE command can be used to make a request. The WEB CONVERSE command combines the options available on the WEB SEND command and the WEB RECEIVE command, so that you can use a single command to issue a request and receive the response.

You need to specify an HTTP method when making a request. The method tells the server what to do with your request.

When you issue your chosen command:

1. Specify the session token for this connection, using the SESSTOKEN option.
2. Specify the HTTP method for the request.
3. Specify the path information for the resource on the server that the application needs to access. The default is the path given in any URMAP definition that you referenced on the WEB OPEN command for this connection. You can specify an alternative path by using the URMAP option to name another URMAP definition from which
the path can be taken. (The new URIMAP definition must specify the correct host name for the current connection.) Alternatively, you can use the PATH and PATHLENGTH options to provide the path information.

4. Specify any query string for your request, using the QUERYSTRING and QUERYSTRLEN options.

5. Specify any entity body for the HTTP request, and its length.

6. Specify the media type for any entity body you are providing, using the MEDIATYPE option. For requests with the POST and PUT methods, which require a body, you need to specify the MEDIATYPE option. For requests with other methods, where no body content is provided, the MEDIATYPE option is not required.

7. If this is the last request that you want to make to this server, specify CLOSE for the CLOSESTATUS option. XCICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. Specifying this option when you make your final request is good behavior, because the information in the headers means that the server can close its connection with you immediately after sending the final response, rather than waiting to see if you send further requests before timing out. The response from the server is still received and made available to your application. However, you will not be able to send any further requests to the server using this connection.

8. If you want to use chunked transfer-coding to send the request body as a series of chunks, follow the additional instructions in "Chunked transfer-coding". Note: Chunked transfer-coding is not supported with:

- Servers below HTTP/1.1.
- The WEB CONVERSE command.
- XCICS documents (the DOCTOKEN option).

XCICS assembles the request line, HTTP headers and request body, and sends the request to the server.

Receiving an HTTP response

The WEB RECEIVE command or the WEB CONVERSE command is used to receive the response from the server. (The WEB CONVERSE command combines the functions of the WEB SEND and WEB RECEIVE commands.) The WEB READ HTTPHEADER command or the HTTP header browsing commands are used to examine the headers.

The time that the application is prepared to wait to receive a response is indicated by the "cwi_timeout" region parameter specified on the xcics configuration file. The timeout limit does not apply to reading the headers of the response. When the period specified by cwi_timeout expires, XCICS returns a TIMEDOUT response to the application.

Using the WEB RECEIVE or WEB CONVERSE command:

1. Specify the session token for this connection, using the SESSTOKEN option.

2. Specify data areas to receive the HTTP status code sent by the server, and any text returned by the server to describe the status code. Note that the data is returned in its unescaped form.

3. Specify a data area to receive the media type of the response body.

4. Receive the response body by specifying either the INTO option (for a data buffer), or the SET option (for a pointer reference), and the LENGTH option. The data is returned in its escaped form, and converted into a code page suitable for the application, unless you request otherwise.

5. If you want to limit the amount of data received from the response body, specify the MAXLENGTH option. If you want to retain, rather than discarding, any data that exceeds this length, specify the NOTRUNCATE option as well. Any remaining data can be obtained using further WEB RECEIVE commands. If the data has been sent using chunked transfer-coding, XCICS assembles the chunks into a single message before passing it to the application, so the MAXLENGTH option applies to the total length of the entity body for the chunked message, rather than to each individual chunk.

6. Examine the HTTP headers of the server's response using the appropriate XCICS commands (READ HTTPHEADER and STARTBR/READNEXT,CLOSEBR HTTPHEADER).

7. Process the server's response and execute the application's business logic. If the response had a "normal" or informational status code, such as 200 (OK), you can process the response as normal. (The status code is received when you issue the WEB RECEIVE command.) If the response had a status code indicating an error or requesting further action, you should carry out alternative processing to account for this.
Closing the connection to an HTTP server

When XCICS is an HTTP client, the connection between XCICS and the server can be terminated by the server, or by XCICS following a command issued by the application program, or at end of task.

To manage connection closure effectively, the following behavior is recommended:

1. On the last request that you want to make to the server, specify CLOSE for the CLOSESTATUS option on the WEB SEND or WEB CONVERSE command. XCICS writes a Connection: close header on the request, or, for a server at HTTP/1.0 level, omits the Connection: Keep-Alive header. Specifying this option when you make your final request is good behavior, because the information in the headers means that the server can close its connection with you immediately after sending the final response, rather than waiting to see if you send further requests before timing out. The response from the server is still received and made available to your application. However, you will not be able to send any further requests to the server using this connection. Specifying the CLOSESTATUS option does not have the same range of effects as issuing the WEB CLOSE command.

2. If you need to test whether the server has requested termination of the connection, use the WEB READ HTTPHEADER command to look for the Connection: close header in the last message from the server. If the server terminates the connection, the application program cannot send any further requests using that connection, but it can receive responses that the server sent before it terminated the connection.

3. When all the HTTP requests and responses are completed, issue a WEB CLOSE command, specifying the session token. When the connection is closed, the session token that applies to it is no longer valid for use. The session token is required to receive a response from the server and to read the HTTP headers for the response, so the WEB CLOSE command should not be issued until you have completed all interaction with the server and with the last response that it sent. If the application program does not issue a WEB CLOSE command, the connection is closed at end of task.

3.2.5.8.2 Creating a URIMAP definition for an HTTP request by CICS as an HTTP client

This topic tells you how to create a URIMAP definition which specifies the components of the URI for an HTTP client request (scheme, host and path), and an SSL client certificate to be used with the request, if required. A URIMAP definition can be named on the WEB OPEN command, to provide a scheme and host name and a default path for the connection. It can also be named on a WEB SEND command, to provide a path for the relevant request. Alternatively, you can use the WEB EXTRACT URIMAP command to extract information from the URIMAP definition and use it directly in the application program that makes the HTTP client request.

1. Identify the URL that you plan to use for the HTTP client request. The URL represents a resource that you plan to access on a server.
2. Identify whether a client certificate might be required for the request, and obtain a suitable certificate label. If the scheme used for the request is HTTPS (HTTP with SSL), the server might request a SSL client certificate. If this happens, XCICS supplies the certificate label that is specified in the URIMAP definition.
3. Divide the URL for the request into its scheme, host and path components.
4. Begin a URIMAP definition with a name, with the "define urimap" statement in the XCICS configuration file.
5. Use the "enabled" attribute to specify whether the URIMAP definition should be installed in an enabled or disabled state.
6. Specify a "usage" attribute of "client" (XCICS as an HTTP client).
7. Specify the "scheme" attribute as the scheme component of the URL for the request. HTTP (without SSL) or HTTPS (with SSL) can be used. Do not include the delimiters :// following the scheme component.
8. Specify the "host" attribute as the host component of the URL for the request. Include the port number (prefixed by a colon) if it has been specified explicitly in the URL.
9. Specify the "path" attribute as the path component of the URL for the request. A wildcard (asterisk) cannot be used in a URIMAP definition for XCICS as an HTTP client.
10. Optional: If SSL is being used, specify the CERTIFICATE attribute as the label of the certificate that is to be used as the SSL client certificate for this request.
11. Optional: If SSL or TLS is being used, specify the CIPHERS attribute as the cipher code that is to be used for this request.
3.2.5.9 XCICS Web support and non-HTTP requests

You can use XCICS Web support to process inbound TCP/IP client requests which are not in the HTTP format. This facility is primarily intended to provide support for requests from user-written clients that use nonstandard request formats. The processing that takes place for requests, and the response that is provided, are defined by the user. No specific support is provided for any formally defined protocols which are used for client-server communication.

XCICS Web support only handles non-HTTP messages when XCICS is the server. Non-HTTP requests cannot be made by XCICS as a client. Client requests made through XCICS Web support use the HTTP protocol.

When XCICS Web support facilities are used for handling non-HTTP requests:

- You can use TCPIPSERVICE resource definitions to control the ports on which requests are received.
- You can use an analyzer program to assemble and parse requests, specify code page conversion, and determine subsequent request processing. You can code the analyzer program to parse requests in accordance with any request format that you have defined, but note that this XCICS facility does not provide specific support for any particular protocol for which a formal definition exists.
- You can use either Web-aware application programs, or non-Web-aware applications with a converter program, to provide responses to requests. Requests and responses can be handled using certain elements of the EXEC XCICS WEB programming interface, or passed between XCICS applications in a COMMAREA.

Some XCICS Web support facilities are not available for non-HTTP requests:

- Some of the facilities that help you interpret HTTP requests and construct the responses are not available. For example, message headers cannot be accessed separately.
- Persistent connections are not supported.
- URIMAP definitions are not used for non-HTTP requests.

3.2.5.9.1 Handling non-HTTP requests

To handle non-HTTP requests using XCICS Web support facilities, you need to code an analyzer program to determine processing for the requests, and application programs to provide responses to the requests. You also need to create some resource definitions.

The following components of XCICS Web support are used for processing non-HTTP requests:

- TCPIPSERVICE resource definitions.
- An analyzer program.
- Converter programs, if required.
- User-written application programs.
- An alias transaction for the application programs.

URIMAP definitions are not used with non-HTTP requests.

Processing for HTTP requests and processing for non-HTTP requests are kept separate. Non-HTTP requests are received using the USER protocol, specified on the TCPIPSERVICE definition. This ensures that XCICS can perform basic acceptance checks on HTTP requests and responses, and that non-HTTP requests are not subjected to these checks. The acceptance checks would produce an error response for non-HTTP requests and the request would not be processed.

TCPIPSERVICE resource definitions are needed for non-HTTP requests. TCPIPSERVICE resource definitions for non-HTTP requests must specify the USER (user-defined) protocol. URIMAP resource definitions are not used when requests are received through the USER protocol.

Analyzer programs and non-HTTP requests

An analyzer program is required for processing non-HTTP requests. It can reconstruct requests that have been divided up for transmission across the network, specify code page conversion of the requests, and perform any parsing that is required to determine subsequent request processing.
Reconstructing a non-HTTP request

An incoming request may be divided into several parts for transmission across the network. For non-HTTP requests, XCICS does not reconstruct the request before calling the analyzer program, and you should write your analyzer code accordingly.

On entry to the analyzer, the user_data pointer addresses a COMMAREA which contains the first part of the incoming request. To receive the next part of the request, set the return code to URP_EXCEPTION and the reason code to URP_RECEIVE_OUTSTANDING. XCICS Web support invokes the analyzer again, and the user_data pointer addresses the next part of the message. You can repeat this process as many times as you need to until the entire request has been received, up to the maximum supported length of 32767 bytes.

The results of this process are not visible to the XCICS WEB API commands. However, the reconstructed message can be passed to a converter program.

Determining non-HTTP request processing

The following input fields which relate to HTTP requests are undefined in an analyzer program for non-HTTP requests:

- The HTTP version
- The method
- The path component of the request
- The request headers

The subsequent processing stages must therefore be determined by examining the content of the request.

The analyzer program can specify subsequent request processing by a converter program, or by a Web-aware application program. "Writing an analyzer program" explains the inputs and outputs from an analyzer program, and how these are used to determine request processing.
3.2.6 Document handler guide

The XCICS document handler reproduces as close as possible the IBM CICS document handler. It allows you to build up formatted data areas, known as documents. Some examples of how these formatted areas, or documents, can be used, are:

- Constructing a COMMAREA.
- Sending HTML data to be displayed by a Web client.
- Creating standard formats for printing (for example, using your own letterhead, address, and so on).

The document handler is controlled by the EXEC CICS APIs:

- DOCUMENT CREATE
- DOCUMENT INSERT
- DOCUMENT RETRIEVE
- DOCUMENT SET
3.2.6.1 Creating a document

You can create an empty document using the DOCUMENT CREATE command, and then build the contents with subsequent DOCUMENT INSERT commands, or use DOCUMENT CREATE to create and build the document in one step. DOCUMENT CREATE has a mandatory DOCTOKEN parameter requiring a 16-byte data-area. The document handler domain uses the DOCTOKEN operand to return a token, which is used to identify the document on subsequent calls. The following example creates an empty document, and returns the token in the variable MYDOC:

```cicx
EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC)
```

To create a document with data, use the DOCUMENT CREATE command in any of the following ways:

- Using the BINARY option
- Using the TEXT option
- Using the FROMDOC option to copy an existing document
- Using the TEMPLATE option. “Setting up document templates”.

The BINARY option

Use this option to add to the document the contents of a data-area that must not undergo conversion to a client code page when the data is sent.

```cicx
EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC)
   BINARY(DATA-AREA)
```

The TEXT option

Use this option to add the specified contents to the document. For example, if you define a character string variable called DOCTEXT and initialise it to "This is an example of text to be added to a document", you can use the following command to create a document consisting of this text string:

```cicx
EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC)
   TEXT(DOCTEXT) LENGTH(53)
```

The FROMDOC option

To copy an existing document into a new document, you can use the DOCUMENT CREATE command with the FROMDOC option. The following example shows this:

```cicx
EXEC CICS DOCUMENT CREATE
   DOCTOKEN(MYDOC2)
   FROMDOC(MYDOC)
```

This results in two identical documents, each containing the text This is an example of text to be added to a document.
3.2.6.2 Setting up document templates

Document templates are portions of a document which can be created off-line, and inserted into the document in the application program. Document templates are XCICS resources, which you define using DOCTEMPLATE definitions in the XCICS configuration file; the name of the template is specified in the TEMPLATENAME attribute.

Templates can contain static data, and symbols whose values are inserted into the template when you issue the DOCUMENT CREATE or DOCUMENT INSERT command. The values to be substituted are specified in the application program; they are associated with a particular document and cannot be used in a different document.

Templates can be retrieved from common files on the UNIX/Linux file system.

Declaring doctemplates in the configuration file

DOCTEMPLATEs must be declared in the XCICS configuration file, with the "define doctemplate" statement. For each document template you must provide:

- the name
- the full path of the file containing the document template

Example:

```
define doctemplate name=MAIN, type=file, path=$HOME/doctemplates/main.html;
define doctemplate name=HOMEPAGE, type=file, path=$HOME/doctemplates/home_page.html;
```
3.2.6.3 Programming with documents and document templates

3.2.6.3.1 Symbols and symbol lists

Symbols represent variable data that is resolved at the time the template is added to the document - at the time the DOCUMENT CREATE or DOCUMENT INSERT is issued.

The application program needs to define values for the symbols that will be substituted when the template is used. These values can be defined on the DOCUMENT CREATE or the DOCUMENT SET commands. These commands both take a SYMBOLLIST operand which allows several symbols to be defined in a single command. You can also define individual symbols by using the SYMBOL and VALUE operands on the DOCUMENT SET command.

When you are planning your use of symbols and templates, note that:

1. When a template containing symbols has been inserted into a document, you cannot change the substituted values of those symbols in the document that is being composed. If you set different values for the symbols, the new values will be used the next time that the template is inserted into a document. Your changes will not affect the values that have already been inserted into the document.

2. If you insert a template before the symbols contained in it are set, the symbols will never be substituted. This can occur if you create a document from a template without specifying a symbol list.

3. Symbol substitution does not occur in HTML comments.

3.2.6.3.2 Setting symbol values

You can define values for symbols using the DOCUMENT SET command or the DOCUMENT CREATE command.

The DOCUMENT CREATE and DOCUMENT SET commands both take a SYMBOLLIST operand which allows several symbols to be defined in a single command. The SYMBOLLIST operand is a character string consisting of one or more definitions with single byte separators. By default, the symbol separator is an ampersand, but you can override this by using the DELIMITER option of the DOCUMENT SET or DOCUMENT CREATE commands. A definition for a symbol consists of a name, an equals sign, and a value. Here is an example of a symbol list:

```
title=New+Authors&author=Halliwell+Sutcliffe&editor=Stanley+Weyman
```

This example defines three symbols. The first symbol called "title" has the value 'New Authors'. The second symbol called "author" has the value 'Halliwell Sutcliffe' and the last symbol called "editor" has the value 'Stanley Weyman'. To adhere to the specification for the content type application/x-www-form-urlencoded, plus signs have been used to indicate spaces within the symbol values; they will be converted to spaces when the symbol values are put into the symbol table.

The following rules apply when setting symbols using a SYMBOLLIST:

- The name of each symbol must contain only uppercase and lowercase letters, numbers and the special characters dollar ('$'), underscore ('_'), hyphen ('-'), number sign ('#'), period ('.') and at sign ('@'). The name is case-sensitive, so uppercase letters are regarded as different from lowercase letters.

- The values in the symbol list can contain any characters. However, special coding is required if you need to include the following characters in symbol values in the symbol list:
  - The character that you have used as the symbol separator (which defaults to an ampersand, but can be overridden by use of the DELIMITER option).
  - The plus sign and the percent sign.

You can use the percent sign, followed by two characters that are hexadecimal digits (that is, 0-9, a-f, and A-F), to include characters such as these that have a special meaning. When the value is put into the symbol table, a percent sign and the two hexadecimal digits following it are interpreted as the EBCDIC equivalent of the single ASCII character denoted by the two digits. Some useful combinations are as follows:

- If you want a plus sign in the value in the symbol table, put %2B in the value in the symbol list.
- If you want a percent sign in the value in the symbol table, put %25 in the value in the symbol list.
- If you want an ampersand in the value in the symbol table, put %26 in the value in the symbol list.
- You can also use this coding to specify a space in the value in the symbol table—put %20 in the value
in your symbol list. Alternatively, you can use the space character or the plus sign, as described below.

If you prefer not to use this special coding, you can specify the UNESCAPED option on the DOCUMENT CREATE or DOCUMENT SET command. When you specify this option, no conversion takes place, and the symbol values are put into the symbol table exactly as you entered them. The example below shows you how to do this. However, the UNESCAPED option does not allow you to include the character that you have used as the symbol separator, within a symbol value in a symbol list. If you want to use the UNESCAPED option, choose a symbol separator that will never be used within a symbol value. Alternatively, you can use the SYMBOL and VALUE options on the DOCUMENT SET command to specify symbol values that contain the character you have used as the symbol separator, because the symbol separator has no special meaning when used in the VALUE option.

If you want to include spaces in a value, XCICS allows you to use the space character. However, to adhere more closely to the specification for the content type, it is suggested that you use a plus sign instead of a space character. The plus sign is interpreted as a space when the symbol value is put into the symbol table. However, you cannot use a plus sign to indicate a space character when the UNESCAPED option is used to prevent XCICS from unescaping symbol values contained in a symbol list or in the VALUE option. In these cases, you must use a space character to indicate a space, because plus signs are not converted to spaces when the UNESCAPED option is used.

Depending on your application, you might find that instead of specifying the exact list length for your symbol list each time you define values for the symbols, it is more convenient to choose a permanent value for the LISTLENGTH option, which will provide a fixed list length for your symbol list. The fixed list length that you choose should be long enough to include the maximum length of symbol list that you expect to supply. However, on those occasions when the fixed list length that you have specified for your symbol list is greater than the actual length of the symbols that you supply, you might find that there are trailing spaces or unpredictable characters in the value of the last symbol in the list. You can avoid this issue by including an extra dummy symbol at the end of your symbol list, such as &END=. Do not include this symbol in any templates or documents. Any trailing spaces or unpredictable characters will be assigned to the dummy symbol and will not appear in your documents, so you can continue to specify a list length that is greater than the actual length of the symbols.

As an alternative to using a symbol list, the DOCUMENT SET command allows you to set individual symbol values with the SYMBOL and VALUE options. The SYMBOL option specifies the name of the symbol, and the VALUE option specifies the value for that symbol. The rules for including spaces in a symbol value in a symbol list also apply to the VALUE option: you can use a simple space character or a plus sign, unless the UNESCAPED option of the DOCUMENT SET command has been specified, in which case you must use a space character. Also, the special coding that is required to include a plus sign or percent sign in symbol lists is similarly required in the VALUE option, unless the UNESCAPED option of the DOCUMENT SET command has been specified. However, ampersands, or any other character that you have specified as a symbol separator for the symbol list, have no special significance when used in the VALUE option.

### 3.2.6.3.3 Embedded template commands

The Document Handler recognises three commands that can be embedded in the template. These commands follow the syntax rules for Server Side Include commands. A Server Side Include command starts with the characters left angle bracket, exclamation mark, hyphen, hyphen, number sign followed by the command and it is terminated with the characters hyphen, hyphen, right angle bracket. For example:

```
<! --#command -->
```

The three commands that are supported are #set, #echo and #include.

#### #set

The #set command is used to set the values of symbols and is useful for setting up default values for symbols. The #set command in the template will be ignored if the symbol has already been given a value using the DOCUMENT SET command. If a previous #set command has been used to assign a value to the symbol, the value will be overridden. A symbol which has been assigned a value using the DOCUMENT SET command can only be changed by issuing another DOCUMENT SET command.

#### #echo

The #echo command identifies a symbol that must be substituted when the template is inserted into the document. The string containing the #echo command will be completely replaced by the value associated with the symbol. If no symbol has been defined with that name, the #echo command will remain in the output data.

An alternative method to using the #echo command is to specify the symbol name, preceding it with an ampersand and
terminating it with a semicolon. If we set a symbol called ASYM and give it a value of 'sample', the following two templates will give the same result after substitution.

Template 1:

This is an example template.
<---#set var=ASYM value='sample'-->
This is a <---#echo var=ASYM--> symbol.

Template 2:

This is an example template.
<---#set var=ASYM value='sample'-->
This is a &ASYM; symbol.

Result of substitution:

This is an example template.
This is a sample symbol.

#include

The #include command allows a template to be embedded within another template. Up to 32 levels of embedding are allowed.

For example:

<---#include template=templatename-->

where templatename is the name of the template (the 48 byte name) defined in the doctemplate definition. The template name can also be enclosed in double quotes.

3.2.6.3.4 Using templates

If you have created a template and defined it to XCICS, the following example shows how you can use the template to create the contents of a document. The following template is created and defined to XCICS with the name ASampleTemplate.

<---#set var=ASYM value='DFLTUSER'-->
This is a sample document which has been created by user
<---#echo var=ASYM-->.

In the application program, you can define a 48-byte variable called TEMPLATENAME and initialize it to a value of 'ASampleTemplate'. Once again you must define a 16-byte field for the document token (in this example, ATOKEN). You can then issue the command to create the document.

EXEC CICS DOCUMENT CREATE
   DOCTOKEN(ATOKEN)
   TEMPLATE(TEMPLATENAME)

This will result in a document being created with the content "This is a sample document which has been created by user DFLTUSER."

To change the symbol to another value, you can issue the DOCUMENT CREATE command with the SYMBOLLIST option:

EXEC CICS DOCUMENT CREATE
This will result in a document being created with the content "This is a sample document which has been created by user Ignazio Ignazi."

### 3.2.6.3.5 Document lifespan

Documents created by an application exist only for the length of the XCICS task in which they are created. This means that when the last program in the XCICS task returns control to XCICS, all documents created during the task’s lifetime are deleted. It is the application's responsibility to save a document before terminating if the document is going to be used in another task. You can obtain a copy of the document by using the DOCUMENT RETRIEVE. The application can then save this copy to a location of its choice, such as a temporary storage queue. The copy can then be used to recreate the document.

Documents retrieved with DOCUMENT RETRIEVE, contain control information, that is useful if an application wishes to recreate a copy of the original document.

It is also possible to issue a DOCUMENT RETRIEVE command and ask for the control information to be omitted. Specify the DATAONLY option on the DOCUMENT RETRIEVE command to instruct the Document Handler to return only the data. This is the case when an application needs the text of the document to use it directly (i.e. printing, emails, etc.)

### 3.2.6.3.6 Constructing a document

Once a document has been created, the contents can be extended by issuing one or more DOCUMENT INSERT commands. The options on the DOCUMENT INSERT command work in the same way as the equivalent commands on the DOCUMENT CREATE command.

The following sequence of commands shows an empty document being created followed by two INSERT commands:

```clisp
EXEC CICS DOCUMENT CREATE  
  DOCTOKEN(ATOKEN)
EXEC CICS DOCUMENT INSERT  
  DOCTOKEN(ATOKEN)  
  TEXT('Sample line 1. ')  
  LENGTH(15)
EXEC CICS DOCUMENT INSERT  
  DOCTOKEN(ATOKEN)  
  TEXT('Sample line 2. ')  
  LENGTH(15)
```

The document resulting from the above commands will contain:

```
Sample line 1. Sample line 2.
```

You can use the DOCUMENT RETRIEVE and DOCUMENT INSERT commands to insert a whole document into an existing document. The following variables must first be defined and initialized in the application program:

- A 16-byte field RTOKEN which contains the document token of the document to be retrieved
- A buffer DOCBUF of sufficient length to hold the retrieved document
- A fullword binary field called RETRIEVLEN to hold the length of the data retrieved
- A fullword binary field called MAXLEN to hold the maximum amount of data the buffer can receive, i.e. the length of DOCBUF
• A 16-byte field ITOKEN which contains the document token of the document that is being inserted into.

The following sequence of commands shows a document indicated by RTOKEN being inserted into another document indicated by ITOKEN:

```cics
EXEC CICS DOCUMENT RETRIEVE
   DOCTOKEN(RTOKEN)
   INTO(DOCBUF)
   LENGTH(RETRIEVLEN)
   MAXLENGTH(MAXLEN)

EXEC CICS DOCUMENT INSERT
   DOCTOKEN(ITOKEN)
   FROM(DOCBUF)
   LENGTH(RETRIEVLEN)
```

The retrieved document is inserted at the end of the document specified in the DOCUMENT INSERT command, and all the control information of the retrieved document will be present in the second document. The LENGTH parameter of the DOCUMENT INSERT command must be equal to the value returned from the DOCUMENT RETRIEVE command into the field RETRIEVLEN.

The DOCUMENT INSERT command allows an operand called SYMBOL to be used to add blocks of data to the document. SYMBOL must contain the name of a valid symbol whose value has been set. The Document Handler inserts the value that is associated with the symbol into the document.

Note that when a value associated with a symbol has been inserted into a document, you cannot change that value in the document that is being composed. If you set a different value for the symbol, the new value will be used the next time that symbol is inserted into a document. Your change will not affect the value that has already been inserted into the document.

### 3.2.6.3.7 Bookmarks

The sequence in which an application inserts data into a document might not reflect the desired sequence that the data should appear in the document. Bookmarks allow the application to insert blocks of data in any order yet still control the sequence of the data in the document. A bookmark is a label that the application inserts between blocks of data. Note: a bookmark cannot be inserted in the middle of a block of data.

The following example creates a document with two blocks of text and a bookmark:

```cics
EXEC CICS DOCUMENT CREATE
   DOCTOKEN(ATOKEN)
   TEXT('Pre-bookmark text. ')
   LENGTH(19)

EXEC CICS DOCUMENT INSERT
   DOCTOKEN(ATOKEN)
   BOOKMARK('ABookmark ')

EXEC CICS DOCUMENT INSERT
   DOCTOKEN(ATOKEN)
   TEXT('Post-bookmark text. ')
   LENGTH(20)
```

The document will now contain:

```
Pre-bookmark text. <ABookmark>Post-bookmark text.
```

Note that the text `<ABookmark>` does not appear in the document content but serves merely as a pointer to that position in the document. To add data to this document, you can insert text at the bookmark as follows:
EXEC CICS DOCUMENT INSERT
  DOCTOKEN ATOKEN
  TEXT('Inserted at a bookmark. '
  LENGTH(25)
  AT('ABookie mark')

Logically, the data of the document will contain the following (Note that in this instance, only the data is being shown and not the position of the bookmark).

Pre-bookmark text. Inserted at a bookmark. Post-bookmark text.

If the AT option is omitted, the data is always appended to the end of the document. A special bookmark of 'TOP' can be used to insert data at the top of the document, making it unnecessary to define a bookmark which will mark the top of the document.
3.2.7 TCP/IP Sockets Programming Guide

Communication throughout these networks has often been based on the Systems Network Architecture (SNA) family of protocols. XCICS TCP/IP offers XCICS users an alternative to SNA, the TCP/IP family of protocols for those users whose native communications protocol is TCP/IP.

Of course on Unix/Linux systems the socket APIs calls are the best way to interact with the TCP/IP protocol but, in order to support application coming from the mainframe as well as to make easier the life of COBOL and PL/I programmer, XCICS/TS supports the IBM EZASOKET API.

Applications running on XCICS/TS may communicate over TCP/IP with both direct socket APIs and EZASOKET APIs.
3.2.7.1 EZASOKET APIs


The APIs currently supported are:

- INITAPI
- ACCEPT
- BIND
- CLOSE
- CONNECT
- GETCLIENTID
- GETHOSTBYNAME
- GETHOSTBYADDR
- GIVESOCKET
- LISTEN
- SELECT
- SOCKET
- READ
- RECV
- RECVFROM
- SEND
- SENDTO
- TAKESOCKET
- WRITE

Samples

XFRAME distribution contains some sample programs that demonstrate the usage of EZASOKET calls. They may be found in the $XFRAMEHOME/samples/xcics/ezasoket directory.

Differences

There are only few differences between IBM CICS and XCICS/TS implementation of EZASOKET. They involve:

- INITAPI
- TAKESOCKET and GIVESOCKET
- SELECT calls between GIVESOCKET and CLOSE calls

INITAPI

INITAPI call has no effect on XCICS/TS: it may be coded in the program but it causes no effect.
GIVESOCKET and TAKESOCKET

With GIVESOCKET and TAKESOCKET calls, the socket is passed between two "peer" processes, via the XCICS core process (main); the socket transfer is implemented with a proprietary mechanism, therefore programmers must only be aware of the following small differences while coding of application:

- there is no need to call the GETCLIENTID
- the socket identifier to transfer from the listener program to the server program is the return code of the GIVESOCKET call

This example shows the difference.

In the listener program:

```plaintext
CALL EZASOCKET USING EZASOCKET-GIVESOCKET
         CHILD-SOCKET
         GIVESOCKET-CLIENT
         EZASOCKET-ERRNO
         EZASOCKET-RETCODE.
IF EZASOCKET-RETCODE < 0
   PERFORM ERROR-HANDLING
END-IF
MOVE EZASOCKET-RETCODE TO COMMAREA-SOCKET-ID
EXEC CICS START TRANSID('esrv')
   FROM(COMMAREA)
   LENGTH(LENGTH OF COMMAREA)
END-EXEC.
```

In the server program:

```plaintext
EXEC CICS RETRIEVE INTO(COMMAREA) LENGTH(CALEN)
END-EXEC
CALL EZASOCKET USING EZASOCKET-TAKESOCKET
         COMMAREA-SOCKET-ID
         TAKESOCKET-CLIENT
         EZASOCKET-ERRNO
         EZASOCKET-RETCODE.
IF EZASOCKET-RETCODE < 0
   PERFORM ERROR-HANDLING
END-IF
MOVE EZASOCKET-RETCODE TO CLIENT-SOCKET
```

SELECT between GIVESOCKET and CLOSE

As explained before, the GIVESOCKET passes the socket descriptor to another process through the core process, which is activated in parallel. There is therefore no longer need for the listener program to monitor the socket the wait for the other task to perform the GIVESOCKET.

These calls to the SELECT should be removed.
3.2.7.2 The EZASOCKET listener

Client applications can communicate with a remote XCICS region using sockets, either via direct socket API calls and EZASOCKET APIs.

Users may write their own socket listener, or take advantage of the listeners supplied by XCICS/TS: They emulate the behaviour of the IBM EZASOKET Listener provided by IBM and may be used to activate and communicate with a transaction from a remote application.

In order to enable activate the predefined socket listeners, a TCP service definition must be added to the XCICS/TS configuration file (see TCP Server options). XCICS/TS will provide to automatically start up the defined TCP service(s). Each TCP service instance will be listening on the pre-defined port for incoming requests, in order to forward them to the transaction server, where requests will be serviced.

The work mode of a TCP listener is specified in its configuration in the XCICS configuration file. The Direct Link mode is the default.

Communication flow

When a socket client program attempts to connect to the well-known port, the TCP service process issues an accept() call to accept the connection, which returns a new file descriptor that represents the socket connection to the client program. The new file descriptor is saved in a connection structure to be passed to the requested server transaction.

Once started, the server transaction, which is the program who performs the data exchange and elaboration, communicates over the socket using the EZASOKET APIs or directly with Unix/Linux socket APIs.

After accepting the connection, the TCP Server acts differently depending on the working mode.

For further information about the EZASOKET APIs, please refer to the XCICS Programming Guide.

Classic mode

Once the connection is established, TCP Server reads incoming data from the socket. This message must conform to the input data structure (defined by IBM in CICS TCP/IP for MVS), described in the following paragraph.

The TCP Server prepares an area according to the output data structure described below and, depending on the startup mode specified in the message sent by the client, it performs a START TRANSID or a WRITEQ TD, providing the output area. After thatTCP Server issues a GIVESOCKET command on the socket, and close it.

When the user requested transaction is started from the Task or ATI service, it must read the data respectively using a RETRIEVE or a READQ TD. Then it must acquire the socket connected to the client issuing a TAKESOCKET call on the GIVE-TAKE-SOCKET field of retrieved area.

From this point on, the server and client programs can continue to exchange data using the file descriptor returned by the TAKESOCKET call.

Server program can use both EZASOKET calls and/or standard UNIX socket APIs.

When running in Classic mode, TCP Server acts like the default listener provided by IBM, in the CICS TCP/IP Socket Interface.

Notes

Classic mode requires GIVESOCKET and TAKESOCKET support. These commands are not supported by the standard TCP API of UNIX/Linux systems, so XCICS must simulate such operations. In order to do that, XCICS defines a work table in a shared memory segment, which size is determined by the give_takesocket_table_size parameter, specified in the configuration file. If a classic mode server is defined and no value is provided for the give_takesocket_table_size, XCICS automatically allocates a 32 elements table.

Direct Link mode

Once the connection is established, TCP Server immediately starts a new transaction, CSAL, by issuing the START TRANSID command.
The new CSAL transaction, also known as the LOADER transaction, acquire the socket file descriptor, while the TCP server closes it. The connection continues to exist because LOADER transaction still has it open.

LOADER transaction calls recv(), which reads the message sent by the client. This message must conform to the input data structure (defined by IBM in CICS TCP/IP for MVS), described in the following paragraph.

After the message is received and parsed, the LOADER looks for the programs corresponding to the user requested transaction and LINKs it, providing data into the COMMAREA, according to the output data structure described below.

This gives the transaction access to the user data received, as well as the socket and the remote address of the connected peer.

From this point on, the server and client programs can continue to exchange data using the file descriptor, contained in the GIVE-TAKE_SOCKET field of the COMMAREA, that represents the socket connection.

To accomplish this, the server program can use both EZASOKET calls and/or standard UNIX socket APIs. Because the latter is a C language library, the user’s server program must code the COBOL CALL statements to pass the arguments using the C language conventions.

Notes

Direct Link mode only supports the immediate remote LINK of programs related to requested transactions, in the transaction named CSAL. The startup option specified in the input data structure is ignored.

This mode offers the advantage to speed up the activation of the serving programs and to avoid for the TCP Server to wait for the first user data message.

The server program has no need to issue a TAKESOCKET call, because the value contained in the field GIVE-TAKE-SOCKET, already represents the client socket FD.

Communication structures

Input data structure

The listening process requires the remote user application to send the first transmission in a prescribed format. After the first transmission, the user should wait for a response before sending any subsequent transmissions.

The default (IBM standard) format for the first transmission (client --> server) is

```
TRANID[,User-Data][,XX[,HHMMSS]]
```

where

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANID</td>
<td>Transaction (or TD queue) identifier, 1 to 4 characters, which must exist in PCT (or DCT)</td>
</tr>
<tr>
<td>User-data</td>
<td>optional user data</td>
</tr>
<tr>
<td>XX</td>
<td>optional startup mode: IC or ic - Interval control</td>
</tr>
<tr>
<td></td>
<td>TD or td - Transient Data</td>
</tr>
<tr>
<td>HHMMSS</td>
<td>Optional field used for hours, minutes and seconds for the interval time when the transaction is started using Interval Control.</td>
</tr>
</tbody>
</table>

Output data structure

TCP Server provides the server user program with an area formatted as follow:
**Java TCP Connector APIs**

The class `com.hite.connectors.TcpConnector` is an user-friendly and ready-to-use, client API interface for the TCP Server. It allows to immediately inter-operate with TCP Server in both direct link and classic mode.

For further information please refer to XJE java APIs.

**Classic mode Sample**

In the following sample the client program (Java language) connects a Classic TCP Server, and activates three times the same program using:

- Interval Control
- Interval Control with a delay of 2 seconds
- Transient Data

Server program (SOCKTEST.pre) sends back the received message.

**XCICS configuration entries**

```plaintext
add tcpservice name=LISTENER, port=8093, classic;
define program name=SOCKTEST;
define transaction code=SOCK, program=SOCKTEST, twa=512, protection=01;
define destination name=TDSK, type=intrapartitioned, reuse=yes, trigger=1,
transid=SOCK;
```

**Server program**

```plaintext
IDENTIFICATION DIVISION.
PROGRAM-ID.   SOCKTEST.
```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 SOCKET-FUNCTIONS.
   02 SOCKET-ACCEPT PIC X(16) VALUE 'ACCEPT ',.
   02 SOCKET-BIND PIC X(16) VALUE 'BIND ',.
   02 SOCKET-CLOSE PIC X(16) VALUE 'CLOSE ',.
   02 SOCKET-CONNECT PIC X(16) VALUE 'CONNECT ',.
   02 SOCKET-FCNML PIC X(16) VALUE 'FCNML ',.
   02 SOCKET-GETCLIENTID PIC X(16) VALUE 'GETCLIENTID ',.
   02 SOCKET-GETHOSTBYADDR PIC X(16) VALUE 'GETHOSTBYADDR ',.
   02 SOCKET-GETHOSTBYNAME PIC X(16) VALUE 'GETHOSTBYNAME ',.
   02 SOCKET-GETHOSTID PIC X(16) VALUE 'GETHOSTID ',.
   02 SOCKET-GETHOSTNAME PIC X(16) VALUE 'GETHOSTNAME ',.
   02 SOCKET-GETPEERNAME PIC X(16) VALUE 'GETPEERNAME ',.
   02 SOCKET-GETSOCKNAME PIC X(16) VALUE 'GETSOCKNAME ',.
   02 SOCKET-GETSOCKOPT PIC X(16) VALUE 'GETSOCKOPT ',.
   02 SOCKET-GIVESOCKET PIC X(16) VALUE 'GIVESOCKET ',.
   02 SOCKET-INITAPI PIC X(16) VALUE 'INITAPI ',.
   02 SOCKET-IOCTL PIC X(16) VALUE 'IOCTL ',.
   02 SOCKET-LISTEN PIC X(16) VALUE 'LISTEN ',.
   02 SOCKET-READ PIC X(16) VALUE 'READ ',.
   02 SOCKET-RECV PIC X(16) VALUE 'RECV ',.
   02 SOCKET-RECVFROM PIC X(16) VALUE 'RECVFROM ',.
   02 SOCKET-SELECT PIC X(16) VALUE 'SELECT ',.
   02 SOCKET-SEND PIC X(16) VALUE 'SEND ',.
   02 SOCKET-SENDTO PIC X(16) VALUE 'SENDTO ',.
   02 SOCKET-SHUTDOWN PIC X(16) VALUE 'SHUTDOWN ',.
   02 SOCKET-SOCKET PIC X(16) VALUE 'SOCKET ',.
   02 SOCKET-TAKESOCKET PIC X(16) VALUE 'TAKESOCKET ',.
   02 SOCKET-TERMAPI PIC X(16) VALUE 'TERMAPI ',.
   02 SOCKET-WRITE PIC X(16) VALUE 'WRITE ',.
01 CLIENTID-LSTN.
   05 CID-DOMAIN-LSTN PIC 9(8) COMP.
   05 CID-NAME-LSTN PIC X(8).
   05 CID-SUBTASKNAME-LSTN PIC X(8).
   05 CID-RES-LSTN PIC X(20).
01 TCPSOCKET-PARM.
   05 GIVESOCKET PIC 9(8) COMP.
   05 LISTN-NAME PIC X(8).
   05 LISTN-SUBNAME PIC X(8).
   05 CLIENT-IN-NAME PIC X(36).
   05 SOCKADDR-IN-PARM.
      15 SIN-FAMILY PIC 9(4) COMP.
      15 SIN-PORT PIC 9(4) COMP.
      15 SIN-ADDRESS PIC 9(8) COMP.
      15 SIN-ZERO PIC X(8).
01 CONSOLE-MSG PIC X(80) VALUE SPACES.
01 REPLY-MSG PIC X(80) VALUE SPACES.
01 QNAME PIC X(4) VALUE SPACE.
01 RC PIC S9(8).
01 FLAGS PIC 9(8) BINARY.
01 NFBYTES PIC 9(8) BINARY.
01 CLCLEN PIC S9(4) COMP.
01 SOCKID PIC S9(4) COMP.
01 ERROR PIC 9(8) COMP.
01 RETCODE PIC S9(8) COMP.
01 START-CODE PIC X(2) VALUE SPACES.
01 SOCK-INTERFACE PIC X(3).
88 API-INTERFACE VALUE "API".
88 EZA-INTERFACE VALUE "EZA".

LINKAGE SECTION
01 DFHCOMMAREA PIC X(72).

PROCEDURE DIVISION
MAIN SECTION
MOVE "EZA" TO SOCK-INTERFACE
EXEC CICS ASSIGN STARTCODE(START-CODE) END-EXEC.
EVALUATE START-CODE
WHEN "SI"
   PERFORM RETRIEVE-INTO
WHEN "QI"
   PERFORM READQ-TD
WHEN OTHER
   MOVE "START-CODE UNRECOGNIZED" TO CONSOLE-MSG
   PERFORM ERROR-Routine
END-EVALUATE.
PERFORM TAKE-SOCKET.

STRING "TRX SAYS: YOU SENT ME" DELIMITED BY SIZE
   CLIENT-IN-DATA DELIMITED BY SIZE
   "" INTO REPLY-MSG.

IF API-INTERFACE
   CALL "send" USING
      BY VALUE GIVE-TAKE-SOCKET
      BY REFERENCE REPLY-MSG
      BY VALUE LENGTH OF REPLY-MSG
      BY VALUE ZERO
   MOVE RETURN-CODE TO RC
   IF RETURN-CODE < 0
      MOVE "ERROR SENDING DATA" TO CONSOLE-MSG
      PERFORM ERROR-Routine
   END-IF
   CALL "close" USING BY VALUE GIVE-TAKE-SOCKET
ELSE
   MOVE LENGTH OF REPLY-MSG TO NBYTES
   MOVE ZERO TO FLAGS
   CALL 'EZASOKET' USING SOCKET-SEND
      SOCKETID
      FLAGS
      NBYTES
      REPLY-MSG
      ERNO
      RETCODE
   IF RETCODE < 0
      MOVE "ERROR SENDING DATA WITH EZASOKET" TO CONSOLE-MSG
      PERFORM ERROR-Routine
   END-IF
   CALL 'EZASOKET' USING SOCKET-CLOSE
      SOCKETID
      ERNO
      RETCODE
END-IF
EXEC CICS RETURN END-EXEC.
GOBACK.
RETRIEVE-INTO
MOVE LENGTH OF TCPSOCKET-PARM TO CLENG
EXEC CICS RETRIEVE INTO TCPSOCKET-PARM
LENGTH(CLENG)
END-EXEC.
EXIT.
Client program

```java
/**
 * TestSock
 * *
 * compile:
 * javac TestSock.java
 * *
 * usage:
 * java TestSock *
 * *
 */
import java.util.*;
import java.net.*;
import java.io.*;
public class TestSock {
    public static void main(String[] argv) {
        String hostname=argv[0];
        int port=Integer.parseInt(argv[1]);
        new TestSock().runTests(hostname, port);
    }

    public void runTests(String h, int p) {
        run(h, p, "SOCK, USERDATA IS HERE!");
        run(h, p, "SOCK, USERDATA IS HERE, TE");
    }

    public void run(String hostname, int port, String trx) {
        try {
            ...
        } catch (Exception e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    }
}
```
Direct Link mode Sample

In the following sample the client program (C language) connects a Direct Link TCP Server, activates the program related to task named TS00 and waits for data. Server program (TS00.pre) sends back date and time.

XCICS configuration entries

```
add tcpserver name=DEFAULT, port=8023;
define transaction code=TS00, program=TS00, twa=512, protection=64;
define program name=TS00;
```

Server program

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TS00.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 ABS-TIME PIC 9(15) COMP=3.
01 WTIME PIC X(6).
01 WDATE PIC X(9).
01 WSBUF PIC X(80).
LINEAGE SECTION.
01 DECKMARCA.
05 GAVE-TAKE-PORT PIC 9(8) COMP.
05 LIST-NAME PIC X(8).
05 LIST-NAME PIC X(8).
05 CLIENT-IN-NAME PIC X(36).
05 SOCKADDR-IN-NAME.
15 SIN-FAMILY PIC 9(4) COMP.
15 SIN-PORT PIC 9(4) COMP.
15 SIN-ADDRESS PIC 9(8) COMP.
15 SIN-ZERO PIC X(8).
PROCEDURE DIVISION.
EXEC CICS ASKTIME ABSTIME(ABS-TIME) END-EXEC.
EXEC CICS FORMATTIME
   ABSTIME(ABS-TIME)
   YYYYMMDD(XDATE)
   WTIME(WTIME)
END-EXEC.
```
Client program

```c
/** *
 * testsock.c
 *
 * usage:
 *   testsock <hostname> <port> <transactioncode>
 */
#include<stdio.h>
#include<stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <netdb.h>
int main(int argc, char **argv) {
    int sock;
    int rc;
    char *host_name;
    int port;
    struct sockaddr_in saDestAddr;
    struct hostent *pHost;
    char buffer[256];
    char *transaction_name;
    if (argc<4) {
        fprintf(stderr,"usage: \n");
        fprintf(stderr," %s hostname port transaction\n", argv[0]);
        exit(1);
    }
    host_name=argv[1];
    port=atoi(argv[2]);
    transaction_name=argv[3];
    printf("starting transaction %s on host %s at port\n", 
          transaction_name,
          host_name,
          port);
    memset(&saDestAddr,0,sizeof(struct sockaddr));
    saDestAddr.sin_family=AF_INET;
    saDestAddr.sin_port=htons(port);
    pHost=gethostbyname(host_name);
    if (pHost==NULL) {
        perror("gethostbyname");
        exit(1);
    }
    saDestAddr.sin_addr.s_addr=((struct in_addr *)(pHost->h_addr))->s_addr;
    sock=socket(AF_INET, SOCK_STREAM, 0);
    if (sock<0) {
        perror("socket");
    }
```
exit(1);

rc=connect(sock, (struct sockaddr *)&saDestAddr, sizeof(struct sockaddr_in));
if (rc!=0) {
  perror("connect");
  close(sock);
  exit(1);
}
strcpy(buffer, transaction_name);
rc=send(sock, buffer, strlen(buffer), 0);
if (rc<0) {
  perror("send");
  close(sock);
  exit(1);
}
rc=recv(sock, buffer, 80, 0);
if (rc<0) {
  perror("recv");
  close(sock);
  exit(1);
}
printf("received %d bytes\n", rc);
printf("[%s]\n", buffer);
close(sock);
exit(0);
3.2.8 XECI APIs Guide

XECI (XCICS External Call Interface) is a programming interface that allows to external applications to invoke programs running under XCICS/TS.

Programs invoked with XECI must be commarea based programs: they receive the input data from the remote application, and return their output in the COMMAREA only. They cannot make use of BMS facility.

XECI is composed by two parts: server and client. The server side is a task running on XCICS/TS and acting as gateway for the requests incoming from the clients. The client side provides the APIs that must be called by the remote application to invoke the program running on XCICS/TS. This part of the software connect the server, will forward the request and receive the response.

XECI emulates many aspects of the IBM ECI interface and may be used to replace both ECI and EXCI calls.

XECI client is available for Unix, Linux and Windows and it has entry points callable by C, C++ and COBOL.
3.2.8.1 Developing applications

The programmers that want to use XECI to communicate with server programs must apply the following guidelines.

Each time the programmer wants to call the remote program, it must invoke one of the XECI functions (CICSEXTERNALCALL or xeci_remote_link), providing the input parameters in the I/O data structure. Once the server program replied, the XECI function will return the control to the caller, which have to analyze to output parameters in the I/O structure.

The programmer identifies the server system by an 8 bytes identifier which is dynamically mapped to the corresponding internet address (IP address + port) where the XECI server is listening. The mapping between the 8 bytes name and the full address is configured in the XECI configuration file.

**Function names**

XECI functions may be invoked by `xeci_remote_link` or by its alias `CICSEXTERNALCALL`. Both these functions have the same behaviour and the same call interface.

**Parameters**

The I/O data area is described in by the C structure `xeci_parms` defined in the header file "xeci.h", and by the COBOL area ECI-PARMS, defined in the copybook "XCICSECY.cpy". Both these files are located in $XFRAMEHOME/utils.

When invoking the remote program at least 2 information must be provided in the I/O area:

- 8 bytes identifier of the server region
- 8 bytes identifier of the server program

If a COMMAREA must be exchanged with the server program, you must also provide:

- length of the communication area
- address of the communication area

**Sample calls**

This is the sample of a client program written in COBOL that invokes a remote program:

```cobol
move low-values to eci-parms
move "XECISRV" to eci-program-name
move "XCICSSRV" to eci-system-name
move "TPNX" to eci-tpn
move "hi! I'm the client" to commarea
move length of commarea to eci-commarea-length
set eci-commarea to address of commarea
call "CICSEXTERNALCALL" using by reference eci-parms
                        returning eci-error-id.
```

or in C language:

```c
memset(&parms, 0x00, sizeof(struct xeci_parms));
memcpy(parms.eci_program_name, "XECISRV ", 8);
memcpy(parms.eci_system_name, "XCICSSRV ", 8);
memcpy(parms.eci_tpns, "TPNX", 4);
strncpy(commarea, "HI! I'm the C client program!");
parms.eci_commarea=commarea;
parms.eci_commarea_length=256;
r=xeci_remote_link(&parms);
```

Complete XECI samples may be found in the directory $XFRAMEHOME/samples/xcics/xeci.
Building client applications

Client applications that use XECI must be compiled including the XECI structures definition: xeci.h for C/C++ and XCICSECI.cpy for COBOL. Both of them are in the directory $XFRAMEHOME/utils. C sources must also include the file "xport.h" (in $XFRAMEHOME/include).

Once compiled, they have to be linked with the XECI shared library (libxeci.so/sldll).

Programs running in the XFRAME environment may take advantage of the dynamic loading features of the XFRAME batch runtime: you simply have to set up the dynamic loading of the shared loading before to start the runtime. I.e.

```bash
export XRUN_LIBRARIES=$XFRAMEHOME/lib/libxeci.so
xrun MYCLIENT
```

If your program does not use XFRAME facilities, you have to specify the XECI library at link time.

I.e.

```bash
include $(XFRAMEHOME)/include/xport.make

COMPILE=$(CC) -I$(XFRAMEHOME)/include -I$(XFRAMEHOME)/utils
all: xeciclt

xeciclt: xeciclt.c
    $(COMPILE) -o xeciclt -L$(XFRAMEHOME)/lib -lexci xeciclt.c
```
3.2.8.2 Configuration

**Configuring the server side**

The server side of XECI must be set up in the XCICS/TS configuration file adding one or more TCP services configured for the XECI protocol.

Each TCP server waits for connection on a pre-defined port. This port, together with the host address, will be referenced in the client application, to identify the XECI server.

```plaintext
add tcpserver name=SERVXECI, port=8077, eci;
add tcpserver name=BCKPXECI, port=8099, eci;
```

**Configuring the client side**

As described in the previous chapter, applications using XECI, identify the server region with a 8 bytes identifier. This name must be mapped to the corresponding server address and port.

The XECI configuration file (xeci.conf) contains the mappings, according to the following syntax:

```plaintext
<NAME>=<hostname>:<port>
```

More than one mapping may be specified in the xeci.conf. Lines beginning with "#" are handled as remark.

I.e.

```plaintext
# xeci.conf
#
APPLLX01=linux01.ht.net:8077
APPLHP02=192.168.1.23:9999
```

The XECI client library searches by default the configuration file xeci.conf in the paths specified below. The environment variable XECI_CONFIGURATION_FILE may be used to set a different path for the configuration file. It must be set and exported before to start the client application.

```plaintext
export XECI_CONFIGURATION_FILE=/my/path/xecli.conf
```

**Unix/Linux**

The file xeci.conf is located by default in the directory $HOME/etc.

**Windows**

The file xeci.conf must is located by default in the directory %HOMEDRIVE%\%HOMEDIR%. 
3.2.9 XJGATE APIs Guide

XJGATE is a set of APIs that allow a program to connect XCICS over its native terminal protocol, in order to activate transactions and to exchange data with them.

XJGATE APIs are available both for the following programming languages:

- C language
- Java.

The use of the XJGATE APIs is discouraged and maintained only for backward compatibility reasons. Programmers that want to communicate with programs running under XCICS/TS should consider the opportunity to use other communication protocols like SNA ISC, TCP/IP sockets or web services.

The gateway transaction

In order to provide an easier gatewaying interface a gateway program has been provided: XJGT.

To activate XJGT, XCICS must be configured with the following entries.

```
define program name=XJGT, language=java, path=com.hite.xjgate.XJGateServer;
define transaction name=XJGT, program=XJGT, twa=512, protection=64;
```

C language APIs

Include files

In order to use XJGATE APIs the following files must be include in the C source:

- xascebc.h
- socklib.h
- xjgate.h

The main source must define the symbol NOEXTERN

1.e.

```
#define NOEXTERN
#include "xascebc.h"
#include "socklib.h"
#include "xjgate.h"
```

Functions

xjgate_connect

prototype

```
struct Connection *xjgate_connect(char *hostname, int port, char *termname, char *cbldbg, char *asmdbg, int *rc);
```

description

Connects the application.
return value

Returns a structure identifying the connection. If failed NULL is returned and "rc" contains the error code.

Example

```c
struct Connection *conn;
conn=xjgate_connect(hostname, port, termname, NULL, NULL, &rc);
if (conn==NULL) {
    fprintf(stderr,"cannot connect (%d): ", rc);
    exit(1);
}
```

xjgate_attach

prototype

```c
int xjgate_attach(struct Connection *c) ;
```

description

reconnects XCICS engine. Must be issued before starting the data exchange.

return value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>successful</td>
</tr>
<tr>
<td>&lt;0</td>
<td>failed</td>
</tr>
</tbody>
</table>

Example

```c
rc=xjgate_attach(conn);
```

xjgate_detach

prototype

```c
int xjgate_detach(struct Connection *c) ;
```

description

disconnects XCICS engine. Must be issued at the end of the data exchange.

return value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>successful</td>
</tr>
<tr>
<td>&lt;0</td>
<td>failed</td>
</tr>
</tbody>
</table>

xjgate_send

prototype

```c
int xjgate_send(struct Connection *c, unsigned char *data, int len);
```
description
Send the specified data to XCICS for the specified len.

return value

<table>
<thead>
<tr>
<th>bytes sent</th>
<th>if successfull</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>if failed</td>
</tr>
</tbody>
</table>

xjgate_sendData

prototype

```
int xjgate_sendData(struct Connection *c, unsigned char *data, int len);
```

description
Send the specified data to XCICS for the specified len. Data are prefixed by 3 bytes for SNA communication.

return value

<table>
<thead>
<tr>
<th>bytes sent</th>
<th>if successfull</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>if failed</td>
</tr>
</tbody>
</table>

xjgate_receive

prototype

```
int xjgate_receive(struct Connection *c, unsigned char *data, int len);
```

description
Receives data from XCICS in the specified buffer.

return value

<table>
<thead>
<tr>
<th>bytes sent</th>
<th>if successfull</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>if failed</td>
</tr>
</tbody>
</table>

xjgate_checkProtocol

prototype

```
int xjgate_checkProtocol(unsigned char* buffer, int len);
```

description
Describes the dialogue status.

return value
current dialog status
**xjgate_close**

**prototype**

    void xjgate_close(struct Connection *c);

**description**

Closes the XCICS connection.

**xjgate_activate**

**prototype**

    void xjgate_activate(struct Connection *c, char *transid);

**description**

Activates the specified transaction.

**Example**

```c
xjgate_activate(conn, "SRV1");
```

**Compiling & Linking**

Programs compiled with XJGATE must use the following options:

```bash
-I $XFRAMEHOME/include -I $XFRAMEHOME/xsdf/cpy
```

and linked with:

```bash
-L $XFRAMEHOME/lib -lxsocket.a
```

```bash
cc -I $XFRAMEHOME/include -I $XFRAMEHOME/xsdf/cpy -L $XFRAMEHOME/lib -lxsocket.a -o pgm1 pgm1.c
```

**An example**

```c
/** Connects XCICS */
struct Connection *conn;
conn=xjgate_connect("hpux01.ht.net",4000,"L001",NULL,NULL,&rc);
if (conn==NULL) {
    fprintf(stderr,"cannot connect (%d):", rc);
    exit(1);
}

/**
 * connected!!
 * initialize dialogue session
 */
rc=xjgate_attach(conn);
if (rc<0) { exit(-1); }
```
```c
/**
 * starting the transid XJGT
 */
rc=xjgate_activate(conn, "XJGT");
if(rc<0) { exit(-1); }

/**
 * data reception loop
 */
while (receive) {
    /** receiving data ... */
    rc=xjgate_receive(conn, buffer, 10000);
    if(rc<0) { return rc; }
    /** checking the protocol command */
    proto=xjgate_checkProtocol(buffer, rc);
    switch (proto) {
        case XJGATE_DETACH_REQUEST:
            /**
             * XICS terminated transaction and
             * asks me to detach
             */
            xjgate_detach(conn);
            receive=FALSE;
            break;
        case XJGATE_RECV_REQUEST:
            /** XICS is ready to receive */
            receive=FALSE;
            break;
        case XJGATE_RECV_BUFFER:
        case XJGATE_TERMINAL_SETUP:
            /**
             * XICS is asking for a RECEIVE BUFFER or a REMOTE TERMINAL
             * SETUP
             */
            receive=FALSE;
            break;
        default:
            /** 3270 data arriving... */
            printf("received %d bytes\n",rc);
            break;
    }
}
strcpy(buffer,"DATA FOR TRANSACTION");
if (!UNPREFIXEDDATA) {
    xjgate_sendData(conn, ebcdic, strlen(buffer));
} else {
    int l=strlen(buffer);
    buffer[0]=0x1d;
    buffer[1]=0x0d;
    buffer[2]=0x0d;
    memcpy(buffer+3, ebcdic, l);
    xjgate_send(conn, ebcdic, strlen(buffer));
}
xjgate_close(conn);
```

**Java APIs**

XJGATE Java APIs may be found in the XJE Java API tree.
3.3 **XIMS IMS-DC compatibility toolkit**

The IMS/DC is an IBM product that serves as data communication monitor on many IBM mainframes. IMS/DC enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It also includes facilities for creating, operating and maintaining IMD/DB databases.

XIMS is a software toolkit which allows COBOL based applications developed to operate under IMS/DC, to run under XCICS Transaction Server on UNIX operating systems.

XCICS Transaction Server and H2R are prerequisites for XIMS toolkit.
3.3.1 Configuration

XIMS toolkit enables the IMS/DC application to run under XCICS TS. That means that XCICS must be correctly configured (please refer to XCICS TS documentation) to host XIMS.

Furthermore, in order to run properly an application, XIMS toolkit also must be correctly configured.

An XIMS application is defined as a set of transaction, programs, maps and other resources, grouped together to run under the control of a single XCICS main process.

XIMS configuration is provided in the same configuration as XCICS TS (xcics.conf).

First of all, when defining an XIMS application, the xcics.conf must include the following directives:

```
set use_xims=yes;
set use_dli=yes;
load library=$H2RHOME/lib/libdl1.sl;
load library=$H2RHOME/lib/libims.sl;
export IMS_FORMATS=<path>
```

Where the IMS_FORMATS paths to the directory where .msg and .fmt are stored (see fmt2bms).

And a default DBC named DL1 must be defined:

```
define dbc name=DL1, database=HTORA1, user=upim, password=upim;
bind dbc=DL1 default;
```

Parameters, resources and environment variables are specified using the define directive

Every directive is terminated using ?. Directive parameters are comma (?,?) separated. If not explicitly required no quotas (?) is required for values.

Lines started with ?#? are treated as remarks.
3.3.2 Transactions

To define an IMS transaction the directive define ims_transaction is used.

Syntax

```
define ims_transaction name=<value>,
    program=<program>,
    psb=<value>,
    spa=<value>;
```

name

This is the name of the transaction (Transaction Code).

program

This the name of the program associated to the specified transaction code.

psb

This is the name of related PSB.

spa

 Specifies the length in bytes of the Scratch-Pad Area (SPA).

Example

```
define ims_transaction name=CMDBATCH, program=Y09961, psb=Y09961, spa=0;
```
3.3.3 Terminals

As we know IMS/DC terminal names are 8 characters long, while CICS terminal names are 4 character long.

In order to return the expected terminal name to the user application a relation between XCICS terminal name and XIMS terminal name must be provided.

To define an IMS terminal relationship the directive `define ims_terminal` is used.

**Syntax**

```
define ims_terminal name=<value>, cics=<value>
  [type=(standard|printer)];
```

**name**

This is the name of the IMS/DC terminal as returned to the application.

**cics**

This the name of the real XCICS terminal (EIBTRMID)

**type**

Terminal type may be standard (default) or printer.

**Example**

```
define ims_terminal name=K02N1V01, cics=TN01;
define ims_terminal name=K09N1P03, cics=PRT0, type=printer;
```

**Maps**

IMS/DC formats does not have to be specified, but the mapsets generated by the format conversion (see fmt2bms) must be defined using the XCICS TS directive `define mapset`.

To define these mapset please refer to the XCICS TS documentation.
### 3.3.4 Formats

In order to run under XIMS, IMS/DC formats must be compiled using `fmt2bms`.

The tool `fmt2bms` generates:

- XIMS message files (.msg)
- XIMS format files (.fmt)
- BMS macro files (.bms)

By default fmt2bms produces its output in the current directory.

**Syntax**

```
fmt2bms [options] <ims format>
```

**Options**

- `-a, --auto`  automatically compiles bms modules
- `-b, --bms <path>`  specifies a path for generated bms file
- `-f, --fmt <path>`  specifies a path for generated fmt file
- `-D, --debug`  runs in debug mode
- `-h, --help`  show this help
3.3.5 Compiling

To compile IMS/DC user programs, the script `xcob` must be invoked, specifying the option `-m`.

Example

```
# xcob -m -s -ua USERPGM pre
```

Please refer to "xcob COBOL compiler script" in the XFRAME bookshelf, for further details.
3.4 XTND Tn3270 Server

XTND is a tn3270 server which allows standard tn3270 terminal emulators (i.e. IBM Personal Communicator (TM), Tn3270plus (TM), etc.) to connect XCICS applications.

**System requirements**

XTND is pure Java (TM) code so it may be run on every system supporting a Java Virtual Machine.

JRE 1.4 or higher is required.

**General concepts**

XTND purpose is to convert native XCICS protocol into tn3270 protocol. The XTND daemon can be run embedded in the XCICS region or as standalone service. When it is embedded in the region, it only serves the region itself. When it runs standalone it can be executed on the same system running the XCICS application or on another system; in this case each instance may handle one or more XCICS regions.

When a terminal emulator connects XTND a terminal identifier is assigned to the session and a connection with the defined XCICS application is established. From this moment on the terminal emulator starts working with XCICS.

The assigned terminal identifier corresponds to the physical name of a terminal entry defined in the XCICS TCT (define terminal).

**How XTND assigns terminal identifiers**

In order to grant a correct terminal identifier assignment, terminal entries must be defined. Each terminal identifier must be qualified by a terminal name, which is the device name of a TCT entry. Optionally it may be related to the following constraints:

<table>
<thead>
<tr>
<th>LU name</th>
<th>the LU name defined in the terminal emulator connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname/IP address</td>
<td>the hostname or IP address of the system where the terminal emulator runs</td>
</tr>
</tbody>
</table>

If LU name and/or Hostname/IP is defined for a terminal identifier, XTND checks if the incoming tn3270 connection satisfies this defined constraint. Therefore only the connections specifying the defined LU name and/or coming from the specified network address may be assigned to that terminal ID.

If terminal identifier is defined without any constraint may automatically assigned to every terminal emulator requiring a connection.

When a connection arrives, XTND performs the following steps:

1. looks for the first available terminal identifier which satisfies the constraints for the terminal: it searches for a terminal id bind to the incoming network address and/or to the incoming LU name, if the terminal emulator sent one
2. if not constraint-related identifier is found, it looks for the first available one without constraints
3. if no available term is detected, the connection is refused
4. if an available terminal id is found, XTND assigns it to the incoming connection and connects XCICS

**Multiple applications**

Standalone XTND instances may serve more than one application: by default terminals connect to application defined as default: if no application has the "default" parameter set to "true", the default application is the first one defined in the configuration file.

The switch from an application to another is controlled by the user programs, by means of the ISSUE PASS command.
**Multiplex mode**

XTND may run in two different modes:

- multithreaded
- multiplexed

In the first case each terminal connection is handled by a pool of thread, while in the second one there is one only thread handling all the connections.

Choosing between the two different modes depends on:

- JVM implementation
- system load
- memory availability

**Usage**

Standalone XTND services are controlled by the script `xtndctl`. Following commands are supported:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Starts up XTND daemon</td>
</tr>
<tr>
<td>stop</td>
<td>Stops XTND daemon</td>
</tr>
<tr>
<td>status</td>
<td>Shows the status of XTND and of the connected terminals</td>
</tr>
<tr>
<td>reset</td>
<td>resets the terminal specified as parm</td>
</tr>
<tr>
<td>tracelevel</td>
<td>Sets the XTND tracing level to the specified parm. Admitted values:</td>
</tr>
<tr>
<td></td>
<td>• none</td>
</tr>
<tr>
<td></td>
<td>• log</td>
</tr>
<tr>
<td></td>
<td>• debug</td>
</tr>
</tbody>
</table>

**Syntax**

```
xtndctl [options] {start|stop|status|tracelevel|reset} [parm]
```

**Options**

- `-d`    java vm debug mode
- `-f`    foreground mode
- `-l<file>` specifies alternate log name
- `-c<file>` xbm start with specified configuration file
3.4.1 Configuration

This section covers the configuration aspects of XTND run as standalone service. To configure region embedded services, please refer to the XCICS/TS Configuration Guide, in the section "TN3270 Services".

XTND is configured using an XML file named `xtnd.xml` normally located in `$HOME/etc` of the owning user.

The configuration file is divided into different sections:

- general settings
- application definition

Whole configuration is identified by the tag `<configuration>`.

**Settings element**

This section, identified by the tag `<settings>`, defines the global parameters for the daemon behaviour.

**Service element**

It defines the main behaviour of XTND. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>None</td>
<td>network address to be bind</td>
</tr>
<tr>
<td>port</td>
<td>23</td>
<td>the port listening for incoming tn3270 connections</td>
</tr>
<tr>
<td>welcome</td>
<td>true</td>
<td>if true a welcome screen is sent to the emulator</td>
</tr>
<tr>
<td>multiplex</td>
<td>false</td>
<td>if true multiplex mode is enabled</td>
</tr>
<tr>
<td>tn3270e</td>
<td>False</td>
<td>if true XTND will try to negotiate TN3270E protocol</td>
</tr>
<tr>
<td>timeout</td>
<td>0</td>
<td>the time, in minutes, after that inactive terminals are automatically</td>
</tr>
<tr>
<td></td>
<td></td>
<td>disconnected. If zero no timeout is set.</td>
</tr>
</tbody>
</table>

**Trace element**

It defines the full path name of XTND trace file. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>-</td>
<td>path name of trace file (%h is substituted with $HOME, %x with $XFRAMEHOME)</td>
</tr>
<tr>
<td>level</td>
<td>none</td>
<td>specified the trace level (none, log, debug)</td>
</tr>
</tbody>
</table>

**License element**

It defines the full path name of XTND license file. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
</table>
file
-  path name of license file (%h is substituted with $HOME, %x with $XFRAMEHOME)

**Administration element**

It defines the administration settings of XTND. If not defined no administration facility is provided. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>-</td>
<td>the port listening for incoming administration connections</td>
</tr>
</tbody>
</table>

**Codepage element**

It defines the codepage to be used by XTND. If defined specified codepage conversion tables are loaded. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full path to the ASCII/EBCDIC EBCDIC/ASCII conversion table (%h is substituted with $HOME)</td>
</tr>
</tbody>
</table>

**Conversion element**

It defines the conversion behaviour of XTND. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uppercase</td>
<td>true</td>
<td>If true XTND automatically converts lowercase chars into uppercase</td>
</tr>
</tbody>
</table>

**security element**

It sets up the eventual usage of SSL in the communication with the terminal emulators. When “ssl” attribute is set to to true, terminal emulators connected to XTND, must communicate over SSL (Secure Socket Layer). The element also defines the attributes of the Key Store.

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssl</td>
<td>false</td>
<td>If true, SSL communication is enabled</td>
</tr>
<tr>
<td>keystore</td>
<td>-</td>
<td>full path of the SSL key store</td>
</tr>
<tr>
<td>keystorepassword</td>
<td>-</td>
<td>pasword to open the keystore</td>
</tr>
</tbody>
</table>

**Application element**

This section, identified by the tag `<application>`, defines the XCICS application parameters. The following attributes may specified:
<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>-</td>
<td>name of application as displayed in welcome screen</td>
</tr>
<tr>
<td>address</td>
<td>-</td>
<td>hostname/IP address of the XCICS application</td>
</tr>
<tr>
<td>port</td>
<td>-</td>
<td>port number of the XCICS application (network_port)</td>
</tr>
<tr>
<td>default</td>
<td>false</td>
<td>if true the current application definition is the one to which terminals connect by default</td>
</tr>
</tbody>
</table>

**Terminal element**

It defines a terminal identifier. The following attributes may specified:

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>-</td>
<td>(mandatory) terminal device identifier. This parameter must match the corresponding &quot;device&quot; (physical) parameter in the &quot;define terminal&quot; statement in xcics.conf</td>
</tr>
<tr>
<td>name</td>
<td>-</td>
<td>synonym for device (for backward compatibility only)</td>
</tr>
<tr>
<td>luname</td>
<td>-</td>
<td>if provided, defines a LU name constraint</td>
</tr>
<tr>
<td>address</td>
<td>-</td>
<td>if provided, defines an hostname/IP address constraint</td>
</tr>
</tbody>
</table>

**Terminals element**

It defines a set of terminals, defined in another application.

<table>
<thead>
<tr>
<th>attribute</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>links</td>
<td>-</td>
<td>application name of the linked terminals</td>
</tr>
</tbody>
</table>

**A sample configuration file**

```xml
<!-- xtnd.xml -->
<!-- xtnd configuration file -->
<configuration>
  <settings>
    <service port="8023" welcome="true" multiplex="true"/>
    <trace file="/home/xtnd/trace.log"/>
    <administration port="7599"/>
    <codepage file="/home/etc/IBM\30.421"/>
    <conversion uppercase="false"/>
  </settings>
  <application name="DEMO" address="ipux01.ibm.net" port="30585">
    <terminal device="TND01DEV0"/>
    <terminal device="TND02DEV0"/>
    <terminal device="TND01DEV1" luname="LU01"/>
  </application>
</configuration>
```
**Java Virtual Machine options**

In order to provide some options directly to the JVM running the XTND server, the environment `XTND_JVM_OPTIONS` must be valorized with the required JVM options, and exported.

```bash
export XTND_JVM_OPTIONS="-Xmx128m -Xms64m"
xtndtl start
```

**Memory issues**

It is very important to properly set up the Java memory heap, when running XTND instances serving a large number of client: a shortage of memory in the JVM running XTND, may cause XTND to freeze. This issue depends on the number of terminals connected and on the maximum heap size defined for the JVM: java heap must be large enough to allow XTND to allocate memory for all the connections.
### 3.4.2 Using SSL

XTND support communication over SSL (Secure Socket Layer). SSL mode only works in multithreaded mode (multiplex=false). To make use of SSL a valid java keystore must be configured.

XTND supports SSLv3 and TLSv1.

#### Generating a keystore

You can use the "keytool" utility, delivered with the Java SDK, to define a valid keystore.

The name of the keystore file (xtndkeystore.jks in the sample below) and the keystore password (xframe in the sample below) must be respectively specified in the keystore and keystorepassword attributes of XTND configuration.

I.e.

```
# keytool -genkey -alias tn3270 -keyalg RSA -keypass xframe -storepass xframe -keystore $HOME/etc/xtndkeystore.jks
```

In the xtnd.xml:

```
<configuration>
  <settings>
    <service port="8040" welcome="true" multiplex="false" tn3270e="true"/>
    <security ssl="true" keystore="/h/etc/xtndkeystore.jks" keystorepassword="xframe"/>
  </settings>
</configuration>
```

Once the keystore has been defined and XTND has been configured and started, terminal emulators with SSL/TSL support may connect it.

#### Setting up emulators for SSL

Many emulators simply require to turn on SSL/TLS support in the session definition before to connect the tn3270 server.

IBM Personal Communication, the most popular 3270 terminal emulator, has a more complex setup. To enable the secure communication between the XTND server and the terminal emulator, the server certificate must be installed on the emulator PC.

To do that follow this procedure:

1. Export the certificate using the keytool utility:

   ```
   keytool -export -rfc -alias tn3270 -keystore xtndkeystore.jks -storepass xframe -file export.arm
   ```

2. Download the "export.arm" file on the PC

3. Start the IBM utility for Certificate Management (IBM Key Management)

4. Open the Personal Communication keys file (normally located in the "private" subdirectory and named PcommClientKeyDb.kdb). To open the file a password is required: "pcomm" is teh IBM default password.

5. Add a certificate: select the downloaded file and provide a label (a mnemonic name)

6. Save the file (overwrite)

7. Define a new session enabling security and connect. If the emulator prompts for a password, supply the password of the key file as in point 4 ("pcomm").

For more detailed information, please refer to your terminal emulator.
3.5 XJWEB Web Adapter

XJWEB is the XFRAME web adapter for XCICS. Its goal is to make available on the WWW data coming form XCICS transaction.

XJWEB is based on JSP/Servlet technology and require a J2EE compatible web server (i.e. Tomcat).

By default XJWEB provides a simple HTML refacing function. It means that no re-arrangement is made to the BMS map sent by XCICS, and the HTML shown by the browser will look very similar to the original 3270 screen.

Otherwise, user may provide its own JSP in order to completely re-arrange the BMS map sent by XCICS and to give a completely new look & feel to the old map.

XJWEB is composed by two main components:

- communication module
- webifier module

The communication module aim is to connect and exchange data with XCICS. It is based on the X4J communication module.

XJWEB is like a normal terminal: it connects XCICS, it works then disconnects. The connection between XCICS and the XJWEB object for the user web session is permanent. Therefore it is very important to correctly define the session timeout, because XJWEB automatically provides disconnection when session is timed out.

The webifier module is the module that allows the transformation of data received by XCICS, in a more programmer-friendly data format. It also provides a set of programming API to make easier the work of the JSP programmers.

XJWEB requires the XCICS transactions to be coded as a pure standard pseudo-conversational programs, that mean that each transaction must be composed by:

1. RECEIVE MAP
2. elaboration
3. SEND MAP

Any other structure may cause unpredictable results (for such situations refer the XJEB/3270 wrapper below).

This documentation assumes you are using Tomcat 4.1 as Java Web Server.

The XJWEB/3270 wrapper

Under some circumstances, the coding of the on line program may result incompatible with the standard required by the XJWEB framework.

In these situations, the XJWEB/3270 servelet may be used. This servlet acts as real terminal emulator, wrapping the underlying 3270 video memory to HTML, enabling the access to the application using a standard web browser without make use of any additional components by the client side.
3.5.1 The XJWEB web application

By the web server point of view, XJWEB is simply a package performing the communication with XCICS. Therefore it must be used in the context of a so called "web-app".

It means that user will have to create one or more webapps to use XJWEB.

For example, on the same webservr, user could have an application to interface XCICS running on system A and another to interface XCICS on system B. Another scenario could be a web server with a webapp to interface only the warehouse transactions, and another to interface the billing transaction

The context

Once installed Tomcat and XFRAME, a dedicated context for the new webapp must be created. To do this simply add to your $CATALINA_HOME/webapps directory a file named <application_name>.xml containing:

```xml
<Context path="/application_name" docBase="/home/webapp" debug="0" reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger" prefix="localhost.application_name.log." suffix=".txt" timestamp="true" />
</Context>
```

bold text above is depending on configuration:

- `/application_name` is the name of the context
- `/home/webapp` is the path to the web app root directory (JSP, HTML, etc.). Since now on this document will refer to to this pas as $DOCBASE

Then the application directories must be create. Under the $DOCBASE the following directory tree must be created:

```
$DOCBASE
    |    +-WEB-INF
    |        +- classes
    |        +- lib
```

then copy the following files in the corresponding path:

- `$XFRAMEHOME/xje/tools/xjweb/web.xml` into $DOCBASE/WEB-INF
- `$XFRAMEHOME/xje/tools/xjweb/*.jsp` into $DOCBASE
- `$XFRAMEHOME/xje/tools/xjweb/*.js` into $DOCBASE
- `$XFRAMEHOME/lib/xjweb.jar` into $DOCBASE/WEB-INF/lib/xjweb.jar

I.e.

```bash
mkdir $DOCBASE
mkdir $DOCBASE/WEB-INF
mkdir $DOCBASE/WEB-INF/classes
cp $XFRAMEHOME/xje/tools/xjweb/web.xml $DOCBASE/WEB-INF
cp $XFRAMEHOME/xje/tools/xjweb/*.jsp $DOCBASE/WEB-INF
cp $XFRAMEHOME/xje/tools/xjweb/*.js $DOCBASE/WEB-INF
cp $XFRAMEHOME/xje/tools/xjweb/*.js $DOCBASE/WEB-INF/lib/xjweb.jar
```
How to create a session with login connection

If you want to add to your XCICS web application an initial login you can define to tomcat a user table containing all enabled users. You have to add this line to your context:

```xml
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
    driverName="oracle.jdbc.driver.OracleDriver"
    connectionURL="jdbc:oracle:thin:@hostname:1521:htordb"
    connectionName="user"
    connectionPassword="password"
    userNameCol="login_name"
    userCredCol="user_pass"
    roleRoleTable="web_users"
    roleNameCol="role_name"
/>```

where driver name define RDBMS in this case is a Oracle DB, connection URL a valid connection to DB instance, connection name and password are valid user and password of defined instance.

All user* parameters are table and columns reference to resolve connection. This is a table describe example

```
SQL> desc web_users;
Name                                      Null?    Type ----------------------------------------
USER_NAME                                  NOT NULL VARCHAR2(15)
USER_PASS                                  VARCHAR2(15)
ROLE_NAME                                  VARCHAR2(15)
LOGIN_NAME                                 NOT NULL VARCHAR2(15)
TERM_NAME                                  CHAR(4)
```

userNameCol setting has to match with user column, and userCredCol has to match with password columns.

See Tomcat documentations for more detailed information.

**Configuring**

To configure the application the file web.xml defined in $DOCBASE/WEB-INF must be edited.

Format of the configuration file:

```xml
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtd/web-app_2_2.dtd"
<web-app>
    <display-name>Application Name</display-name>
    <description>This is the description for the application</description>
    <context-param>
        <param-name>xjweb.license.file</param-name>
        <param-value>license_file</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.default.action</param-name>
        <param-value>default_action</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.default.transaction</param-name>
```
More than one application may be defined and more than one terminal per application may be defined.

The following items may be configured:

<table>
<thead>
<tr>
<th>xjweb.license.file</th>
<th>path to the XJWEB license file (normally under $XFRAMEHOME/license)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xjweb.default.action</td>
<td>name of the XJWEB processing form</td>
</tr>
<tr>
<td>xjweb.default.transaction</td>
<td>default transaction (optional, XSSN by default)</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>xjweb.template.jsp</td>
<td>name of the template JSP for default HTML refacing for BMS maps</td>
</tr>
<tr>
<td>xjweb.template.3270.jsp</td>
<td>name of the template JSP for default HTML refacing for raw 3270 screens</td>
</tr>
<tr>
<td>xjweb.wrapper.template.jsp</td>
<td>name of the template JSP for the XJWEB/3270 wrapper (XJWEB wrapping mode)</td>
</tr>
<tr>
<td>xjweb.resource.path</td>
<td>path to resources (normally the same as the context name)</td>
</tr>
<tr>
<td>xjweb.session.timeout</td>
<td>the session timeout in minutes</td>
</tr>
<tr>
<td>xjweb.timeout.handler.jsp</td>
<td>JSP to handle the timeout event</td>
</tr>
<tr>
<td>xjweb.error.handler.jsp</td>
<td>JSP to handle errors</td>
</tr>
<tr>
<td>xjeb.terminal.prefix</td>
<td>the terminal identifier prefix as specified in xcics.conf</td>
</tr>
<tr>
<td>xjeb.terminal.sessions.number</td>
<td>the maximum terminal sessions available for connected users</td>
</tr>
</tbody>
</table>

An example

```xml
<web-app>
    <display-name>XJWEB Application</display-name>
    <description>This is the xjweb development application</description>
    <context-param>
        <param-name>xjweb.default.action</param-name>
        <param-value>reply</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.default.transaction</param-name>
        <param-value>MENU</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.terminal.prefix</param-name>
        <param-value>WK</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.template.jsp</param-name>
        <param-value>xjweb.jsp</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.template.3270.jsp</param-name>
        <param-value>xjweb3270.jsp</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.wrapper.template.jsp</param-name>
        <param-value>wrapper3270.jsp</param-value>
    </context-param>
    <context-param>
        <param-name>xjweb.timeout.handler.jsp</param-name>
        <param-value>timeout.jsp</param-value>
    </context-param>
</web-app>
```
Connecting XCICS

A connection page must be provided.

It must provide the following fields data to the XJWEB servlet pointed by the parameter xjweb.default.action:

<table>
<thead>
<tr>
<th>xjweb_host</th>
<th>hostname of XCICS server</th>
</tr>
</thead>
<tbody>
<tr>
<td>xjweb_port</td>
<td>port of XCICS application</td>
</tr>
<tr>
<td>xjweb_termname</td>
<td>terminal name (optional)</td>
</tr>
</tbody>
</table>

The connect.jsp provided with XFRAME is a sample of the connect. This is another sample (index.jsp):

```jsp
<%@ page import="java.util.*" %>
<html>
<head><title>Connecting XCICS</title></head>
<body>
  <p>Press the button to connect XCICS</p>
  <form name="xjweb form ACTION="xjweb default.action" method=post">
    <input type="hidden" name="xjweb_host" value="hpux01.ht.net">  
  </form>
</body>
</html>
```
Terminal connected through XJWEB, must be defined as "web" type

If you want to use the XJWEB/3270 wrapper, change the ACTION to the mapping of the servlet XJWeb3270Servlet (wrapper in the sample above) and define terminal as "standard" type.

Handling errors

Each time an error should occur (i.e. XCICS is not available, the terminal is not available, etc), the XJWEB servlet, will invoke the error defined JSP, providing the attribute xjwe.exception, to the user code, to let it handle the error condition. This is a sample error handling page.

```html
<p>
XJWEB get an error:
</p>
<%    Exception e=(Exception) session.getAttribute("xjweb.exception");
    if (e instanceof java.net.ConnectException) {
        out.println("connection refused");
    } else if (e instanceof com.hixj.device.TerminalUnDefinedException) {
        out.println("terminal not defined");
    } else if (e instanceof com.hixj.device.TerminalBusyException) {
        out.println("terminal already in use");
    } else {
        out.println("unhandled exception caught: "+e);
    }
%>
</p>
</html>
3.5.2 The BMS re-facing

XJWEB allows the re-facing of the BMS maps into HTML. To do this it requires a JSP to control this function. User have to provide a default re-facing JSP, which will control all the maps that have no dedicated JSP. Dedicated JSPs may be created simply defining a JSP named as <mapset>.<map>.jsp.

I.e., for a map named MENU01 in the mapset MPSXY user will provide a page named MENU01.MPSXY.jsp.

Programming the re-facing pages

These pages are standard JSP files, therefore they have to be coded respecting the JSP/J2EE standards. Re-facing pages mandatory have to include the file named xjweb.include.jsp:

```jsp
<%@ include file="/xjweb.include.jsp" %>
```

The following objects will be available for programmer after included the file above:

<table>
<thead>
<tr>
<th>Class</th>
<th>Object name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.hite.x4j.device.CXMap</td>
<td>xjMap</td>
<td>Contains all the information provided in the incoming map</td>
</tr>
<tr>
<td>com.hite.x4j.jsp.XJWebHtmlBean</td>
<td>xjweb</td>
<td>Supplies a set of facility to handle map fields in HTML</td>
</tr>
<tr>
<td>java.util.Properties</td>
<td>xjwebSettings</td>
<td>Supplies all the settings for XJWEB</td>
</tr>
<tr>
<td>com.hite.x4j.device.CXResponse</td>
<td>cxResponse</td>
<td>Contains information about latest XCICS communication</td>
</tr>
<tr>
<td>com.hite.x4j.device.CXServletConnector</td>
<td>cxConnector</td>
<td>XCICS connector object, which provides methods to get information about terminal connection, and to interface it (i.e. disconnect)</td>
</tr>
</tbody>
</table>

For detailed information concerning XJWEB API please refer to the document “XJE - Xframe Java Environment” in the XFRAME bookshelf.

Formatted and unformatted screens

XCICS/XJWEB distinguish between formatted screens coming from BMS maps (therefore the output of a SEND MAP command) and raw 3270 data. In the first case XCICS will convert for XJWEB the sent map into an XML stream describing the map. In the second case a raw 3270 stream will be sent to XJWEB.

The BMS fields

When a BMS map is sent to the screen, and therefore, to XJWEB, this one handles the XML provided by XCICS and generates a CXMap object to be used in the JSP to retrieve information about the map and its fields.

For each field a CXField object is created and through this, programmers may get information about the field. These information may only be use to display data in the HTML screen.

I.e.
The input form

All the data involved in a map that have to be returned to XCICS must be included into an HTML form, and must be handled as HTML form fields.

This form must be defined using the methods provided by the class `com.hite.x4j.jsp.XJWebHtmlBean`:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createFormHead()</code></td>
<td>generates the HTML form header</td>
</tr>
<tr>
<td><code>createFormTail()</code></td>
<td>generates the HTML form footer</td>
</tr>
</tbody>
</table>

I.e.

```html
<p>an html paragraph</p>
<% xjweb.createFormHead()%> <!-- body of the map form -->
<% xjweb.createFormTail()%>
```

The form fields

As describe above data returning to XCICS, must be handled as HTML field. These fields must be defined as specified as follows:

the name must be the same as BMS field

the field type may be text, hidden or password

Moreover, in order to grant input integrity, user edited fields should make use of the following Javascript functions on the onChange event:

- `_xjweb_on_change_(this)` if a field is not numeric
- `_xjweb_on_changenum_ (this)` if a field is numeric

I.e.

```html
<p>an html paragraph</p>
<% xjweb.createFormHead()%>
```
However, XJWebHtmlBean provides a set of method to easily wrap the data coming from XCICS into HTML fields. I.e.

```html
<!-- body of the map form -->
<input type="hidden" name="CODE" value="123">

Please enter your name:
<input type="text" name="NAME" onChange="_xjweb_on_changenum_(this)">

Please enter your age:
<input type="text" name="NAME" onChange="_xjweb_on_changerum_(this)"

```xjweb createFormTail()

The AID keys

In order to simulate the AID keys, the form field `_xjweb_eibaid_` is automatically defined by the `createFormHead()`. This field must be set according with the following to return the desired AID to XCICS:

<table>
<thead>
<tr>
<th>Value</th>
<th>AID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DFHCLEAR</td>
</tr>
<tr>
<td>1</td>
<td>PF1</td>
</tr>
<tr>
<td>2</td>
<td>PF2</td>
</tr>
<tr>
<td>3</td>
<td>PF3</td>
</tr>
<tr>
<td>4</td>
<td>PF4</td>
</tr>
<tr>
<td>5</td>
<td>PF5</td>
</tr>
<tr>
<td>6</td>
<td>PF6</td>
</tr>
<tr>
<td>7</td>
<td>PF7</td>
</tr>
<tr>
<td>8</td>
<td>PF8</td>
</tr>
</tbody>
</table>
The function `_xjweb_setaid_(<value>)` should be used to set the desired AID.

In any case, user should use XJWebHtmlBean methods to define buttons:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createButton</td>
<td>generates a button providing caption and/or image</td>
</tr>
<tr>
<td>createHtml4Button</td>
<td>generates an HTML 4 button providing HTML code</td>
</tr>
<tr>
<td>createButtonPanel</td>
<td>generates the entire the whole panel</td>
</tr>
</tbody>
</table>

I.e.
**Transaction processing**

As well known by CICS programmers, data entered on the screen may determine the transaction run by CICS/XCICS. If the program is under the control of a RETURN TRANSID the TP monitor knows what to do and, when user enters data, starts the transaction previously requested by the application. Instead, when the program simply executes a RETURN command, the transaction started will be determined depending on the first bytes coming from the 3270 screen.

Using XJWEB programmers must take care about this condition. While there is no problem with the RETURN TRANSID, when program terminates with a simple RETURN, XJWEB programmer must communicate to XCICS which transaction to start.

The may provide this information in two ways:

- directly specifying the transaction code to start
- specifying the name of the field that will contain the transaction code

In the first case, programmers must use the method `createTransactionField()` to specify the transaction code to restart. I.e.

```html
&lt;xjweb createTransactionField("MENR")&gt;
&lt;xjweb createFormHead()&gt;
&lt;xjweb createField("ITEM")&gt;
&lt;xjweb createButton("Proceed", XNdbHtmlBean.KEY_ENTER) &gt;
&lt;/xjweb createFormTail() &gt;
```

In the second one, programmers must use the method `createTransactionFieldId()` to specify the field from which XCICS will get the transaction code to restart. This field may be either a BMS field or another HTML form field (maybe hidden) containing the information.

```html
&lt;xjweb createFormHead()&gt;
&lt;xjweb createField("MSG")&gt;
if (f!-null) {
  if ( !f.hasInitial() ) {
    out.println("the system says: "+ f.getInitial());
  } else {
    out.println("no message from the system");
  }
} else {
  out.println("The field MSG cannot be retrieved");
}
&lt;/xjweb createFormTail()&gt;
```
Defining the default re-facer for BMS maps

The default wrapper for BMS maps is the JSP handling all the non customized maps. It have to make use of the `createFormBody()` method. This method provides a generic body for all.

This is the simplest default re-facer JSP:

```html
<%@ include file="/xjweb.include.jsp" %>
<html>
<head>
<title>XJWEB default re-facer</title>
</head>

<cxf createFormHead() %>
<cxf createFormBody() %>
<cxf createButtonPanel() %>
<cxf createFormTail() %>
</html>
```

Starting from the JSP above, user may customize its default wrapper adding any kind of resource (logos, pictures, links, etc).

Defining the wrapper for 3270 unformatted screens

The wrapper for 3270 unformatted screens is the JSP handling all the data not included in a BMS definition coming from XCICS.

This is the simplest wrapper JSP for 3270 unformatted screens:

```html
<%@ include file="/xjweb.include.jsp" %>
<html>
```
Starting from the JSP above, user may customize its default wrapper adding any kind of resource (logos, pictures, links, etc).
3.5.3 The XJWEB/3270 wrapper

As described in the chapters above, the XJWEB/3270 wrapper servlet allows the users to connect to an XCICS application using a web browser, without the need for any additional software. This means the no Applet or Active/X is downloaded by the client side. The only requirement is a web browser.

Note: please remeber to define terminal connected via XJWEB/3270 as "standard" type.

**Client side requirements**

By the client side Microsoft Internet Explorer 6 or Netscape 6 are required.

**Known limitations**

The XJWEB/3270 technology cannot be mixed with the standard XJWEB technology.

**Keyboard key handling**

If you wanna enable the use of keyboard keys to access the terminal keys (i.e. DFHENTER, DFHPF23, etc) the script named "xjweb.handlekeys.js" must be included in the wrapper page.

When the page is loaded, the function must be initialized issuing
3.6 XCICS Terminal Emulators

XCICS/TS is deployed with its own terminal emulator:

- X4J - the terminal emulator for the Java Virtual Machine
- xsclient - the terminal emulator for Unix/Linux tty sessions (telnet/ssh)

Both these emulators may be used to connect an active XCICS/TS region over its proprietary protocol. X4J is designed for the every-day work of the terminal operator; xsclient is delivered to all offer terminal emulation capabilities to those users that only have access to the Unix/Linux shell.

If you want to connect an XCICS/TS region with a standard tn3270 emulator (i.e. IBM Personal Communicator), please refer to the XTND Protocol Adapter guide.
3.6.1 X4J

X4J is a 3270 terminal emulator for XCICS/TS. Using X4J, you may connect to any active XCICS region, from every system supporting the JAVA Runtime Environment (JRE) version 1.4 or higher.

X4J, is the most efficient way to connect to an XCICS session, not only because it improves the host server and network performances, but also because it supplies an "user-friendly" interface, supporting mouse, menus and all features of a windowing system.

For further information please refer to the X4J section.
3.6.2 xsclient terminal emulator for UNIX

This is the basic client for UNIX/Linux. It may be used both on UNIX/Linux servers running XCICS, and on a remote server connected via TCP/IP to the main server. It can be launched from each shell on a terminal connected to the system, both by serial connection, Telnet and SSH connection. XCICS will accept connection if the IP address of the server on which Xsclient is running and the physical name of the workstation has been enabled in the application Terminal Control Table. This tool will supply a character based interface to the application, using the capabilities of UNIX terminals (tty).

Syntax

xsclient [ -?Ulwshpta]

If started without options, xsclient will try to connect the Xcics session, running on the local machine, and listening to default TCP port (30585).

Options

- **h<host>** to connect a session running on the specified host. If no host option is specified, local host is assumed by default.
- **-p<port>** to connect a session listening on the specified port. If no port option, port 30585 is assumed by default.
- **-U** to enable auto uppercase writing. If this option is not specified both uppercase and lowercase characters are accepted.
- **-s<f|r|n|u>** to specify the status line mode:
  - f means full mode. The status line uses the 25th line of the screen, and it is always visible for displaying information about connection, keyboard and transmission state.
  - r means reduced mode. The status line uses the 25th line of the screen, and it shows only keyboard and transmission state.
  - n means none. Status line uses the 24th line of the screen, showing flags only for a locked keyboard and in a waiting state. This is the default value.
  - u means in the upper side of screen. Status line uses the 1st line of the screen, showing flags only for a locked keyboard and in a waiting state.
- **-r<file>** to record the transaction in the specified file.
- **-t<ttty>** to ask for COBOL debug, starting the MicroFocus Animator session on the specified tty terminal.
- **-a<ttty>** to ask for X370 assembler debugging, starting the debug session on the specified tty terminal.
- **-l<term>** to ask the system to associate to this workstation the logical name supplied with the parameter.
- **-w** enables wait state for START TRANSID.

**User Interface**

A User interface for the enhanced client, is provided for handling of capabilities of Unix terminals, by using Curses libraries, providing an interface for all terminals supported by Curses and TAM libraries of the machine running the xclient interface.

Obviously, like for the standard client, features of this interface depend on terminal capabilities, but, what about concerning keyboards, the same facilities as for the standard client are provided. Please refer to the table "Special keys for character based terminals" below.
# Special Keys

## Table 1- Special keys for character based terminals

<table>
<thead>
<tr>
<th>3270 Terminal</th>
<th>Terminal Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>CTRL G</td>
</tr>
<tr>
<td>[PF1- PF9]</td>
<td>CTRL F + [1 - 9]</td>
</tr>
<tr>
<td>[PF10 - PF24]</td>
<td>CTRL F + [a - o]</td>
</tr>
<tr>
<td>PA1</td>
<td>CTRL A</td>
</tr>
<tr>
<td>PA2</td>
<td>CTRL B</td>
</tr>
<tr>
<td>PA3</td>
<td>CTRL C</td>
</tr>
<tr>
<td>Erase Field</td>
<td>CTRL U</td>
</tr>
<tr>
<td>Erase until end of screen</td>
<td>CTRL L</td>
</tr>
<tr>
<td>Erase all unprotected</td>
<td>CTRL P</td>
</tr>
<tr>
<td>Delete</td>
<td>CTRL H</td>
</tr>
<tr>
<td>Tab</td>
<td>Tab / CTRL T</td>
</tr>
<tr>
<td>Back Tab</td>
<td>CTRL X</td>
</tr>
<tr>
<td>Home</td>
<td>CTRL Y</td>
</tr>
</tbody>
</table>

## Table 2- Special keys for Windows

<table>
<thead>
<tr>
<th>3270 Terminal</th>
<th>PC Enhanced keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR</td>
<td>ESCAPE</td>
</tr>
<tr>
<td>[PF1- PF9]</td>
<td>[F1-12]</td>
</tr>
<tr>
<td>[PF10 - PF24]</td>
<td>SHIFT + [F1-12]</td>
</tr>
<tr>
<td>PA1</td>
<td>Pg Up</td>
</tr>
<tr>
<td>PA2</td>
<td>Pg Down</td>
</tr>
<tr>
<td>PA3</td>
<td>SHIFT + Pg Up</td>
</tr>
<tr>
<td>Start Of Field</td>
<td>HOME</td>
</tr>
<tr>
<td>End Of Field</td>
<td>END</td>
</tr>
<tr>
<td>New Line</td>
<td>ENTER</td>
</tr>
<tr>
<td>Enter</td>
<td>Right CTRL</td>
</tr>
</tbody>
</table>
Part IV

X4J 3270 Terminal Emulator
4 X4J 3270 Terminal Emulator

X4J is a 3270 terminal emulator for XCICS/TS. It runs on every Java Virtual Machine (1.4 or higher) and allows the terminal communication over the XCICS/TS native protocol.

The X4J's most important features are:

- Java WebStart ready
- centralized session and configuration deployment
- auto font sizing
- visually configurable keyboard
- configurable messages
- distributable configuration files
- PowerGUI
- text area selection
- configurable user color
- configurable code page
- automatic connection
- cross beam cursor
- keyboard buffering
- Full JAVA 2 coding
- Swing! based user interface
- pluggable Look & Feel support
- Macro play/recording
- Mouse fast copy&paste
- Toggable Ctrl key handling
- Window size and position back up
- Macro termination command
- Network, System & JVM information
- Port range assignment for incoming connections
4.1 The menuing system

Most of the new features of X4J are accessible via menu. Three different menus are available from the menu bar: the Session menu, the Edit menu, and the Help menu.

**Connection menu**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Opens the direct connection dialog, in order to connect directly to an XCICS application.</td>
</tr>
<tr>
<td>Session</td>
<td>Opens the stored sessions dialog, to edit the list of available sessions, and to connect one of them.</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Enabled only if connected, order disconnection from current XCICS application.</td>
</tr>
<tr>
<td>Print Screen</td>
<td>Starts an hardcopy of the screen, if connected;</td>
</tr>
<tr>
<td>Options</td>
<td>Opens the options dialog, to set up X4J parameters.</td>
</tr>
<tr>
<td>Exit</td>
<td>Terminates X4J</td>
</tr>
</tbody>
</table>
# Edit menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>copies the selected text area (if existing) into the system clipboard.</td>
</tr>
<tr>
<td>Copy append</td>
<td>copies the selected text area (if existing) appending to the system clipboard</td>
</tr>
<tr>
<td>Paste</td>
<td>pastes the system clipboard at the current cursor position.</td>
</tr>
<tr>
<td>Paste resume</td>
<td>continues to paste the system clipboard content</td>
</tr>
<tr>
<td>New line</td>
<td>moves the cursor on next editable line</td>
</tr>
<tr>
<td>HOME</td>
<td>locates cursor on the first editable field on screen</td>
</tr>
<tr>
<td>Erase Field</td>
<td>erases (in fact fulfills with binary zeroes) the content of the field where the cursor is positioned.</td>
</tr>
<tr>
<td>Erase EOF</td>
<td>erases the current unprotected field from the current cursor position.</td>
</tr>
<tr>
<td>Erase EOS</td>
<td>erases all unprotected fields from the current cursor position.</td>
</tr>
<tr>
<td>Erase ALL</td>
<td>erases all unprotected fields on the screen.</td>
</tr>
</tbody>
</table>
Macro menu

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record</td>
<td>Starts the macro recording mode</td>
</tr>
<tr>
<td>Stop recording</td>
<td>Stops the macro recording mode</td>
</tr>
<tr>
<td>Play</td>
<td>Gives access to the defined macro playback</td>
</tr>
<tr>
<td>Delete</td>
<td>Shows the delete macro form</td>
</tr>
<tr>
<td>Stop macro</td>
<td>Stop macro playback</td>
</tr>
</tbody>
</table>
Help

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>opens the online help system.</td>
</tr>
<tr>
<td>About</td>
<td>shows information about X4J.</td>
</tr>
<tr>
<td>Connection status</td>
<td>shows network connection status</td>
</tr>
<tr>
<td>Dump last 3270 order</td>
<td>shows the last SNA order sent or received in hex mode.</td>
</tr>
</tbody>
</table>
4.2 The toolbar

Some of the function performed by X4J are reachable directly using the toolbar buttons.

All buttons are completed with a tooltip explaining their function: simply locate the cursor on the button and a tooltip is shown.

Moreover, by means of the Function Key Bar, 3270 Terminal AIDs are made available by a single-click on the mouse. It acts like a normal toolbar, and it supports the ENTER key, the CLEAR key, the attention identifiers PA1, PA2 and PA3 and the program function keys PF1 through PF12 (PF13 - PF24 with PF keys set ON).
4.3 Terminal Display Area

This is the terminal display buffer, and it is the area that interacts with the XCICS session. Every information sent to a terminal from the XCICS application is collected and shown in this area as soon as a complete message has been received. Moreover, when the cursor is located in this area and the host has released the keyboard (Free Keyboard Order), it is possible to enter data on it.

If PowerGUI is set (see Options section) screen will look as a Windows? form.

Mouse interaction

By clicking left mouse button, cursor locates to the specified position.

By double-clicking left mouse button, clipboard content is pasted to the current position.

By clicking right mouse button and dragging mouse, text is selected. If auto copy is set, selected text is automatically copied to the clipboard.

Keyboard selection

By pressing Shift and moving the cursor using the arrows key it is also possible to select text.
4.4 The Keyboard

Obviously, data are entered by using the keyboard. Some special keys supported by the 3270 terminals have been redirected using the keys of an enhanced PC keyboard.

The keyboard is configurable: see section "Configuration".

Status Bar

This area is used to notify a particular host status.

From left to right the following information are found:

- connection message
- EIBTRMID (logical terminal name)
- Physical name
- PID (CXRUN process identifier)
- Last elapsed time
- Overwrite/Insert status
- Keyboard status (locked/unlocked)
- Terminal status (waiting/ready)
4.5 Connect dialog

Sometimes we need to log in with special parameters or to a spot session. In these case we can use the Connect dialog to connect an XCICS session with all needed parameters.

Host Name and Port are required to login to a specified XCICS session. The Host Name refers to the IP address or domain name of the host server where XCICS is running, as defined in the 'hosts' file (normally located under the windows directory. It can also have the name 'hosts.sam').

The port number specifies the listening port of the XCICS Dispatcher Daemon, and it must be supplied to the user by the XCICS Administrator. The default port is 30585.

The Session Identifier combo is used to select what kind of identification must be supplied to XCICS. Possible values are:

- User defined (user must enter the terminal physical name)
- Network identified (X4J supplies the network/display name as p.n.)
- Auto session 1 to 4 (X4J identifies as session 1, 2, 3, or 4)

Logical name is an optional parameter, and it has to be supplied only when users want to handle a specified logical terminal (User defined selected). In these cases, the name supplied overrides the name entered in the TCT in correspondence to the TCP/IP address of the involved workstation.

XCICS Server type defines the protocol used to connect XCICS. Possible values are:

- Dedicated process server
  Connect XCICS serving via a terminal dedicated single process (XCICS version 6.x or earlier)
- Shared engine server
  Connect XCICS serving via a set of preallocated engine processes (XCICS version 7.x)

NOTE: since version 2.0 the XCICS server type is always forced to Shared engine server and the selector is not available. To enable the protocol selection, the JVM must be started with the property "enable.protocol.v6" set to true.
4.6 Sessions dialog

Session entries can be stored, edited and recalled by X4J. When the sessions dialog is called, it is possible to handle sessions.

It is possible, using Add and Edit buttons, to insert and update sessions.

The information required by this dialog are the same of the Connect dialog, plus Session name and Session description.

The Remove button removes one entry from the connection list.

The Set as default makes the selected session as the default session.

X4J stores the predefined sessions in file called 'x4j.ses', after every modification, and once defined it can be distributed to all other users, avoiding the necessity to configure the sessions for each single workstation.

By pressing the Connect button X4J connects the specified session.

Disabling session management

Session management may be disabled using the "Disable local management" option in the Options form.

This can be useful to system administrators when deploying centralized sessions and configuration from a server, in order to avoid the user to the defined inopportune sessions.
4.7 Options dialog

Option dialog allow the user to configure X4J. All settings are store in file called ‘x4j.rc’. This file, once defined, may be deployed to all user granting them a standard base configuration.

Options are collected into three different tabs:

- General
- Display
- Color setup

General settings

Though this tab user can set general parameters for X4J. The following options are accessible from this tab:

<table>
<thead>
<tr>
<th>Screen size</th>
<th>Screen size</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 x 80</td>
<td>24 x 80</td>
</tr>
<tr>
<td>43 x 80</td>
<td>43 x 80</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Force uppercase</th>
<th>Force uppercase</th>
</tr>
</thead>
<tbody>
<tr>
<td>if flagged uppercase</td>
<td>if flagged uppercase</td>
</tr>
</tbody>
</table>

© 2007 H.T.W.C. Srl
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Respect numeric fields</td>
<td>If flagged non numeric input is denied on numeric defined field</td>
</tr>
<tr>
<td>Switch PF key set</td>
<td>If flagged PF keys are switched to PF13 to PF24</td>
</tr>
<tr>
<td>Keyboard buffering</td>
<td>If flagged X4J bufferizes user input, even if host set keyboard to LOCKED</td>
</tr>
<tr>
<td>Automatic cursor positioning</td>
<td>If flagged X4J automatically locates the cursor</td>
</tr>
<tr>
<td>Handle question mark (?) as numeric</td>
<td>If flagged X4J considers the &quot;?&quot; as a digit</td>
</tr>
<tr>
<td>Handle Control key (CTRL)</td>
<td>If flagged X4J handle Ctrl key</td>
</tr>
<tr>
<td>Block mode cursor</td>
<td>If flagged X4J draws a rectangular cursor</td>
</tr>
<tr>
<td>Preserve left Control (CTRL) key for clipboard</td>
<td>If flagged X4J handle only right Ctrl, allowing the use of the left Ctrl key for clipboard operations (i.e. CTRL+C)</td>
</tr>
<tr>
<td>Show small toolbars</td>
<td>If flagged a smaller sized toolbar is shown</td>
</tr>
<tr>
<td>Block-mode paste</td>
<td>If flagged block mode past from clipboard is activated</td>
</tr>
<tr>
<td>Auto copy on selection</td>
<td>If flagged clipboard content is paste by a mouse double-click</td>
</tr>
<tr>
<td>Show macro toolbar</td>
<td>If flagged the macro toolbar is shown</td>
</tr>
<tr>
<td>Select by right mouse button</td>
<td>If flagged text selection is activate only by right mouse button</td>
</tr>
<tr>
<td>Ask for confirmation on exit</td>
<td>If flagged X4J asks the user to confirm the exit command</td>
</tr>
<tr>
<td>Save settings on exit</td>
<td>If flagged settings and sessions are automatically stored at X4J termination</td>
</tr>
<tr>
<td>Save window size and position</td>
<td>If flagged X4J saves window size and position</td>
</tr>
<tr>
<td>On Startup</td>
<td>It determines the X4J behaviour on startup. Possible choices are:</td>
</tr>
<tr>
<td></td>
<td>• Show session dialog</td>
</tr>
<tr>
<td></td>
<td>• Connect the default session (as defined in sessions dialog)</td>
</tr>
<tr>
<td></td>
<td>• Do Nothing</td>
</tr>
</tbody>
</table>

**Keyboard customization**

By pressing the "Customize keyboard" button, the keyboard customization window is shown.
To setup a key function:

- select the key from the keyboard shown
- associate the desired function for the base key
- associate the desired function for the key pressed together with "SHIFT" key
- associate the desired function for the key pressed together with "CTRL" key
- press the "Set" button

The "Reset" button restores X4J original keyboard setting for the selected key.

Once done press the "Ok" button to confirm all changes, or "Cancel" to discard everything.
### Communication Settings

#### Port range
User may specify the port range on which X4J listens for incoming connections. This option is useful when X4J is run behind a firewall.

#### Enable session management
Enables/disables the session management in the Sessions form.

#### Provide session deployment password
User may specify here his session deployment password eventually provided from network/system administrator.

#### Codepage
User may select an alternate codepage using the button "choose" or reset to default using the button "reset". Some codepage files (.cpg extension) are distributed with X4J, or may be found in the XFRAME installation in $XFRAMEHOME/tools.
Display Settings

Display settings in this tab are contained settings involved with display.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show attribute byte</td>
<td>If flagged the 3270 attribute is shown in the display</td>
</tr>
<tr>
<td>Power GUI</td>
<td>If flagged terminal display is shown as a Windows™ form</td>
</tr>
<tr>
<td>Font</td>
<td>The font used to display the terminal area. Be very careful specifying fonts as most of the available font are not cross-platform. By default &quot;Monospaced&quot; is used</td>
</tr>
<tr>
<td>Bold font</td>
<td>If flagged X4J shows bold characters</td>
</tr>
<tr>
<td>NULL characters</td>
<td>NULL characters can be:</td>
</tr>
<tr>
<td></td>
<td>- hide</td>
</tr>
<tr>
<td></td>
<td>- shown</td>
</tr>
<tr>
<td></td>
<td>- shown only if unprotected fields</td>
</tr>
<tr>
<td>Look &amp; Feel</td>
<td>According to installed Swing! installed LAF, user can choose his preferred look &amp; feel</td>
</tr>
</tbody>
</table>
If host colors is selected, display shows text with standard host colors. If user colors is selected, user can define its own colors (see color setup tab).

### Color setup

If user selects user color mode, he can define its own colors.

To define a color, just choose the text type and press Choose: a color chooser dialog will be shown.

User can define color for:

- normal protected text
- highlight protected text
- dark protected text
- blinking protected text
- normal unprotected text
- highlight unprotected text
- dark unprotected text
- blinking unprotected text
Print settings

User can setup some parameters that affect the print screen function. The parameters are:

- font size
- left margin size
- top margin size
4.8 Using macro

X4J allow the definition of 12 macro. A macro is a snapshot of some user input.

**Recording macro**

To record a macro simply select the Record item from the Macro menu. The Macro definition dialog is shown.

![Macro definition dialog](image)

User must supply a name for the macro and an HotKey definition. An HotKey is keyboard combination composed by ALT +F1/12.

By selecting Record from the dialog above, X4J starts recording the macro.

To stop recording, simply select the Stop recording item from the Macro menu.

**Macro playback**

To execute a macro once again, simply use the associated HotKey (ALT+F1/12), or select the Play item from the Macro menu. A submenu showing all the defined macros is shown.
Deleting a macro

To delete a macro, select Delete from the Macro menu. The Delete macro dialog will be shown.

By selecting the macro to delete in the list shown, and pressing delete button, the macro will be deleted.
4.9 Miscellaneous

**About dialog**

Information about version of X4j are supplied by the About dialog.

![About dialog](image)

**3270 Order dialog**

In order to allow easy application debug, last 3270 order may be dumped.
4.10 Configuration

X4J configuration is based on five files, which can be easily distributed. The files are:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x4j.rc</td>
<td>X4J settings as defined in the options dialog</td>
</tr>
<tr>
<td>x4j.ses</td>
<td>predefined session as defined in the session dialog</td>
</tr>
<tr>
<td>x4j.msg</td>
<td>all text messages</td>
</tr>
<tr>
<td>x4j.kbd</td>
<td>keyboard definition</td>
</tr>
<tr>
<td>x4j.mac</td>
<td>macro definition</td>
</tr>
<tr>
<td>x4j.adm</td>
<td>common settings</td>
</tr>
</tbody>
</table>

By default files are searched in the working directory for WIN32 systems, and in the HOME directory for UNIX Systems.

Common settings

In order to prevent users from performing certain tasks on settings, a common configuration file has been provided. By supplying the file named x4j.adm, containing the settings that the system administrator wants to deploy, X4J assumes them as fixed, and prevents them from final user modification.

Text messages

All text messages can be user-defined (i.e., translated), just updating the file x4j.msg. This is an ASCII text file, structured in pairs key/value.

Following keys are available:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>buttons.ok</td>
<td>buttons.cancel</td>
</tr>
<tr>
<td>buttons.add</td>
<td>buttons.remove</td>
</tr>
<tr>
<td>buttons.edit</td>
<td>buttons.refresh</td>
</tr>
<tr>
<td>buttons.setdefault</td>
<td>buttons.connect</td>
</tr>
<tr>
<td>buttons.choose</td>
<td>buttons.reset</td>
</tr>
<tr>
<td>msg.error</td>
<td>msg.warning</td>
</tr>
<tr>
<td>msg.connection.closed</td>
<td>msg.connection.refused</td>
</tr>
<tr>
<td>msg.host.unknown</td>
<td>msg.terminal.busy</td>
</tr>
<tr>
<td>msg.terminal.unavailable</td>
<td>msg.error.detected</td>
</tr>
<tr>
<td>msg.really.exit</td>
<td>msg.disconnect</td>
</tr>
<tr>
<td>msg.missinghost</td>
<td>msg.missingport</td>
</tr>
<tr>
<td>msg.missingterm</td>
<td>msg.wrongterm</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>msg.noname</td>
<td>msg.nodesc</td>
</tr>
<tr>
<td>msg.press.enter.to.restart</td>
<td>msg.connected.to</td>
</tr>
<tr>
<td>msg.not.connected</td>
<td>mainframe.title.unconnected</td>
</tr>
<tr>
<td>fastconnect.title</td>
<td>fastconnect.edit.title</td>
</tr>
<tr>
<td>fastconnect.sessionname</td>
<td>fastconnect.description</td>
</tr>
<tr>
<td>fastconnect.hostname</td>
<td>fastconnect.port</td>
</tr>
<tr>
<td>fastconnect.terminal</td>
<td>fastconnect.sessionid</td>
</tr>
<tr>
<td>settings.title</td>
<td>settings.general</td>
</tr>
<tr>
<td>settings.display</td>
<td>settings.screensize</td>
</tr>
<tr>
<td>settings.nullmode</td>
<td>settings.colormode</td>
</tr>
<tr>
<td>settings.uppercase</td>
<td>settings.switchpf</td>
</tr>
<tr>
<td>settings.powergui</td>
<td>settings.buffering</td>
</tr>
<tr>
<td>settings.askonexit</td>
<td>settings.saveonexit</td>
</tr>
<tr>
<td>settings.showsession</td>
<td>settings.showattr</td>
</tr>
<tr>
<td>settings.hostcolor</td>
<td>settings.enhancedcolor</td>
</tr>
<tr>
<td>settings.usercolor</td>
<td>settings.shownull</td>
</tr>
<tr>
<td>settings.hidenull</td>
<td>settings.showunprot</td>
</tr>
<tr>
<td>settings.startup</td>
<td>settings.donothings</td>
</tr>
<tr>
<td>settings.connectdefault</td>
<td>settings.color.setup</td>
</tr>
<tr>
<td>setting.protected.normal</td>
<td>setting.protected.highlight</td>
</tr>
<tr>
<td>setting.protected.dark</td>
<td>setting.protected.blinking</td>
</tr>
<tr>
<td>setting.unprotected.normal</td>
<td>setting.unprotected.hilite</td>
</tr>
<tr>
<td>setting.unprotected.dark</td>
<td>setting.unprotected.blinking</td>
</tr>
<tr>
<td>setting.background</td>
<td>sessions.title</td>
</tr>
<tr>
<td>menu.session</td>
<td>menu.session.fastconnect</td>
</tr>
<tr>
<td>menu.session.sessions</td>
<td>menu.session.disconnect</td>
</tr>
</tbody>
</table>
menu.session.options
menu.session.crossbeam

menu.session.printscreen
menu.session.exit

menu.edit
menu.edit.copy

menu.edit.paste
menu.edit.delete

menu.edit.erasefield
menu.edit.eraseeos

menu.edit.eraseeof
menu.edit.eraseall

menu.help
menu.help.contents

menu.help.about
menu.help.dump3270

Sample (italian translation):

```
# Pulsanti e bottoni
#
buttons, ok=Ok
buttons, cancel=Annulla
buttons, add=Aggiungi
buttons, remove=Rimuovi
buttons, edit=Modifica
buttons, refresh=Aggiorna
buttons, setdefault=Imposta come default
buttons, connect=Collegati
buttons, choose=Selezione
buttons, reset=Ripristina default
#
# messaggi e segnalazioni
#
msg.error=Errore
msg.warning=Attenzione
```

**Keyboard configuration**

Also keyboard can be user defined, by updating the file x4j.kbd. This is an ASCII text file, structured in pairs key/value.

Please note that keyboard configuration is affected by the option Handle Control Key. In fact, if X4J can handle the Ctrl key, some accelerators will not available, instead, if X4J does not handle Ctrl key, it can be used for accelerator keys (such Ctrl+C or Ctrl+V).

Some systems (Linux) seems to handle Ctrl and Alt keys differently from other system, such Windows or Solaris, therefore X4J must not be configured to handle Ctrl key.

For example, if handling Ctrl key, configuration may look like this:

```
Ctrl=7d
Enter=12
```

instead, if not handling Ctrl key:

```
Enter=7d
```
### Key list (each key is a keyboard key)

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backspace</td>
<td>Ctrl</td>
</tr>
<tr>
<td>Delete</td>
<td>Down</td>
</tr>
<tr>
<td>End</td>
<td>Enter</td>
</tr>
<tr>
<td>Escape</td>
<td>F1</td>
</tr>
<tr>
<td>F2</td>
<td>F3</td>
</tr>
<tr>
<td>F4</td>
<td>F5</td>
</tr>
<tr>
<td>F6</td>
<td>F7</td>
</tr>
<tr>
<td>F8</td>
<td>F9</td>
</tr>
<tr>
<td>F10</td>
<td>F11</td>
</tr>
<tr>
<td>F12</td>
<td>Home</td>
</tr>
<tr>
<td>Insert</td>
<td>Left</td>
</tr>
<tr>
<td>Page\ Down</td>
<td>Page\ Up</td>
</tr>
<tr>
<td>Pause</td>
<td>Right</td>
</tr>
<tr>
<td>Tab</td>
<td>Up</td>
</tr>
</tbody>
</table>

### Function list/value (each value is function performed)

<table>
<thead>
<tr>
<th>Function</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cursor Left</td>
<td>0a</td>
</tr>
<tr>
<td>Cursor Right</td>
<td>0b</td>
</tr>
<tr>
<td>Cursor Up</td>
<td>0c</td>
</tr>
<tr>
<td>Cursor Down</td>
<td>0d</td>
</tr>
<tr>
<td>Cursor Home</td>
<td>0e</td>
</tr>
<tr>
<td>Next Field</td>
<td>10</td>
</tr>
<tr>
<td>Previous Field</td>
<td>11</td>
</tr>
<tr>
<td>New Line</td>
<td>12</td>
</tr>
<tr>
<td>Erase end-of-screen</td>
<td>14</td>
</tr>
<tr>
<td>Erase all</td>
<td>15</td>
</tr>
<tr>
<td>Action</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Erase end-of-field</td>
<td>16</td>
</tr>
<tr>
<td>Erase field</td>
<td>17</td>
</tr>
<tr>
<td>Backspace</td>
<td>18</td>
</tr>
<tr>
<td>Cancel</td>
<td>19</td>
</tr>
<tr>
<td>Toggle insert</td>
<td>1a</td>
</tr>
<tr>
<td>DFHENTER</td>
<td>7d</td>
</tr>
<tr>
<td>DFHCLEAR</td>
<td>6d</td>
</tr>
<tr>
<td>PA1</td>
<td>6c</td>
</tr>
<tr>
<td>PA2</td>
<td>6e</td>
</tr>
<tr>
<td>PA3</td>
<td>6b</td>
</tr>
<tr>
<td>PF1</td>
<td>f1</td>
</tr>
<tr>
<td>PF2</td>
<td>f2</td>
</tr>
<tr>
<td>PF3</td>
<td>f3</td>
</tr>
<tr>
<td>PF4</td>
<td>f4</td>
</tr>
<tr>
<td>PF5</td>
<td>f5</td>
</tr>
<tr>
<td>PF6</td>
<td>f6</td>
</tr>
<tr>
<td>PF7</td>
<td>f7</td>
</tr>
<tr>
<td>PF8</td>
<td>f8</td>
</tr>
<tr>
<td>PF9</td>
<td>f9</td>
</tr>
<tr>
<td>PF10</td>
<td>7a</td>
</tr>
<tr>
<td>PF11</td>
<td>7b</td>
</tr>
<tr>
<td>PF12</td>
<td>7c</td>
</tr>
<tr>
<td>PF13</td>
<td>c1</td>
</tr>
<tr>
<td>PF14</td>
<td>c2</td>
</tr>
<tr>
<td>PF15</td>
<td>c3</td>
</tr>
<tr>
<td>PF16</td>
<td>c4</td>
</tr>
<tr>
<td>PF17</td>
<td>c5</td>
</tr>
</tbody>
</table>
By default X4J keyboard is

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF18</td>
<td>c6</td>
</tr>
<tr>
<td>PF19</td>
<td>c7</td>
</tr>
<tr>
<td>PF20</td>
<td>c8</td>
</tr>
<tr>
<td>PF21</td>
<td>c9</td>
</tr>
<tr>
<td>PF22</td>
<td>d3</td>
</tr>
<tr>
<td>PF23</td>
<td>d4</td>
</tr>
<tr>
<td>PF24</td>
<td>d5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key Combination</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>PF18</td>
<td>c6</td>
</tr>
<tr>
<td>PF19</td>
<td>c7</td>
</tr>
<tr>
<td>PF20</td>
<td>c8</td>
</tr>
<tr>
<td>PF21</td>
<td>c9</td>
</tr>
<tr>
<td>PF22</td>
<td>d3</td>
</tr>
<tr>
<td>PF23</td>
<td>d4</td>
</tr>
<tr>
<td>PF24</td>
<td>d5</td>
</tr>
<tr>
<td>Cursor Left</td>
<td>Cursor Left</td>
</tr>
<tr>
<td>Cursor Right</td>
<td>Cursor Right</td>
</tr>
<tr>
<td>Cursor Up</td>
<td>Cursor Up</td>
</tr>
<tr>
<td>Cursor Down</td>
<td>Cursor Down</td>
</tr>
<tr>
<td>HOME</td>
<td>Cursor Home</td>
</tr>
<tr>
<td>TAB</td>
<td>Next Field</td>
</tr>
<tr>
<td>Shift + TAB</td>
<td>Previous Field</td>
</tr>
<tr>
<td>Numpad + Enter</td>
<td>DFHENTER</td>
</tr>
<tr>
<td>Enter</td>
<td>New Line if handling Ctrl Key</td>
</tr>
<tr>
<td>Ctrl</td>
<td>DFHENTER if handling Ctrl key</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>do nothing if not handling Ctrl key</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>do nothing if handling Ctrl Key</td>
</tr>
<tr>
<td>Ctrl+Enter</td>
<td>New Line if not handling Ctrl Key</td>
</tr>
<tr>
<td>END</td>
<td>Erase end-of-screen</td>
</tr>
<tr>
<td>Shift + END</td>
<td>Erase all</td>
</tr>
<tr>
<td>Shift + Delete</td>
<td>Erase end-of-field / Erase field</td>
</tr>
<tr>
<td>Backspace</td>
<td>Backspace</td>
</tr>
<tr>
<td>Delete</td>
<td>Cancel</td>
</tr>
<tr>
<td>Ins</td>
<td>Toggle insert</td>
</tr>
<tr>
<td>Key Combination</td>
<td>Function</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
</tr>
<tr>
<td>ESC / Pause</td>
<td>DFHCLEAR</td>
</tr>
<tr>
<td>Pg Down</td>
<td>PA1</td>
</tr>
<tr>
<td>Shift + Pg Up</td>
<td>PA2</td>
</tr>
<tr>
<td>Pg Up</td>
<td>PA3</td>
</tr>
<tr>
<td>F1</td>
<td>PF1</td>
</tr>
<tr>
<td>F2</td>
<td>PF2</td>
</tr>
<tr>
<td>F3</td>
<td>PF3</td>
</tr>
<tr>
<td>F4</td>
<td>PF4</td>
</tr>
<tr>
<td>F5</td>
<td>PF5</td>
</tr>
<tr>
<td>F6</td>
<td>PF6</td>
</tr>
<tr>
<td>F7</td>
<td>PF7</td>
</tr>
<tr>
<td>F8</td>
<td>PF8</td>
</tr>
<tr>
<td>F9</td>
<td>PF9</td>
</tr>
<tr>
<td>F10</td>
<td>PF10</td>
</tr>
<tr>
<td>F11</td>
<td>PF11</td>
</tr>
<tr>
<td>F12</td>
<td>PF12</td>
</tr>
<tr>
<td>Shift + F1</td>
<td>PF13</td>
</tr>
<tr>
<td>Shift + F2</td>
<td>PF14</td>
</tr>
<tr>
<td>Shift + F3</td>
<td>PF15</td>
</tr>
<tr>
<td>Shift + F4</td>
<td>PF16</td>
</tr>
<tr>
<td>Shift + F5</td>
<td>PF17</td>
</tr>
<tr>
<td>Shift + F6</td>
<td>PF18</td>
</tr>
<tr>
<td>Shift + F7</td>
<td>PF19</td>
</tr>
<tr>
<td>Shift + F8</td>
<td>PF20</td>
</tr>
<tr>
<td>Shift + F9</td>
<td>PF21</td>
</tr>
<tr>
<td>Shift + F10</td>
<td>PF22</td>
</tr>
<tr>
<td>Shift + F11</td>
<td>PF23</td>
</tr>
</tbody>
</table>
Shift + F12  PF24
4.11 Running X4J

X4J can be run on each system supporting JAVA 2 (1.4.x or higher)

Java can be obtained for free from Sun Microsystems at http://java.sun.com.

The following sections describe the raw startup of X4J from command line, which is obsolete. WebStart feature have to be used instead (see WebStart section).

Unix systems

On Unix System, X4J can be started using 'x4j.sh'.

Usage

```
java -jar x4j.jar [session]
```

If session parameter is supplied, X4J tries to attach the session specified (must be defined in x4j.ses).

Windows system

On Windows systems use:

```
java -jar x4j.jar
```

On Windows systems normally .jar files are started using current VM, so double-clicking the files shold be enough to start up X4J.

If session parameter is supplied, X4J tries to attach the session specified (must be defined in x4j.ses).

Other systems

On other systems (such MacOS) use:

```
java [options] -jar x4j.jar [session]
```

If session parameter is supplied, X4J tries to attach the session specified (must be defined in x4j.ses).
4.12 Using Java WebStart

X4J is deployed in a format compatible with the Java WebStart Technology (tm).

With Java Web Start, user launches applications simply by clicking on a Web page link. If the application is not present on your computer, Java Web Start automatically downloads all necessary files. It then caches the files on local computer so the application is always ready to be relaunched anytime you want — either from an icon on your desktop or from the browser link.

Moreover if application files are out-of-date, Java Web Start automatically provides application updating by downloading newer files.

Requirements

The client machine requires support for the Java Runtime Environment (JRE), version 1.3 or later. Java Web Start (tm) is available for Windows 95/98/NT/2000, Solaris (SPARC & Intel editions), and Linux (RH6.1/x86).

Applications can be deployed from any standard Web server. In order to use Java Web Start (tm), the Web server must be configured with support for a new MIME type as explained below.

Installing on a J2EE Web Server

XFRAME is deployed with a J2EE Web Application that can be used to deploy the X4J via Java Webstart, without having to configure anything. The only thing to do is to create a context in the J2EE WebServer hosting the web application located in $XFRAMEHOME/xje/webapps/webstart.

Installing on Tomcat 4.1

To install the deployment application on Tomcat 4.1 simply add in the Tomcat webapps directory ($CATALINA_HOME/webapps by default) an XML file named as the context you are going to use (i.e. x4j.xml) containing:

```xml
<Context path="< CONTEXT_NAME >" docBase="< XFRAMEHOME >/xje/webapps/webstart"
        debug="0" reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
           prefix="localhost.xframe.log.
                   suffix=".txt"
           timestamp="true"
        />
</Context>
```

replacing the CONTEXT_NAME with the real context name, and XFRAMEHOME with the real XFRAMEInstallation path ($XFRAMEHOME).

Once configured the XML file and restarted the web server simply connect to http://your.tomcat.web.server/CONTEXT_NAME/x4j.jnlp (where CONTEXT_NAME is the name specified above) to start X4J.

Installing on a Standard Web Server

To make use of Java Web Start (tm) technology with X4J simply follow this step:

- configure the Web server so that all files with the .jnlp file extension are set to the application/x-java-jnlp-file MIME type
- update the supplied JNLP files (x4j.jnlp and jh.jnlp) with the correct codebase information
- make the application accessible on the Web server by uploading application's JARs, JNLP and GIF files, at the URLs listed in the JNLP file.
- Create a link from the Web page to the JNLP file. For example:
This first step is critical for making sure that a Web browser will launch Java Web Start when downloading the file.

A link to the Java Web Start installer should also be provided on the Web page, so users who do not already have Java Web Start installed can download and install the software.

Further information

For further information on Java Web Start (tm) please refer to: http://java.sun.com/products/javawebstart.
4.13 Session and settings deployment

X4J is capable to download a set of predefined session and settings from a remote source. The settings and sessions are defined by the system administrators and made available through the XFRAME Web Deployment application.

This web application is useful for:

- automate the distribution of X4J using Java (tm) WebStart Technology
- centralize the definition of common settings
- automate the definition and distribution of X4J sessions

Furthermore, an interactive web application is available for administrators to easily analyze and reload the configuration.

**Installing the web deployment application**

To install the XFRAME Web Deployment Application a Java Web Server is required.

This documentation assumes you are using Tomcat 4.1 as Java Web Server.

Once installed Tomcat and XFRAME, a dedicated context must be defined. To do this simply add to your $CATALINA_HOME/webapps directory a file named webstart.xml (this name may change) containing:

```xml
<Context path="/webstart" docBase="/xframe/xje/webapps/webstart" debug="false" reloadable="true">
  <Log ...
  </Context>
```

- /webstart is the name of the context
- /xframe is the path to xframe installation
- /home/user/sessions.xml is the configuration file

Then, you have to define the role and users allowed to access the administrative pages. Role must be named "x4jdeploy" and users must belong to the role named "x4jdeploy".

By default Tomcat configures users in the file $CATALINA_HOME/conf/tomcat-users.xml.

I.e.

```xml
<tomcat-users>
  <user username="root" password="secret" roles="x4jdeploy"/>
</tomcat-users>
```

Once done you may start up Tomcat and let normal user download X4J directly from the "/webstart/x4j.jnlp" page (i.e. http://your.tomcat.web.server/webstart/x4j.jnlp). Administrators may connect directly to the administrative pages opening the "/webstart/index.jsp" (i.e. http://your.tomcat.web.server/webstart/index.jsp).
Configuring the XFRAME Web Deployment application

To configure XFRAME Web Deployment application the file defined by the x4.deployment.configuration.file parameter (/home/user/sessions.xml in the example above) must be edited.

Format of the configuration file:

```
<configuration name="">
  <settings>
    <master ( file="" | uri="" ) />
    <local ( file="" | uri="" ) />
    <keyboard ( file="" | uri="" ) />
    <macro ( file="" | uri="" ) />
    <base uri="" />
  </settings>
  <application name="" address="" port="" description="">
    <terminal name="" [ autosession="" ] [ user="" | address="" ] [ workstation="" ] [ windomain="" ] [ password="" ]
  </application>
</configuration>
```

More than one application may be defined and more than one terminal per application may be defined.

**The settings element**

In the settings element following items may be configured:

<table>
<thead>
<tr>
<th>master</th>
<th>the master settings source</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>the local settings source</td>
</tr>
<tr>
<td>keyboard</td>
<td>the keyboard definition source</td>
</tr>
<tr>
<td>macro</td>
<td>the macro definition source</td>
</tr>
<tr>
<td>base</td>
<td>the settings base</td>
</tr>
</tbody>
</table>

If an item is defined as URI (Uniform Resource Indicator) the X4J will provide to read/download it automatically. If the item is defined as file the XFRAME Web Application will transfer it.

**The application element**

Each application element defines an application entry, and requires:

<table>
<thead>
<tr>
<th>name</th>
<th>application name</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>the internet address of the machine hosting the XCICS application</td>
</tr>
<tr>
<td>port</td>
<td>the port number for the XCICS application</td>
</tr>
</tbody>
</table>
Each application may contain one or more terminal entry.

The terminal element

Each terminal element defines a session assignment rule.

<table>
<thead>
<tr>
<th>name</th>
<th>terminal name</th>
</tr>
</thead>
<tbody>
<tr>
<td>autosession</td>
<td>auto session number (1-4)</td>
</tr>
<tr>
<td>user</td>
<td>user name</td>
</tr>
<tr>
<td>address</td>
<td>host name of the client machine</td>
</tr>
<tr>
<td>workstation</td>
<td>workstation name of the client machine</td>
</tr>
<tr>
<td>windomain</td>
<td>the Windows domain name (available only win32 system)</td>
</tr>
<tr>
<td>password</td>
<td>additional password</td>
</tr>
</tbody>
</table>

An example

```xml
<configuration name="x4j">
  <settings>
    <!-- master file is a local file on the web server -->
    <!-- master file="/home/webstart/master.properties"/>
    <!-- local settings are stored in the C:/Settings directory for each workstation -->
    <settings uri="/c:/settings/x4j.rc"/>
    <!-- keyboard is downloaded from another web server -->
    <keyboard uri="http://home.intra.net/files/x4j.kbd"/>
  </settings>
  <!-- definitions for the test application on the host test.intra.net at port 40000 -->
  <application name="test" address="test.intra.net" port="40000">
    <description>Test application</description>
    <!-- user01 has a session USR1 wherever he logs on -->
    <terminal name="USR1" user="user01"/>
    <terminal name="USR2" user="user02"/>
    <!-- MO0 is for workstation wks.intra.net -->
    <terminal name="MO0" address="wks1.intra.net"/>
    <terminal name="MO0" address="wks2.intra.net"/>
    <!-- the workstation W001 uses N00 -->
    <terminal name="N00" workstation="W001"/>
    <!-- every one has an "auto session 1" -->
    <terminal autosession="1"/>
  </application>
  <application name="production" address="as.intra.net" port="40000">
    <description>Production app</description>
    <terminal name="USR1" user="user01"/>
    <terminal name="USR2" user="user02"/>
  </application>
</configuration>
```
<terminal name="MC00" address="wks1.intra.net"/>
<terminal autosession="1"/>
<terminal autosession="2"/>
<!-- if user is root a third session is allowed -->
<terminal autosession="3" user="root"/>
</application>
</configuration>
Part V

XVSAM File System
5 XVSAM File System

This book contains the guidance for XVSAM file system. It contains information on concepts, administration and programming of XVSAM.

The XVSAM (XFRAME Virtual Storage Access Method) file system provides on Unix/Linux systems indexed and non-indexed file access, with the same features of VSAM access method of the mainframe environments.

The most important features of XVSAM are:

- indexed file organization
- relative file organization
- sequential file organization
- alternate indexes
- catalogs
- dispositions
- warm start capabilities
- recovery capabilities (after-image)

XVSAM is based on the IBM Informix C-ISAM interface, and it produces exactly one C-ISAM file for each user XVSAM data set.
5.1 Concepts

XVSAM (XFRAME Virtual Storage Access Method) is a file access method to emulate the features of the IBM VSAM on UNIX/Linux systems.

Using XVSAM, you may organize and access records in a file in physical sequence (the sequential order that they were entered), logical sequence using a key, or by the relative record number on direct access storage devices.

XVSAM organizes the files in repositories. An XVSAM repository is composed by a master catalog and by the set of files belonging to the repository.

The repository is logically divided into catalogs, and each catalog contains one or more cluster. The cluster is the XVSAM file entity. i.e.

<table>
<thead>
<tr>
<th>XVSAM-repository</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ CATALOG 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+ CATALOG 2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Each XVSAM file (AKA dataset or cluster), even if physically handled with two files (data and index), is handled by the user applications as a single entity. XVSAM stores the all information about the files in the master catalog, which is file directly managed by the XVSAM subsystem.

XVSAM files are identified by the following characteristics, that must be defined in the creation phase.

CLUSTER NAME

The cluster is the physical name of the XVSAM file. In conjunction with the catalog name, it identifies univocally the files in the XVSAM repository.

The cluster name cannot exceed 44 characters and is univocal in a catalog.

CATALOG NAME

This is the name of the XVSAM catalog that contains the file.

TYPE

There are three types of VSAM data sets:

- KSDS - Key Sequenced Data Set
- ESDS - Entry Sequenced Data Set
- RRDS - Relative Record Data Set

RECORD FORMAT AND LENGTH

XVSAM stores the record length of the files in its master catalog. This length may be fixed or variable. If the length is variable both minimum and maximum length must be defined at creation time.

KEY

For KSDS files, XVSAM stores the offset and the length of the key. Unlike IBM VSAM, the XVSAM key may be composed by multiple part of the record (8 part maximum).
5.1.1 File types

**KSDS - Key Sequenced Data Set**

A key-sequenced data set is a data set in which each record is identified by means of a key. The key of any record is stored as a field in a predefined position as part of the record.

Each key value must be unique in the data set. When the data set is initially loaded with data and when new records are added, the physical order of the records is determined by the collating sequence of the key field.

This also determines the order in which records are retrieved when browsing through the data set.

To enable XVSAM to determine the physical location of a record in a KSDS file, it creates and maintains an index which relates the key of each record with the record’s relative location in the data set.

Whenever a record is added or deleted, the index is updated to reflect the change.

**ESDS - Entry Sequenced Data Set**

An entry-sequenced data set is a data set in which each record is identified by its relative byte address.

Records are stored in an ESDS file in the order in which they are initially loaded into the data set. Further records added to the file are always stored after the last record in the data set.

Records may not be deleted from an ESDS file, nor may their length be altered. After a record has been stored in an ESDS file, its RBA will remain unchanged.

When browsing through an ESDS file, records are retrieved in the order in which they were added to the data set.

**RRDS - Relative Record Data Set**

A record in an RRDS file is identified by the relative record number of the slot in which it is stored.

When a new record is added to an RRDS file, XVSAM assigns the next sequential number in the data set or the number supplied with the request.

Unlike records in a KSDS or an ESDS, records in an RRDS data set must be of fixed length.

In order to maintain information necessary to access RRDS or ESDS files, XVSAM adds to each record a ten bytes long header, where the RRN or the RBA are stored. These area is not returned to application programs that use the file.
5.1.2 Alternate indexes

To access records using an alternate key an alternate index file must be defined. This file behaves like a key-sequenced data set in which records are accessed using the specified alternate key.

An alternate key is defined in the same way as the primary key in a KSDS, as a field with fixed length and fixed position within the record. Any number of alternate keys may be defined and, unlike the base or primary key, an alternate key need not to have a unique value.

XVSAM allows alternate indexes to be defined for KSDS files only at any time. The alternate index must be defined in the same catalog of its primary cluster.

Unlike IBM VSAM, where three entities are involved in a alternate index relationship, being they the base cluster, the alternate index and the path, XVSAM simply needs the base cluster and alternate index.
5.1.3 Configuration

XVSAM is configured using the environment variable named "XVSAM". This variable indicates the directory where the repository is stored and the user defined catalogs are located.

XVSAM can be used from programs running in every directory of the UNIX/Linux file system, just setting before the execution the XVSAM environment variable as follows:

```bash
export XVSAM = <repository full path>
```

and after having checked to have sufficient UNIX rights on the declared directory.

If the repository directory pointed by the XVSAM environment variable is not initialized, the first file creation command will initialize it, creating the master catalog files (Catalog.dat and Catalog.idx).
5.1.4 File names conventions

The name of the physical files that compose a single XVSAM file, are composed by CLUSTER.NAME.dat and CLUSTER.NAME.idx.

By default XVSAM stores the data set file in a sub-directory of the XVSAM Master Repository ($XVSAM) which name is equal to the catalog name. I.e.

$XVSAM/<CATALOG NAME>

If desired, you may specify an alternative path during the creation of the data set where the data and index files will be stored.
5.1.5 Storage management systems

XVSAM offers the possibility to use different storage systems:

- isam
- managed-isam
- database

**ISAM**

The ISAM storage management is the default one: XVSAM files are directly stored on ISAM files, without any further intervention.

**Managed ISAM**

Files are stored on the ISAM files but, before output operation or after input operation, the record is passed to an user program which may alter its content.

**Database**

Files are stored on a relational database system (i.e. Oracle): each file is contained on a table, and indexes/alternate indexes are reproduced by means of database indexing features.

This storage system is also know as V2R (VSAM 2 Relational).
5.1.6 Transactions management and recovery

XVSAM has both transaction management and data integrity protection capabilities. XCICS/TS takes advantage of these features to offer those data safety features mandatory for a TP monitor.

XVSAM may track all the performed operation on its recovery log files.

The recovery log is used to perform:

- Warm start
- After-Image processing

For safety reasons, it's better to keep this file not on the same disk where XVSAM data are stored.

Warm start

The warm start feature is very important for XCICS/TS: at warm start time, XVSAM reads the existing recovery log, and backs-out all the "in-flight" transactions of the previous instance. In other words it removes all uncommitted transaction from files.

Warm start processing on XVSAM files may be also activated by command line using the xvsam utility (--warm-start option).

After-Image processing

On some situations, i.e. serious hardware faults, it is necessary to rebuild the XVSAM information starting from a backup.

With the after-image processing, it is possible to restore an existing backup and use the XCICS recovery log to apply all changes occurred on files tracked in the log.

In this case, user have to:

1. restore the latest XVSAM backup
2. re-apply the changes tracked on files using the xvsam utility (--recover option)

Of course, backup data and recovery log must be congruent.
5.2 Using XVSAM

XVSAM files may be used and accessed from different interfaces:

- Unix/Linux command line and shell scripts
- Interactively
- Batch programs (COBOL & PLI)
- XCICS/TS on-line programs

Command line and shell scripts

XVSAM files may be managed directly from the command line or in shell scripts using the xvsam utility. This tool allows performing all the management operations required to administer and operate with XVSAM files.

I.e.

```
xvsam --create -C SAMPLE --type=ksds --record-length=30 --key=0,6
XVSAM SAMPLE
```

For further informations, please refer to the Command line utilities Section.

Interactive Mode

XVSAM files may be managed directly from an interactive mode by the tool xvsamAdm.

For further informations, please refer to the xvsamAdm Section.

Batch programs

XVSAM files may be access and used by batch programs written in COBOL or PL/I. Like on the mainframe side each file is identified by the program with two names: internal and external name. The internal name is the one used by the program to logically identify the file in the code and it is the one used in the program’s I/O statements. The external name is the one used by the program to identify the physical file.

The link between the file external name and the physical file is provided by the XVSAM run-time system, which binds the external name with the value of a corresponding environment variable.

I.e. (csh)

```
setenv dlbl_MYFILE "MY.CLUSTER, cat=MYCAT, disp=(OLD, KEEP), type=vsam"
```

I.e. (sh/ksh)

```
eexport dlbl_MYFILE="MY.CLUSTER, cat=MYCAT, disp=(OLD, KEEP), type=vsam"
```

ESDS files

XVSAM ESDS files are identified from COBOL by the ORGANIZATION IS SEQUENTIAL clause. You can access (read or write) their records only sequentially.

ESDS files cannot be accessed from PL/I.

I.e.

COBOL:
PL/I: Not supported.

KSDS

XVSAM KSDS files are identified from COBOL by the ORGANIZATION IS INDEXED clause. The primary key for file is coded using the RECORD KEY clause. You can also use alternate keys and an alternate indexes.

I.e.

COBOL:

```
SELECT I-FILE ASSIGN TO myfile
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS FILE-RECORD-KEY
    FILE STATUS IS FS-CODE.
```

PL/I:

```
DCL I-FILE FILE RECORD INPUT KEYED ENV(VSAM);
```

RRDS

VSAM RRDS files are identified from COBOL by the ORGANIZATION IS RELATIVE clause. The RELATIVE KEY IS clause is used to associate each logical record with its relative record number.

RRDS files cannot be accessed from PL/I.

I.e.

COBOL:

```
SELECT R-FILE ASSIGN TO myfile
    ORGANIZATION IS RELATIVE
    ACCESS IS RANDOM
    RELATIVE KEY IS FILE-RELATIVE-KEY
    FILE STATUS IS FS-CODE.
```

PL/I: Not supported.

Alternate indexes

The alternate indexes are defined by the xvsam --create command and the --aix option.

In the COBOL program the alternate index must be declared as the primary. It is not possible to declare two different indexes for the same file in the same program.

I.e.
SELECT I-FILE ASSIGN TO INDEXED-FILE
    ORGANIZATION IS INDEXED
    ACCESS IS DYNAMIC
    RECORD KEY IS ALTERNATE-KEY
    FILE STATUS IS FS-CODE.

**XCICS/TS On-line programs**

Each XVSAM file to be used by the programs running in the XCICS/TS region, must be define in the XCICS configuration file, specifying the dataset name (name referenced by the programs), the cluster and catalog names, the access and logging mode.

I.e.

```sql
define dataset name=MYDATS, cluster=XVSAM.SAMPLE, catalog=SAMPLE, mode=updt, logging=on;
```

Programs will access the XVSAM files by means of the standard EXEC CICS APIs. XCICS/TS will take care about the LUW management and files integrity.

I.e.

```sql
EXEC CICS READ DATASET('MYDATS')
    LENGTH (WS-LENGTH)
    INTO  (WS-REC)
    RIDFLD  (WS-KEY)
    UPDATE
END-EXEC.
```
### 5.2.1 Functional commands

The XVSAM functional commands are:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Create</strong></td>
<td>to create catalogs, clusters and alternate indexes</td>
</tr>
<tr>
<td><strong>Delete</strong></td>
<td>to delete clusters</td>
</tr>
<tr>
<td><strong>Erase</strong></td>
<td>to remove all records from a cluster</td>
</tr>
<tr>
<td><strong>List</strong></td>
<td>to list catalogs and clusters</td>
</tr>
<tr>
<td><strong>Export</strong></td>
<td>to write the content of an XVSAM file into a sequential file or clusters</td>
</tr>
<tr>
<td><strong>Import</strong></td>
<td>to load the data contained in a sequential file into an XVSAM data set</td>
</tr>
<tr>
<td><strong>Repro</strong></td>
<td>to perform simple or complex copies of the XVSAM data sets with format conversion, record and key exclusion</td>
</tr>
<tr>
<td><strong>Check</strong></td>
<td>to verify the integrity of an XVSAM file</td>
</tr>
<tr>
<td><strong>Recover</strong></td>
<td>to perform an after-image processing, re-applying all the changes tracked in the RECOVERY.log on a backup copy of the data set</td>
</tr>
<tr>
<td><strong>Warm-start</strong></td>
<td>to remove all the changes caused by the uncommitted transactions after a system crash</td>
</tr>
</tbody>
</table>
5.2.2 Creating a file

You can use XVSAM files only after you define them through access method services.

Use the `xvsam --create` command to create XVSAM data sets.

The following information are required to create a cluster:

- Name of the cluster
- Name of the catalog
- Type (esds, ksds, or rrds)
- Record length
- Key definition (KSDS only)

XVSAM files may be defined using interactively with `xvsamAdm` or from scripts and command line using `xvsam`.

I.e.

```
# xvsam --create -C MYCATALOG --type=ksds --record-length=30 --key=0,6 MYCLUSTER
```

This command creates a KSDS file name MYCLUSTER in the catalog MYCATALOG. The length of the record is 30 with a primary key starting at offset 0 with 6 bytes length. The files MYCLUSTER.dat and MYCLUSTER.idx are created in the path `$XVSAM/MYCATLOG`

If the physical files of the cluster must be created in a non default path, the option `--path` must be used. I.e.

```
# xvsam --create -C MYCATALOG --type=ksds --record-length=30 --key=0,6 --path=/home/my/data MYCLUSTER
```

This command creates the XVSAM file entry and generates MYCLUSTER.dat and MYCLUSTER.idx in the directory `home/my/data/MYCATLOG`.

Once a cluster has been create an alternate index may also be defined, with `--aix` option. I.e.:

```
# xvsam --create --aix -C MYCATALOG --key=10,4 --duplicate --primary=MYCLUSTER MY. ALTERNATE.INDEX
```

This creates an alternate index named MY.ALTERNATE.INDEX on the KSDS file MYCLUSTER with an alternate key beginning at offset 10 and ending at offset 14. The option `--duplicate` instruct XVSAM to accept duplicate values for the alternate key.
## 5.2.3 Browsing the repository

The option `--list` of the `xvsam` command gets a list of the catalogs and the relative clusters. I.e.

```
# xvsam --list
<table>
<thead>
<tr>
<th>catalog</th>
<th>cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE.AIX</td>
</tr>
<tr>
<td>SAMPLE1</td>
<td>XVSAM SAMPLE1</td>
</tr>
</tbody>
</table>
```

The option `--logical-info` shows the logical informations of the files, like type, format, record length, logical name, the duplicate option and the position of the key.

```
# xvsam --list --logical-info
<table>
<thead>
<tr>
<th>catalog</th>
<th>cluster</th>
<th>type</th>
<th>fmt</th>
<th>reclen</th>
<th>logical</th>
<th>dups</th>
<th>key(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE</td>
<td>ksds</td>
<td>fixed</td>
<td>80</td>
<td>LOGICAL</td>
<td>no</td>
<td>(0,80)</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE.AIX</td>
<td>aix</td>
<td>-</td>
<td>80</td>
<td>no</td>
<td>(20,6)</td>
<td></td>
</tr>
</tbody>
</table>
```

The option `--physical-info` shows the physical informations of the files, as the storage, the number of the records, and the path the physical files (.dat and .idx).

```
# xvsam --list --physical-info
<table>
<thead>
<tr>
<th>catalog</th>
<th>cluster</th>
<th>storage</th>
<th>records</th>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE</td>
<td>isam</td>
<td>0</td>
<td>SAMPLE/</td>
</tr>
<tr>
<td>SAMPLE</td>
<td>XVSAM SAMPLE.AIX</td>
<td>isam</td>
<td>0</td>
<td>SAMPLE/</td>
</tr>
</tbody>
</table>
```

The `--list-catalog` option is used to list the catalog names and the number of the clusters contained.

```
# xvsam --list-catalog
<table>
<thead>
<tr>
<th>catalog</th>
<th>entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLE</td>
<td>2</td>
</tr>
<tr>
<td>SAMPLE1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
5.2.4 Erasing and deleting

The option `--erase` of the `xvsam` command remove the content of a data set. The entry is not removed from the XVSAM master catalog and the physical files are left on the file system. Only the records stored are deleted. I.e.

```
xvsam --erase --catalog SAMPLE XVSAM SAMPLE
```

The option `--delete` of the `xvsam` command deletes a data set. The entry is removed from the XVSAM master catalog and the physical files are removed from the file system. I.e.

```
xvsam --delete --catalog SAMPLE XVSAM SAMPLE
```

Deleting a KSDS files also removes any eventual alternate index referring to the cluster removed.
5.2.5 Import and Export

XVSAM files may be imported from or exported to any sequential file. Also the master catalog definitions may be exported and re-imported in a different repository.

**Importing a file**

The option `--import` of the `xvsam` command loads the content of an XVSAM dataset from a sequential file. I.e.

```bash
# xvsam --import -C SAMPLE --input /my/sam/FLAT.SAMPLE XVSAM.SAMPLE
```

The command loads the content of the file `/my/sam/FLAT.SAMPLE` into the cluster `XVSAM.SAMPLE`. The sequential file must have the same record length of the cluster.

Also line sequential files may be imported using the option `--stdin`. In this case every line in the input file is handled as a single record and padded if necessary. I.e.

```bash
# xvsam --import -C SAMPLE --stdin XVSAM.SAMPLE < /my/lsam/FLAT.SAMPLE.txt XVSAM
```

**Exporting a file**

The option `--export` of the `xvsam` command stores the content of an XVSAM dataset in a sequential file. I.e.

```bash
# xvsam --export -C SAMPLE --output /my/sam/FLAT.SAMPLE XVSAM.SAMPLE
```

The command writes the content of the cluster `XVSAM.SAMPLE` file into the file `/my/sam/FLAT.SAMPLE`. The sequential file will have the same record length of the cluster.

**Importing and Exporting the master catalog**

The options `--export-catalog` and `--import-catalog` of the `xvsam` command may be used to export and import the file definitions stored in an XVSAM repository.

I.e.

```bash
# xvsam --export-catalog > catalog.txt
# xvsam --import-catalog < catalog.txt
```

Please note that the import and export catalog options do not export or import the content of the XVSAM files.
5.2.6 Repro

The **--repro** option of the **xvsam** command may be used to perform many types of manipulation between XVSAM and sequential files, allowing to:

- Convert the record format
- Skip or include only a certain amount of records
- Copy records which key is contained in a predefined range
- Initialize the first record

These are some samples of file manipulation allowed by the repro option.

Copy of an XVSAM file to another XVSAM file:

```
xvsam --repro --input-catalog IN.SAMPLE --input-cluster IN.XVSAM.SAMPLE\
        --output-catalog OUT.SAMPLE --output-cluster OUT.XVSAM.SAMPLE
```

Copy of an XVSAM file to another XVSAM file with external file definition:

```
setenv dlb_INDD "MY.INPUT.CLUSTER,cat=CAT1,disp=(OLD,DELETE)"
setenv dlb_OUTDD "MY.OUTPUT.CLUSTER,cat=CAT2,disp=(OLD,KEEP)"
xvsam --repro --dd-in INDD --dd-out OUTDD
```

Copy of sequential file into another sequential file with record format conversion:

```
xvsam --repro --input-file /home/data/FIXED.SAMPLE\
        --output-file /home/data/VARIABLE.SAMPLE\
        --output-format var --input-record-length 80
```

Copy of an XVSAM dataset into a sequential file with line sequential record format:

```
xvsam --repro --input-catalog SAMPLE --input-cluster XVSAM.SAMPLE\
        --output-format lineseq\
        --output-file /home/data/LINE.SAMPLE
```

Copy of a sequential file with variable record length into an XVSAM file, replacing existing key with the new one:

```
xvsam --repro --replace \
        --input-file /home/data/VAR.SAMPLE --input-format var \
        --output-cluster XVSAM.SAMPLE --output-catalog SAMPLE
```
5.2.7 Warm start and recovery

XVSAM has both transaction management and data integrity protection capabilities. XCICS/TS takes advantage of these features to offer those data safety features mandatory for a TP monitor.

XVSAM may track all the performed operation on its recovery log files.

The recovery log is used to perform:

- Warm start
- After-Image processing (Recovery)

For safety reasons, it's better to keep not this file on the same disk where XVSAM data are stored.

**Warm start**

The warm start feature is very important for XCICS/TS: at warm start time, XVSAM reads the existing recovery log, and backs-out all the "in-flight" transactions of the previous instance. In other words it removes all uncommitted transaction from files.

The warm start may be automatically activated by XCICS/TS starting it up with in warmstart mode.

Warm start processing on XVSAM files may be also activated by command line using the `xvsam` utility (\(--\text{warm-start}\) option). I.e.

```
# xvsam --warm-start --log home/my/data/RECOVERY.LOG
```

**After-Image processing (Recovery)**

On some situations, i.e. serious hardware faults, it is necessary to rebuild the XVSAM informations starting from a backup.

XVSAM Recovery is the process of regaining access to lost XVSAM data. It is used to rebuild data after it has been damaged or destroyed. It is the restoration of resources following an error.

With the after-image processing, it is possible to restore an existing backup and use the XCICS recovery log to apply all changes occurred on files tracked in the log.

In this case, user have to:

1. restore the latest XVSAM backup
2. re-apply the changes occurred on files using the `xvsam` utility (\(--\text{recover}\) option)

Of course, backup data and recovery log must be congruent.

```
xvsam --recover
```
5.2.8 Backup and restore

There are two ways to backup/restore the XVSAM files and/or repository

**Physical repository backup**

You may backup/restore the whole XVSAM repository using the standard system features.

I.e.

```
# tar cvf xvsam-backup.tar $XVSAM
```

**Single file physical backup**

You may backup a specific file entry using the standard system features to backup the physical files of the desired XVSAM cluster.

To restore these files, you must check that the corresponding entry exists in the master catalog and then restore them as any normal file.

I.e.

Backup phase on system 1:

```
# xvsam --list --logical-info
catalog  cluster                                      type fmt      reclen
logical  dupes  key(s)                                  ________________  _______
PRMPRD90 FFFFFF                                    rrdc fixed    33
PRMPRD90 no                                        rrdc fixed    33
PRMPRD90 XXXX                                     ksds fixed    25
PRMPRD90 ZASIC030, CICS, C30HER                    ksds fixed    25
PRMPRD90 no                                       ksds fixed    33
PRMPRD90 erere                                    rrdc fixed    33
PRMPRD90 no                                       rrdc fixed    33

# cd $XVSAM
# tar cvf C30HER.tar PRMPRD90/ZASIC030, CICS, C30HER.dat
# tar cvf C30HER.idx
```

Restore phase on system 2:

```
# xvsam --create --catalog PRMPRD90 --type=ksds --record-length=25 --key=0,10
ZASIC030, CICS, C30HER
# cd $XVSAM
# tar xvf C30HER.tar
```

**Single file content backup**

You may backup a specific file entry exporting the content of the XVSAM dataset on a sequential file with the export feature, and then re-import it.

I.e.
Backup phase on system 1:

```
# xvsam --list --logical-info
catalog     cluster          logical  type fmt  reclen logical
----------------------- ------ ------- ------- ------- ------- -------
PRMPRD90     YYYY      no          erere   rdds fixed  33         
PRMPRD90     XXXX      no          erere   rdds fixed  25         
PRMPRD90     ZASIC030.CICS.C30HER  no (0,10) rdds fixed  25         
C30HER       no (0,10) rdds fixed  33 weeeeee  
PRMPRD90     erere      no          erere   rdds fixed  3443         
PRMPRD90     erere      no          erere   rdds fixed  3443         
# xvsam --export --catalog PRMPRD90 --output C30HER.sam ZASIC030.CICS.C30HER
```

Restore phase on system 2:

```
# xvsam --create --catalog PRMPRD90 --type=kdsd --record-length=25 --key=0,10 ZASIC030.CICS.C30HER
# xvsam --import --catalog PRMPRD90 --output C30HER.sam ZASIC030.CICS.C30HER
```
5.3 Utilities

5.3.1 Command line utilities

5.3.1.1 xvsam

xvsam is a command line utility that groups all the features provided by all the other XVSAM utilities. xvsam can perform many operations on an XVSAM file or on the XVSAM catalog, and in particular:

- create
- delete & erase
- import/export
- check
- catalog listing
- catalog import/export
- warm start and recovery functions

Syntax

```
xvsam [command] [options] <cluster>
```

Commands

```
--version, -V          show version
--create, -c           creates an XVSAM file
--delete, -d           deletes an XVSAM file
--erase, -e            erases an XVSAM file
--list, -l             lists the catalog entries
--list-catalog, -N     lists the catalogs
--import, -i           imports a flat file on an XVSAM file
--export, -e           exports an XVSAM file to a flat file
--repro, -R            performs REPRO-like file copy processing
--export-catalog, -X   exports the catalog entries
--import-catalog, -I   imports the catalog entries
--check, -k            checks the integrity of an XVSAM file
--recover, -r          recovers an XVSAM repository from a log file
--warm-start, -w       removes uncommitted on-flight transactions from an XVSAM repository
--help, -?             show the online help
```

Parameters for `--create`

```
--catalog, -C          the catalog name (default MASTERCAT)
<catalog>
```
--aix creates an alternate index
--duplicate alternate index allows duplicate keys
--no-duplicate alternate index does not allow duplicate keys
--record-length <len> fixed record length
--min-record-length <len> minimum record length for variable record length files
--max-record-length <len> maximum record length for variable record length files
--key <keydef> adds a key definition (offset,length)
--path, -P <path> full path of the cluster files
--type, -t <type> file type (ksds, esds, rrd)
--logical, -L logical name of the file
--storage-engine <type> defines the storage management system:

  * isam
  * managed-isam
  * database

--xml-fd <path> path for the XML file that describes the record structure (for database managed only files)

**Parameters for --list**

--logical-info shows logical information
--physical-info shows physical information

**Parameters for --delete and --erase**

--catalog, -C <catalog> the catalog name (default MASTERCAT)

**Parameters for --import**

--catalog, -C <catalog> the catalog name (default MASTERCAT)
--input <path> path of the flat file to be imported
--stdin import from STDIN
--replace replace existing keys during import
--text input file is line sequential (newline/0x0A record delimiter)
--binary input file is binary (no record delimiter)
--init <value> initialize a new record to the specified value. Value can either a char or an hexadecimal value, prefixed by 0x

**Parameters for --export**
Parameters for --repro

--input-catalog <value> input XVSAM catalog name (default MASTERCAT)

--input-cluster <value> input XVSAM cluster name

--dd-in <value> gets input file definition from DD environment variable (dbname)

--input-file <path> input flat file

--input-record-length <len> record length for fixed record input

--input-format <value> input file format (lineseq, fixed, var)

--output-catalog <value> input XVSAM catalog name (default MASTERCAT)

--output-cluster <path> output XVSAM cluster name

--output-file <path> output flat file

--dd-out <value> gets output file definition from DD environment variable (dbname)

--output-record-length <len> record length for fixed record output

--output-format <value> output file format (lineseq, fixed, var)

--stdin reads input from STDIN

--stdout writes output on STDOUT

--replace file replaces existing keys in XVSAM output

--skip <N> skips the first <N> records

--count <N> copies only <N> records

--init <value> initializes a new record to the
specified <value>. This can be either a char or an hexadecimal value, prefixed by 0x.

--from-key <value> copies records starting from <key>
--to-key <value> copies record up to <key>
--from-key-hex <value> copies record starting from hexadecimal <key>
--to-key-hex <value> copies record up to hexadecimal <key>
--warn-on-duplicate exits with positive RC if duplicate(s) found

Parameters for --recover
--log path to the XVSAM log (RECOVERY.LOG) file

Parameters for --warm-start
--log path to the XVSAM log (RECOVERY.LOG) file
--force forces the file fix-up even if errors are detected
--show-only transactions do not fix the file, only shows uncommitted
5.3.1.2 Backward compatibility

5.3.1.2.1 xvsamCreate

xvsamCreate creates the definition for a new base cluster in the catalog. It creates only a definition, so that data set remains empty until it has been filled up by means of the xvsamimport function. During the creation, the primary key and all the secondary key records are written in both files Catalog.dat and Catalog.idx, and the correspondent C-ISAM components - vsamfile.idx and vsamfile.dat are created and initialized too.

For this reason you are no longer requested to build up explicitly the alternate indices relative to a base cluster by means of the BLDINDEX feature of IDCAMS.

xvsamCreate reads the launch parameters from standard input, line after line until the end-of-data indicator is encountered or you interrupt it by means of <CTRL+D>. That allows you to group together the creation of more than one base cluster or alternate index path within a single xvsamCreate command.

Its functionality corresponds to those of the IDCAMS statements:

- DEFINE CLUSTER
- DEFINE ALTERNATE INDEX
- DEFINE PATH

Syntax

xvsamCreate

Standard input format for clusters is the following:

C <catalog> <NAME> <path> (K|R|E) (F|V) <min> <max> <nk> <koff> <klen> <name> <Y|N>

while for alternate indexes is:

A <catalog> <NAME> <primary> (D|U) <nk> <koff> <klen>

Parameters

<table>
<thead>
<tr>
<th>C</th>
<th>A</th>
<th>Function code, mandatory operand:</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>Define Cluster</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Define alternate index path</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Catalog</th>
<th>Name of the catalog</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Name of the cluster or the alternate index path, in uppercase, that is to be created</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>Physical path under $XVSAM, where the file is to be created</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Primary</th>
<th>Name of the associated base cluster; to be supplied only in case of Define Alternate index path</th>
</tr>
</thead>
</table>

| K|R|E | File organization: KSDS, RRDS or ESDS; to be supplied only in case of Define Cluster |
|---|---|-------------------------------------|

<table>
<thead>
<tr>
<th>F</th>
<th>V</th>
<th>Record type: Fixed or Variable record length; to be supplied only in case of Define Cluster</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Min</th>
<th>Minimum record size. If zero or nothing is entered for minimum record size, then xvsamCreate calculates it automatically, on the base of the key lengths (primary and secondary). This is due to the fact the C-ISAM puts all keys in the fixed part of the record. This value is required only in case of Define Cluster</th>
</tr>
</thead>
</table>
max   maximum record size; to be supplied only in case of Define Cluster
nk    number of parts composing the key
koff   key offset; it represents the displacement of the key field from the beginning of the record, relative to zero
klen   it represents the length of the key
name   name of the cluster or the alternate index path, in lowercase, that is to be created
Y|N    Reuse/No reuse flag. Y (yes) OPEN OUTPUT of non-empty files is allowed; N (no) returns an 712 error if an OPEN OUTPUT operation is started on a non-empty file. If this value is omitted, N is assumed by default
D|U    this parameter must only be specified in case of Define Alternate index path:
      D Duplicate key
      U Unique key

During the creation of a cluster or an alternate index path, you can inform xvsamCreate to produce a logging file containing the results of operations.

To do it, you should provide the <name> optional parameter. If this parameter is present, the utility creates a logging file with the name specified in <name>. If only -d is given, the logging file will be created in the same directory where xvsamCreate has been started, with the default name xvsamCreate.log.

The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the -level optional parameter.

For example, let us create a KSDS file named mycluster, belonging to the TESTCAT catalog and having fixed length records. The record size - minimum and maximum record length - is 38 bytes and the primary record key is 10 bytes long and starts from the beginning of the record. The syntax of the command for creating the file is the following:

```bash
# xvsamCreate
C TESTCAT MYCLUSTER TESTCAT K F 38 38 1 0 10 mycluster Y
<CTRL-D>
```

It is also possible to create more than one cluster at the same time as shown in the following script. In this example we intend to create one KSDS cluster - VSAMFILE - reusable, with variable length records and the path entries relative to two alternate indices.

We want also the run of the procedure to be logged in a protocol file named mydebug.log.

```bash
#!/bin/ksh
xvsamCreate -dmydebug.log <<<
C TESTCAT VSAMFILE TESTCAT K V 50 100 1 0 10 vsamfile Y
A TESTCAT ALTIND1 VSAMFILE D 1 5 10
A TESTCAT ALTIND2 VSAMFILE U 1 20 10
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>
5.3.1.2.2 xvsamDelete

xvsamDelete allows you to delete one or more XVSAM files. It removes the C-ISAM components from the UNIX file system and it deletes the corresponding entries from the specified catalog. If you have specified a base cluster name, also the associated alternate index path entries are deleted, if existing.

Syntax

```bash
# xvsamDelete [-f -d<filename> -l<debug level>] <catalog> <cluster>
```

Parameters

- `-f` force to produce no error if dataset doesn’t exist
- `catalog` is the name of the catalog the cluster belongs to
- `cluster` is the name of the cluster to be deleted

As already seen for xvsamCreate, a protocol file can be created during deletion, by specifying `-d<name>`. If no name is specified, the protocol file will be created as xvsamDelete.log. The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the `-l<level>` optional parameter.

WARNING: A cluster or an alternate index path MUST never be deleted through using the `rm` UNIX command, because of its indexation in Catalog.dat and Catalog.idx. To delete a cluster or an alternate index path, use always xvsamAdm or xvsamDelete!!!

Example:

```bash
xvsamDelete TESTCAT mycluster
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

5.3.1.2.3 xvsamErase

By means of xvsamDelete, catalog definitions and content of an XVSAM data set can be removed in one step.

xvsamErase allows you to delete only the content of the data file. The definition of the cluster remains unchanged in the catalog and it can be reused later.

Its functionality corresponds to those of the IDCAMS statements: DELETE CLUSTER PURGE

Syntax

```bash
xvsamErase [-d<filename>] <catalog> <cluster>
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f</td>
<td>force to produce no error if dataset doesn’t exist</td>
</tr>
<tr>
<td>-d&lt;filename&gt;</td>
<td>log file (if forced -d without filename xvsamDelete.log was created)</td>
</tr>
<tr>
<td>-l(0-15)</td>
<td>debugging level</td>
</tr>
<tr>
<td>catalog</td>
<td>is the name of the catalog the cluster belongs to</td>
</tr>
<tr>
<td>cluster</td>
<td>is the name of the cluster to be erased</td>
</tr>
</tbody>
</table>

A protocol file can be created during deletion, by specifying -d<name>. If no name is specified, the protocol file will be created as xvsamDelete.log.

The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the -l<level> optional parameter.

Example:

```
# xvsamErase TESTCAT SNICTHLP -d
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

5.3.1.2.4 xvsamImport

xvsamImport loads the content of a UNIX flat file into the XVSAM file system. The UNIX input file can be a binary or a text file. The difference between a binary file and a text file resides in the fact that each record in the text file is terminated by the new line character ("\n" in C-language).

In case of binary files, each record is written at the end of the preceding one.

Syntax

```
xvsamImport [-p][-d][-l<debug level>][-r][-s][-i<char|0xHH>]
<catalog> <cluster> [<data file> <B|T>]
```

Option

```
-r existing record keys are overwritten
-s input is read from STDIN
-p print input parameters
-i<char|0xHH> inits record to value specified (in hex or
character )
-s input is read from STDIN

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog</td>
<td>is the name of the catalog the cluster belongs to</td>
</tr>
<tr>
<td>cluster</td>
<td>is the name of the cluster to be deleted</td>
</tr>
<tr>
<td>flat file</td>
<td>is the complete absolute path of directories and the name of the UNIX file which is to be imported</td>
</tr>
<tr>
<td>B</td>
<td>T</td>
</tr>
</tbody>
</table>

A protocol file can be created during the import function, by specifying `-d<name>`. If no name is specified, the protocol file will be created as `xvsamImport.log`. This logging file is useful when the number of read records does not correspond to the number of written records; in the logging file each input and output record is saved, and in case of error an explanation message is written. The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the `-l<level>` optional parameter.

At the end of the import operation, `xvsamImport` reports how many records have been read from the UNIX file and how many records have been written in the XVSAM data set. The two numbers should correspond; if not, check the reason in the `xvsamImport.log` protocol file.

Example

```
# xvsamImport TESTCAT SNICTHLP /some/where/CUAHLP B
xvsamImport: Records read[159] written[159]
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

5.3.1.2.5 xvsamExport

`xvsamExport` performs an opposite operation to `xvsamImport`; it saves the content of an XVSAM cluster in a UNIX flat file. Data can be stored in two different formats: binary form - each record is written at the end of the preceding one - or ASCII form, where a new line character is added at the end of each record.

Syntax

```
xvsamExport [-p][-d][-l<debug level>][-k<skip>][-c<count>][-b<from_key>][-B<from_key_hex>][-e<to_key>][-E<to_key_hex>]
```

© 2007 H.T.W.C. Srl
<catalog> <cluster> [<data file> <B|T>]

Option

-<k<skip> records number to skip
-c<count> records number to export
-b<from_key> define start key to export records from (char)
-B<from_key_hex> define start key to export records from (hex)
-e<to_key> define end key to export records to(char)
-E<to_key_hex> define end key to export records to(hex)
-p prints input parameters

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog</td>
<td>is the name of the catalog the cluster belongs to</td>
</tr>
<tr>
<td>cluster</td>
<td>is the name of the cluster to be deleted</td>
</tr>
<tr>
<td>flat file</td>
<td>is the complete absolute path of directories and the name of the UNIX file that must be exported</td>
</tr>
</tbody>
</table>
| B|T | output file type:  
  B the file to be exported is a binary file  
  T the file to be exported is an ASCII text file |

A protocol file can be created during the file export, by specifying -<d<name>. If no name is specified, the protocol file will be created as xvsamExport.log.

The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the -<l<level> optional parameter.

At the end of the export operation, xvsamExport reports how many records have been read from the XVSAM data set and how many of them have been written on the UNIX file.

Example

```
# xvsamExport TESTCAT SNICTHLP /some/where/else/snicthlp B
xvsamExport: Records read[159] written[159]
```

Return codes

<table>
<thead>
<tr>
<th>Return code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>
5.3.1.2.6 xvsamLog

xvsamLog is a utility which provides you the possibility to analyze what C_ISAM has stored in the RECOVERY.LOG file, with the aim to identify all operations executed on a single record, a single file etc.

Syntax

```
xvsamLog [-r] [-d] [-p pid] [-f file [-x] [-k regexp]] < RECOVERY.LOG > file
```

Launched without any parameter, xvsamLog analyzes the input-output sequence of operations stored by C-ISAM on the RECOVERY.LOG file, and it produces a printout in a readable format.

At the moment, recognized I/O operations are:

- Begin Transaction beginning of a LUW - logical unit of work -
- Rollback LUW rollback
- Open data set OPEN
- Close data set CLOSE
- Insert insert of a new record
- Delete deletion of an existing record
- Update rewriting of an existing record
- Build data set up
- Create Index creating index record
- Erase data set removing

For each operation contained in the RECOVERY.LOG file, the following information is printed: time of operation, internal process identifier of the transaction - that, for UNIX servers, coincides with the pid of the process - and file sequence number - isfd - that is assigned at the moment of the OPEN of each file and remains unique within the transaction.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-r</code></td>
<td>to print the content of each record for INSERT, DELETE and UPDATE operations</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>to switch report to the distributive mode. In this way, all operations belonging to one single transaction are grouped and they will appear all together in the output file. Transactions will be in any case presented ordered by starting time</td>
</tr>
<tr>
<td><code>-p</code> &lt;pid&gt;</td>
<td>to print only transactions with the specified identification number, that means a specific UNIX process. The UNIX process is characterized by its pid: Process ID</td>
</tr>
<tr>
<td><code>-f</code> &lt;file&gt;</td>
<td>is used to print out only operations that were performed on the specified data set. &lt;file&gt; must contain the complete path to the file, without any extensions as i.e., .dat or .idx. The distributive mode is automatically activated when this parameter is invoked and in the print out only transactions which have opened the specified file are listed</td>
</tr>
<tr>
<td><code>-k</code> &lt;regexp&gt;</td>
<td>may be specified only together with the <code>-f</code> parameter. It extracts only operations on the specified file and the records containing the supplier string of characters &lt;regexp&gt;. Binary strings for comparison may be also entered; they must be prefixed by the <code>-x</code> parameter with indicates that the subsequent string of data is in hexadecimal format</td>
</tr>
</tbody>
</table>

Examples

```
# xvsamLog -r < RECOVERY.LOG > recovery.txt
```

All input-output operations and contents of each record are listed out on the RECOVERY.TXT file.
This command extracts only the input-output operations performed by transactions which have opened the specified XVSAM file PRONTOT, belonging to the USRCAT3 catalog, and it lists them out on RECOVERY.TXT.

```
# xvsamLog -f /host/valb/data/USRCAT3/PRONTOT < RECOVERY.LOG > recovery.txt
```

This command extracts only input-output operations performed on all the records of a specified file - the XVSAM file PRONTOT from catalog USRCAT3 - which contain in their 4 first bytes the hexadecimal value X'0219040C' or X'0219040F'.

```
# xvsamLog -r -f /host/valb/data/USRCAT3/PRONTOT -x -k "^0219040[CF]" < RECOVERY.LOG > recovery.txt
```

### 5.3.1.2.7 xvsamInfo

`xvsamInfo` returns information about the C-ISAM version with which the XVSAM components are linked-in or, if you enter a cluster name as parameter, it gives information about the required data set. It calls the ‘isindexinfo’ C-ISAM routine and it does not use any catalog information.

`xvsamInfo` may have one of the two following formats:

1. No parameter is supplied: in this case information about the version number of the C-ISAM component are returned.

```
xvsamInfo
```

2. The second format of xvsamInfo contains a cluster name. The command should be given in the directory where the data set is physically stored or the complete path of directories must be supplied.

```
xvsamInfo <cluster>
```

Where :

- `cluster` is name of the XVSAM cluster to for which information are required

#### Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

### 5.3.1.2.8 xvsamRepro

The xvsamRepro command copies VSAM and non-VSAM data sets.

#### Syntax

```
xvsamRepro <defines input file> <defines output file>
```

can use contract parameter

```
xvsamRepro --ic=<input catalog> --iu=<input cluster> --of=<output file> --v
```

or extended
xvsamRepro --input_catalog=<input catalog> --input_cluster=<input cluster> --output_file=<output file> --verbose

Defines input and ouput files is mandatory. Input and output options depends on files type (VSAM or flat). All parameter are list below. If parameter has a corresponding value in second column means parameter have to be followed by equals sign and the required value

--ic=<input catalog>

else use it alone

--debug

Parameters

### VSAM input file

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ic</td>
<td>&lt;input catalog&gt;</td>
<td>is the name of the cluster to be deleted</td>
</tr>
<tr>
<td>--iu</td>
<td>&lt;input cluster&gt;</td>
<td>input xvsam cluster</td>
</tr>
<tr>
<td>--fk</td>
<td>&lt;char...&gt;</td>
<td>from key in input xvsam file</td>
</tr>
<tr>
<td>--tk</td>
<td>&lt;char...&gt;</td>
<td>to key in input xvsam file</td>
</tr>
<tr>
<td>--fkh</td>
<td>&lt;hh...&gt;</td>
<td>from key hexadecimal in input xvsam file</td>
</tr>
<tr>
<td>--tkh</td>
<td>&lt;hh...&gt;</td>
<td>to key hexadecimal in input xvsam file</td>
</tr>
</tbody>
</table>

### FLAT input file

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--if</td>
<td>&lt;input path&gt;</td>
<td>input flat file path</td>
</tr>
<tr>
<td>--si</td>
<td>stdin</td>
<td>read stdin in input</td>
</tr>
<tr>
<td>--im</td>
<td>&lt;B</td>
<td>T&gt;</td>
</tr>
<tr>
<td>--ifo</td>
<td>&lt;V</td>
<td>F&gt;</td>
</tr>
<tr>
<td>--il</td>
<td>&lt;number&gt;</td>
<td>input flat file length</td>
</tr>
</tbody>
</table>

### VSAM output file

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--oc</td>
<td>&lt;output catalog&gt;</td>
<td>output xvsam catalog</td>
</tr>
<tr>
<td>--ou</td>
<td>&lt;output cluster&gt;</td>
<td>output xvsam cluster</td>
</tr>
</tbody>
</table>

### FLAT output file

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--of</td>
<td>&lt;output path&gt;</td>
<td>output flat file path</td>
</tr>
<tr>
<td>--r</td>
<td>replace</td>
<td>replace records in output xvsam file</td>
</tr>
<tr>
<td>--so</td>
<td>stdout</td>
<td>write stdout in output</td>
</tr>
</tbody>
</table>
--om|output_mode <B|T> output flat file mode
--ofo|output_format <V|F> output flat file format
--ol|output_len <number> output flat file length
--a|append open output flat file in append mode

Options

--s|skip <n> skip n records in input
--c|count <n> stop after n records in output
--i|init [char|0xhh] write one initialized record
--h|help print help
--v|verbose print accepted parameters and values
--dl|debug_level <0-15> set debug level
--d|debug produce debug file: xvsamRepro.log

Examples

Copy vsam file to flat file

```
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile
Warning: undefined output format: fixed forced
```

Copy vsam file to flat file formatted with newline at end of each record:

```
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile --om=T
Warning: undefined output format: fixed forced
```

Copy flat formatted file in vsam file

```
xvsamRepro --if=$HOME/tmp/reprofile --im=T --oc=TEST --ou=VSAMTEST
Warning: undefined input format: fixed forced
```

Copy VSAM file to flat, skipping first n record.

```
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile --om=T --s=3
Warning: undefined output format: fixed forced
```
Copy first n record of VSAM file to flat file

```bash
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile --cm=t --c=3
```

Warning: undefined output format: fixed forced

```bash
```

Debug & Logs

You can force command --v to enable verbose mode, to obtain print of all input parameters.

```bash
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile --v
```

You can produce log file (xvsamRepro.log) forcing --d options. You can also defines debugging level from 0 to 15. Default is 15, if you want to suppress area dump you can force debug level to 5.

```bash
xvsamRepro --ic=TEST --iu=VSAMTEST --of=$HOME/tmp/reprofile --d
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

### 5.3.1.2.9 xvsamEdit

*xvsamEdit* allows you to edit an existing data set to check for example if it has been created with the right characteristics.

It can be also used to check the content of the data set after it has been loaded in the XVSAM environment by means of xvsamImport.

The presentation of the records takes place according to the same layout as for xvsamAdm, already described.

**Syntax**

```
xvsamEdit [ <catalog> <cluster>]    
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog</td>
<td>is the name of the catalog the cluster belongs to</td>
</tr>
<tr>
<td>cluster</td>
<td>is the name of the cluster to be deleted</td>
</tr>
</tbody>
</table>

A protocol file can be created during editing, through specifying -d<name>. If no name is specified, the protocol file will be created as xvsamEdit.log.

The level of detail with which XVSAM should report the sequence of operations on the protocol file, may be specified by means of the -l<level> optional parameter.

If catalog and cluster names are omitted, *xvsamEdit* operates in the same way as *xvsamAdm*, and the catalogs choice list is output on your screen.

**Return codes**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>
5.3.1.2.10  xvsamRecover

xvsamRecover analyzes an XCICS recovery log file, and re-apply all changes occurred on files tracked in the log.

After an abnormal XCICS termination (i.e. hardware failure), xvsamRecover can be used to reproduce changes tracked during an XCICS sessions, on XVSAM data restored from an existing backup.

Syntax

```
xvsamRecover
```

Parameters

- `<filename>` specifies the recovery log file to analyze

Example

```
# xvsamRecover -f /home/data/RECOVERY.LOG
```

Return codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>

5.3.1.2.11  xvsamWarmStart

xvsamWarmStart analyzes an XCICS recovery log file, showing eventual un-committed transactions (AKA in-flight transaction) and, optionally, rolls them back.

After an abnormal XCICS termination (i.e. hardware failure), xvsamWarmStart should be used to remove all uncommitted transactions from the XVSAM files, to ensure data integrity.

xvsamWarmStart is automatically invoked by the XCICS control procedure, `xcicsctl`, when invoked with command `warmstart`.

Syntax

```
xvsamWarmStart [-f<filename>] [-x] [-r]
```

Parameters

- `<filename>` specifies the recovery log file to analyze
- `-r` performs un-committed transaction rollback. Without this option it only shows "in-flight" transactions.
- `-x` forces the recovery, even if errors should occur

Example

```
# xvsamWarmStart -f /home/data/RECOVERY.LOG -r
```
### Return codes

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Function correctly executed</td>
</tr>
<tr>
<td>12</td>
<td>error(s) trapped</td>
</tr>
</tbody>
</table>
5.3.1.3 xvsamRts

xvsamRts is an embedded runtime system to execute batch COBOL programs, providing direct support to the XVSAM file system, through the standard COBOL I/O instructions (i.e. READ, WRITE, etc).

Depending on the XFRAME setup, xvsamRts provides runtime support for Microfocus COBOL or for ACUCOBOL.

**Configuration**

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XVSAM</td>
<td>path to the XVSAM repository root</td>
</tr>
<tr>
<td>XVSAMRTSDBG</td>
<td>if set to “YES” xvsamRts will produce a trace file named xvsamRts.log</td>
</tr>
<tr>
<td>XVSAMRTS0D</td>
<td>if set to “YES”, xvsamRts will add a carriage return char (\r - 0x0d) at the end of the line of printing files</td>
</tr>
<tr>
<td>XVSAM_VAR_FMT</td>
<td>specifies the format of variable record length files. The admitted values are: IBM (default) XFRAME</td>
</tr>
<tr>
<td>XVSAMRTS_IGNORE_SE5_SYMBOLS</td>
<td>if set to “YES”, xvsamRts ignores the special files used by Microfocus Server Express 5.0 runtime to handle the COBOL internal SORT statements, suppressing the no-sense warnings displays caused by their usage.</td>
</tr>
<tr>
<td>XVSAMRTS_CHECK_FLUSH</td>
<td>if set to “YES” xvsamRts will perform addition error checking on file flushing</td>
</tr>
<tr>
<td>XVSAMRTSFCC</td>
<td>it specifies the global FCC behaviour. Please look at the FCC section below.</td>
</tr>
<tr>
<td>COBLPFORM</td>
<td>it defines up to 12 printing channels in the form: setenv COBLPFORM &quot;c1:c2:......:c12&quot; see the your Microfocus ServerExpress documentation for further details</td>
</tr>
<tr>
<td>XVSAMRTSSIGHUP</td>
<td>if set to “YES”, xvsamRts will additionally handle SIGHUP signal.</td>
</tr>
<tr>
<td>ACUDBGSRV</td>
<td>address of the ACUCOBOL debugger. The format is depending in the mode set in ACUDBGMODE</td>
</tr>
<tr>
<td>ACUDBGMODE</td>
<td>mode for ACUCOBOL debugger. Admitted value: display (default)</td>
</tr>
</tbody>
</table>

**Files specification**

In order to provide to the programs running on the XVSAM RTS, all the information about files and their characteristics from the scripts, environments have to used.
For each file specified in the program to be run, an environment named `dlbl_FILENAME` must be provided. This environment will contain all the information required (path, catalog, disp, etc).

For further information about `dlbl_` environments, please refer to the DD & DLBL Section, in the JCL translation book.

**Feed Control Character (FCC) management**

It is possible to set FCC at the global level or for each file separately.

Global setting is performed by defining the environment variable named `XVSAMRTSFCC`.

Individual settings for specified files may be done in different ways:

- setting environments named `XVSAMRTSFCC_FILE`, i.e.

  ```
  setenv XVSAMRTSFCC_FILE01 "ASA"
  ```

- by specifying the "fcc=" in the `dlbl_` environment, i.e.

  ```
  setenv dlbl_FILE01 "FILE.01,cat=SYSLST,disp=(NEW,KEEP),fcc=ASA"
  ```

- forcing `XVSAMRTSFCC_FILE` directly in the batch COBOL program, i.e.

  ```
  DISPLAY 'XVSAMRTSFCC_FILE01' UPON ENVIRONMENT-NAMEDisplay 'ASA' UPON ENVIRONMENT-VALUE
  ```

Admitted FCC values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA</td>
<td>add an ASA control character before each line</td>
</tr>
<tr>
<td>ASN</td>
<td>as ASA, but add newline (0a) after each line</td>
</tr>
<tr>
<td>AMA</td>
<td>as ASA, but put the control character in the first line position instead of adding it before</td>
</tr>
<tr>
<td>AMN</td>
<td>as AMA, but add newline (0a) after each line</td>
</tr>
<tr>
<td>AFP</td>
<td>print HTAFPHT as the first line, add newline (0a) after each line</td>
</tr>
<tr>
<td>FORCE</td>
<td>force the ASA management, wait the control character in the first byte of each line</td>
</tr>
<tr>
<td>PRINT</td>
<td>force the file be LINE SEQUENTIAL: remove trailing spaces and put newline (0a) after each line</td>
</tr>
</tbody>
</table>

**xvsamRts error codes**

xvsamRts can return following error codes and messages.

<table>
<thead>
<tr>
<th>Exit code</th>
<th>File status</th>
<th>Message displayed</th>
<th>Exit condition</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>219</td>
<td>91</td>
<td>-</td>
<td>file status not declared</td>
<td>I-O operation on a closed file.</td>
</tr>
<tr>
<td>Code</td>
<td>Signal</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>-</td>
<td>xvsamRts: error closing file [&lt;ind&gt;] always closing a file that has not been opened.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>-</td>
<td>xvsamRts: error keynum is too large [&lt;keynum&gt;] always An attempt was made to operate with an undefined secondary key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>-</td>
<td>xvsamRts: error the variables vsam_&lt;file&gt; and dd_&lt;file&gt; are not defined always Control variables vsam_ or dd_ are not defined. Check script for correctness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>-</td>
<td>xvsamRts: error &lt;file&gt; must be assigned to EXTERNAL name always Error in COBOL SELECT statement. File must be assigned via EXTERNAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>128+signal n.</td>
<td>-</td>
<td>always Signal received</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>22</td>
<td>xvsamRts: error I-O [&lt;file&gt;] [22] file status not declared Duplicate key</td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>23</td>
<td>xvsamRts: error I-O [&lt;file&gt;] [23] file status not declared Record not found</td>
<td></td>
<td></td>
</tr>
<tr>
<td>155</td>
<td>4D</td>
<td>xvsamRts: error I-O [&lt;file&gt;] [10027] file status not declared Record locked</td>
<td></td>
<td></td>
</tr>
<tr>
<td>223</td>
<td>95</td>
<td>xvsamRts: error I-O [&lt;file&gt;] [95] file status not declared An attempt was made to open for output an non-empty non-reusable file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>10</td>
<td>- always End of File (EOF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>90</td>
<td>xvsamRts: error I-O [&lt;file&gt;] [90] [&lt;lisorerr&gt;&lt;errmsg&gt;] file status not declared Generic I-O error. Please refer to ERRMSG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>218</td>
<td>90</td>
<td>xvsamRts: Unsupported operation [&lt;opcode&gt;] always An attempt was made to perform an unsupported operation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5.3.2 xvsamAdm

XVSAM Access Method Administration Service, is a utility program which provides the possibility to create, update or delete XVSAM objects in one or more XVSAM catalogs. The product is comparable with the IBM mainframe tool IDCAMS, running in DOS/VS, VSE or OS/390 environment. xvsamAdm can only operate in an interactive mode.

To start xvsamAdm just enter the command xvsamAdm at the shell prompt.

The following screen will appear on your screen:

This is the main menu of xvsamAdm. It provides you the possibility to choose among the following 12 functions:

1. Create Cluster to create the catalog definition for a cluster file you intend to handle under XVSAM
2. Create Altindex to create a catalog definition of an alternate index for an already existing base cluster
3. Delete file to delete a base cluster or an alternate index previously selected. The file content and its catalog definition are removed.
4. Erase file to clean a base cluster or an alternate index previously selected. Only the file content is cleaned; its definition in the catalog remains unchanged.
5. Edit file to edit the content of a base cluster or an alternate index
6. Import to import binary or text files into the XVSAM system
7. Export to export XVSAM files as binary or text files
8. Select file to select a base cluster or an alternate index on which you intend to perform afterwards XVSAM operations like Delete, Erase, Import, Export etc.
9. Check file to check characteristics of the base cluster
R. Connect remote computer to connect a remote xvsamServer, running on the same or another server
E. Edit catalog entry to edit the properties of the selected cluster or alternate index
0. Exit to leave xvsamAdm and to return to UNIX level.

By entering on the prompt:

Choice ==> _

the number or letter corresponding to the desired function, you may directly invoke it. The message line MESSAGE: is used for displaying eventual error messages or missing information.
By pressing the <ESC> (ESCAPE) key, you can leave xvsamAdm and return to UNIX command level.

The on-line HELP function supplies you information about the use of xvsamAdm can be invoked from all panels, simply by pressing the F1 key.

Different on-line help items can be chosen:

1. **Help on Help** general information about the using of HELP
2. **Keyboard** function keys meaning
3. **Xvsam6 – Release notes** release notice and technical information about XVSAM
4. **xvsamEdit** help about the xvsamEdit feature
5. **xvsamAdm** general information about xvsamAdm

To choose an item, you should place the cursor >, by means of the Up and Down Arrow keys, in front of the wished item and then press the <ENTER> key.
5.3.2.1 Creating a cluster

To create a new base cluster, enter 1 in the selection field of the main menu.

Validation of your choice, which takes place when the <ENTER> key is pressed, produces the display of the EDITING CLUSTER INFORMATION window:

On this screen, the following information should be entered:

- **Catalog**: It is the name of the Catalog to which the cluster belongs to. The physical location of the file may be given by means of the PATH parameter. When the referenced Catalog does not yet exist, it is automatically created. Its name may be up to 44 characters long, and it must be conform to the UNIX naming conventions.

- **Cluster**: Here the logical name of the base cluster should be entered. This name too may be up to 44 characters long.

- **Path**: It is the name of the directory where the physical file, containing user data, is located. The assigned pathname is the concatenation of the name of the Catalog to the XVSAM path. For easiness, it is suggested to use the same name for the Catalog and its path.

- **Type (K|E|R)**
  Here the data organization is specified. The three possible data organizations are:
  
  K for KSDS
  E for ESDS
  R for RRDS

- **Format (F|V)**
  Here the record format can be entered:
  
  F for fixed length records
  V for variable length records

- **Recmin**: In the Recmin field you should supply the minimum record length, in bytes, allowed for the data set.

- **Recmax**: In the Recmax field you should supply the maximum record length, in bytes, allowed for the data set. In the case of a file with fixed length records, Recmin and Recmax have the same value.

- **Nparts**: This field is used for defining the number of parts the primary key of a cluster file is composed by. Such a
feature has no correspondence in the VSAM environment, where the fields composing a primary key must be contiguous. It is only used for reasons of compatibility with the Siemens product LEASY, where a key field may also be composed by not contiguous areas. For VSAM files Nparts is always set to 1, while for ESDS and RRDS files, this field should be kept to zero.

- Keys if the file organization is KSDS, the characteristics of the key should be specified using the following scheme:
  
  [offset keylength offset keylength ...]
  
  offset is the displacement of the key from the beginning of the record and relative to zero;
  
  keylength is the length of the key in bytes.
  
  For VSAM files only one pair of values is allowed, while for LEASY files, more than one pair can be specified.

- Lname it serves to specify the logical name. The logical name can be maximal 8 bytes long. It is reserved for future use.

- Reuse one byte indicator to specify if the file is reusable or not. Possible values are: Y or N. If set to Y (yes) OPEN OUTPUT on non-empty files is allowed. If N is set, error 712 is returned if an OPEN OUTPUT operation is tried on a non-empty file.

- Lockmode Lock management:
  
  M – manual lock forced
  A – auto lock forced
  E – exclusive lock forced
  D – lock mode specified by caller

While defining a base cluster with ESDS or RRDS organization, the Key fields are automatically skipped.

The following function keys are available:

- PF1 to invoke the on-line Help
- Cursor keys to move among fields
- Escape key to interrupt the definition of the cluster and to return to the main menu
- Enter key to make the base cluster definition active
5.3.2.2 Creation of an alternate index

If you want to create an alternate index for an existing base cluster, you should enter 2 in the main menu. The EDITING ALTDINDEX INFORMATION map, as shown in the next figure, will then be displayed.

On this screen, the following information should be entered:

- **Catalog** Here should be entered the name of the Catalog containing the cluster to which the alternate index belongs to.

- **Cluster** Here the name of the alternate index you intend to create must be specified. It can be maximal 44 characters long.

- **Primary** This field is used to specify the name of the base cluster the alternate index is related to.

- **Type [D|U]** The two possible values are:
  - D duplicate keys are allowed; it means that it is possible to have different primary keys associated with the same alternate index
  - U unique; only one primary key may be associated with the alternate index

- **Nparts** This field is used for defining the number of parts the secondary key is composed of. As in case of primary key when defining a cluster, such a feature has no correspondence in the VSAM environment, where the fields which compose also the secondary key must be contiguous, and it is used only for reasons of compatibility with the Siemens product LEASY. In this case a key, being it primary or secondary may also be composed by several not consecutive fields. For VSAM files Nparts is always set to 1, while for ESDS and RRDS files, this field should be kept to zero.

- **Keys** Information about the secondary key should be specified here using the following scheme:
  
  [offset keylength offset keylength ...]

  offset is the displacement of the key from the beginning of the record and relative to zero

  keylength is the length of the key in bytes.

  For VSAM files only one pair of values is allowed, while for LEASY files several pairs can be specified.

The following function keys are available:

- **PF1** to invoke the on-line help
Cursor keys to move among fields

Escape key to abort the definition of the cluster and to return to the main menu

Enter key to activate the definition
5.3.2.3 Selection of a file

Before performing any kind of operation on an already existing base cluster, you are required to select its name. Entering 2 in the CHOICE field of main menu causes the following map to be displayed:

xvsamAdm searches the information contained in the Master Catalog File (Catalog file in the directory indicated by the XVSAM environment variable), and it builds up a list of all the available XVSAM user catalogs.

The first ten catalog names, sorted in ascending alphabetic sequence, are listed in the dialog box presented in the middle of the screen. By moving the cursor by means of the Up and Down Arrow keys, you can choose the Catalog in which the file you intend to process is located.

To speed up the searching of the Catalog, you can also use the F8 and F7 function keys, which scroll the list respectively one page forwards or backwards.

The <ESC> key terminates the file selection and control is returned to the main menu.

As soon as you have selected the Catalog you are interested in, you are required to specify the name of the cluster file you want to processed by means of the SELECTING FILE panel:
On the dialog box, in the middle of the screen, all the base cluster / alternate indices entries contained in the selected catalog are listed. You can scroll them up and down until the name of the desired cluster has been reached. All clusters, together with their related alternate index files are listed, sorted alphabetically by name.

Alternate index (AIX) files are marked by the --> sign, and they are listed immediately after the name of the corresponding base cluster.

The following information is furthermore shown:

- **Catalog** name of the selected Catalog
- **File Name** name of the cluster
- **XVSAM** organization:
  - K for KSDS
  - E for ESDS
  - R for RRDS
  - For alternate index files only:
    - D indicates that duplicated keys are allowed
    - U indicates that duplicate keys are not allowed (UNIQUE KEY)
- **Record format**
  - F for fixed length records
  - V for variable length records
- **Min. record size** minimum record size
- **Max. record size** maximum record size
- **Key position** relative displacement of the record key, starting from the beginning of the record (position 0)
- **Key length** length of the record key
- **UNIX path of the file** UNIX path under $XVSAM
The following function keys are available:

**Cursor keys** to scroll among fields

**Escape key** to go back

**Enter key** to choose one entry

As soon as the Catalog and the cluster you intend to process have been selected, these two information are shown in the xvsamAdm main menu panel as shown below:

All operations listed in the following require a previous selection of the file to be processed.
Each data set contained in the XVSAM system can be easily processed by using the EDIT FILE feature. In this case you should enter function - 5 - Edit File in the selection field. The map shown in the next figure will then be displayed.

The Editing file panel is split into three parts: on the first five upper lines of the map, the relevant information about the file are listed:

- Catalog name
- Cluster name
- Key position and key length
- Record number
- XVSAM organization: KSDS - ESDS - RRDS
- Record size
- Minimum and maximum record length
- Number of records contained in the data set
- Current record length

In the middle part of the map, the content of each record both in character and in hexadecimal format is shown, while the third one, located in the lower part of the screen contains the message line and information about the usage of available function keys.

Two viewing modes are possible: ASCII or EBCDIC. The actual value is displayed, but it can be toggled from one to the other through pushing the F9 (or Ctrl+A) key. This will only affect the display of the records as UNIX works in ASCII.

The cursor is positioned at the beginning of the record. Through using the right and left arrow keys, you can move it within the record and overwrite one or more characters on it. Both the content of a record or its key can be modified.

The third part of the screen provides information about the usage of the keys:

- **Cursor keys** to move among fields
- **F1** (or Ctrl+K) to invoke the on-line help
- F3 (or Ctrl+E) to go back to the main menu
- F5 (or Ctrl+B) to search the next occurrence of a pattern
- F7 (or Ctrl+P) to read previous record
- F8 (or Ctrl+N) to read next record
- F9 (or Ctrl+A) to switch view mode ASCII/EBCDIC
- F10 (or Ctrl+X) to select a new file

CTRL + F to activate the advanced function menu

The advanced processing features, made available by CTRL+F, can be used to execute the following input-output operations on the current record:

- s to start a browsing (sequential read) operation from the current record
- r to read a record directly
- w to write the current record
- d to delete the current record
- o to read the current record with lock
- u to rewrite the current record
- n to release lock on the current record
- f to read the first record in the file
- l to read the last record in the file
- e to refresh the screen (all the field contents are initialized to binary zeroes)
- x to define a search pattern
- c to change the record length (for a variable length files)
- k to change the record key (RBA for ESDS data sets, RRN for RRDS data sets)

Please consider that some features as i.e. a browsing operation or the direct reading of a record - with or without lock - need a key field to be supplied. Positioning on the first or last record of a file does not require any key.

If you want to insert a new record in a data set by means of the WRITE (w) feature, you have to supply one of the following Record Identification Fields:

- Primary Record Key: in case of KSDS data sets the new record will be inserted, if not already existing, in a position determined by its primary key.
- Relative Record Number: in case of RRDS files, the new record will be inserted, if not already existing in the position corresponding to the supplied RRN.
- Relative Byte Address: in case of ESDS data sets the new record will always be added at the end of the data set.

If otherwise, you intend to rewrite an already existing record, you should first read it with lock and after having modified the content of one or more fields - but not the primary key - you can proceed with updating it by means of the u (rewrite) feature.

Record deletion can be done directly, that's to say that it is not necessary to read previously the record with lock before entering the d (delete) command code.
5.3.2.5 Import of data

This function can be used to load into an XVSAM cluster a UNIX flat file, already in ASCII code. This file can be both the result of converting mainframe archives, or the output of a UNIX native procedure, or whatever else is recognized to be a flat file.

If you have entered 6 in the CHOICE field, the panel shown in the next figure will then be displayed.

The TYPE field serves to specify which kind of data are to be imported. The following two types are possible:

- **B** binary data; all records are one after the other.
- **T** text file; each record is terminated by a new line character ('
').

As soon as the <ENTER> key is pressed, data will be imported. Once the process is terminated, the xvsamAdm main menu panel is displayed and the message line updated with the following information:

```
Message: Import read[159] written[159] -----------------------------
```

The message line informs you about the number of records found in the source file and the number of records written on the XVSAM data set. If they correspond, then the data set has been correctly imported, otherwise a problem may have occurred during the import phase and it will be necessary to correct it before repeating the operation.

The most common problems which can arise are:

```
Message: Error Importing file: No such file or directory -----------
```

Information supplied in the PATH field is invalid; check the existence of the directory and of the source data set on it.

The number of read records is higher than the number of written records. Such a situation can arise in the following cases:

- the data set already contains records with the same key: before repeating the IMPORT function, it is necessary to erase them by means of function 4 - ERASE FILE
- probably a wrong value has been supplied in one the KEYS fields during the cluster’s definition; check them with the original values

The following function keys are available:

- **PF1** to invoke the on-line help
- **Cursor keys** to scroll among fields
- **Escape key** to go back
- **Enter key** to choose one entry
5.3.2.6 Exporting a file

In addition to the import of a data set, it is also possible to execute the reverse operation, that is to say to export an XVSAM file as a binary or text file. This feature can be invoked by selecting - 7 - EXPORT on the main menu. For this operation too, it is necessary to have previously selected the name of the cluster you intend to process.

You are required to specify the full path name into the PATH field. It corresponds to the directory where the file to be exported is to be created.

The TYPE field serves to specify how data should be exported. The following two types are possible:

- **B** binary data; all records are written one after the other.
- **T** text file; each record is terminated by a new line character ('\n').

As soon as the <ENTER> key is pressed, data will be exported. Once the process is finished, the xvsamAdm Main Menu panel is displayed and the message line updated with the following information:

> Message: Export read[159] written[159] ---------------------------

This message line informs now about the number of records contained in the XVSAM file to be exported and about the number of records which have been effectively written on the output file.

The following function keys are available:

- **F1** to invoke the on-line help
- **Cursor keys** to move among fields
- **Escape key** to go back
- **Enter key** to choose an entry
5.3.2.7 Erasing a file

The ERASE FILE function allows you to clean and zeroise an XVSAM cluster without removing its definition from its relative catalog. This can be done through entering \texttt{2 - ERASE FILE} in the selection field of main menu, after having ensured that a valid cluster name has been selected. Enter "Y" to confirm erasing, "N" to abort the operation.

**WARNING:** The erase function cleans the content of a file, which cannot be therefore recovered; it has the same effect as the UNIX rm command.

The following function keys are available:

- PF1 to invoke the on-line help
- Cursor keys to move among fields
- Escape key to go back
- Enter key to choose an entry

### Delete a vsam file

As an alternative to the ERASE FILE function, which only removes the cluster from the file system, the DELETE FILE function deletes the cluster and removes its definition from the relative catalog. When an alternate index is deleted by using this function, only the catalog entry is deleted. The deletion of a KSDS cluster leads to the deletion of all its alternate keys too.

A catalog is automatically deleted as soon as the last cluster contained in it has been deleted. The directory with the same name of the catalog, if existing, is also automatically removed under $XVSAM$.

The procedure for deleting a file is exactly the same as for erasing it, except that \texttt{3 - DELETE FILE} must be chosen in the main menu.

**WARNING:** The delete function physically removes the XVSAM file from the UNIX file system, and it cannot be therefore recovered

### Check file

This function, accessed by typing \texttt{9 -} in the main menu, provides you the possibility to control if the file characteristics contained in the XVSAM catalog are compatible with information available to C-ISAM for a specific data set. In case of congruity the confirmation message
Message: File O.K.

is sent back, while in case of not compatible information, the following message will appear on the message line:

Message: Error checking file: data_set_name
5.4  Programmer's reference
5.4.1  XVSAM logical structure

The catalog

XVSAM uses the ISAM file Catalog as VSAM master catalog, to keep all information about the structure of data files. Catalog is 400 bytes fixed record length file containing two different kind of records: cluster records and alternate index records with the same structure, but identified by the first character in the record layout, as shown in the following figure:

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Cluster record</th>
<th>Alternate index record</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATALOG</td>
<td>44</td>
<td>name of the logical catalog which contains the file</td>
<td>same</td>
</tr>
<tr>
<td>CLUSTER</td>
<td>44</td>
<td>cluster name</td>
<td>same</td>
</tr>
<tr>
<td>PRIMARY</td>
<td>44</td>
<td>blank</td>
<td>primary cluster name</td>
</tr>
<tr>
<td>PATH</td>
<td>80</td>
<td>UNIX directory (see “path convention” below)</td>
<td>blank</td>
</tr>
<tr>
<td>RECFORMAT</td>
<td>1</td>
<td>‘F’ for fixed-length file or ‘V’ otherwise</td>
<td>blank</td>
</tr>
<tr>
<td>RECMIN</td>
<td>5</td>
<td>minimum length of record in C format “%05d”</td>
<td>number of key in C format “%05d”</td>
</tr>
<tr>
<td>RECMAX</td>
<td>5</td>
<td>maximum length of record in C format “%05d”</td>
<td>blank</td>
</tr>
<tr>
<td>NPARTS</td>
<td>5</td>
<td>key parts number in C format “%05d”</td>
<td>same</td>
</tr>
<tr>
<td>8 KEYOFF</td>
<td>5</td>
<td>key offset in C format “%05d”</td>
<td>same</td>
</tr>
<tr>
<td>times</td>
<td>KEYLEN</td>
<td>5 key length in C format “%05d”</td>
<td>same</td>
</tr>
<tr>
<td>RESERVED</td>
<td>83</td>
<td>Reserved for future using</td>
<td>same</td>
</tr>
</tbody>
</table>

The Catalog file, which is stored in the directory pointed by the XVSAM environment variable, must not be cancelled or moved somewhere else.
5.4.2 XVSAM programming interface

XVSAM can be used to access data by on-line as well as by batch programs, written in COBOL or C language.

**COBOL interface**

For on-line programs, XCICS takes care to interface XVSAM whenever a file operation is required by user programs. For COBOL batch programs, the XVSAM access interface has been embedded into a proprietary COBOL External File Handler (EFH), which provides the connection with the XVSAM interface. Almost no change is required to let your batch COBOL programs run access XVSAM files.

The only required modification involves the COBOL SELECT clause that should be modified in the ASSIGN clause as follows:

In IBM COBOL VS

```
ASSIGN TO SYSnnn-DA-disktype-S-name
```

it becomes in MicroFocus COBOL:

```
ASSIGN TO EXTERNAL name
```

The linkage between the name supplied in the SELECT clause and the XVSAM data set takes place at the moment of the program’s run by means of the setenv (C-shell) command or export (Korn-shell) command, which correspond to the VSE DLBL or MVS DD statements:

I.e.

**IBM VSE**

```
DLBL NAME  filename,VSAM,CAT=catalog
```

**IBM z/OS**

```
NAME DD DSN=filename,CAT=catalog
```

Becomes in UNIX csh:

```
setenv vsam_NAME "filename,cat=catalog"
```

or

```
setenv dlb1_NAME "filename,cat=catalog,type=vsam"
```

**C language interface**

XVSAM can be accessed from any program written in C language by using the xvsamapi function: source code must include xvsamapi.h and use xvsamapi() entry point.

The object code has to be linked, dynamically or statically, with the XVSAM libraries, libxvsam6.a or libxvsam6.so.

**xvsamapi function**

The `xvsamapi` function has the following prototype:
int xvsamapi ( char ident [11],
    int cop,
    int isfd,
    char catalog [45],
    char cluster [45],
    int mode,
    int lock,
    int nkey,
    int *length,
    long *rrn,
    unsigned char *iobuf);

where:

ident is a null-terminated character identification string. Actually user programs should use the string “VSUS210” for right identification.

cop is an operation code. The set of all supported operation codes is defined in xvsamapi.h include file (see appendix A).

isfd is a Xvsam6 file handler. The xvsamapi returns a handler for all open file operations and wait for it for all other operations on files. NULL can be passed for non open file operations because Xvsam6 uses isfd parameter for the file identification.

catalog is the name of the XVSAM Catalog, where are stored all user data files.

cluster is the name of the file on which an open file operation should be performed.

mode is the C-ISAM mode for open, read and start. Xvsam6 adds two new modes for read: XVSAMLESS for “read less than” and XVSAMLTEQ for “read less or equal”.

lock is the C-ISAM lock parameter for open or read.

nkey is the key number for read.

length is a pointer on integer containing the record length.

rrn is a pointer on long containing the relative record number (for RRDS files)

iobuf is a pointer on the data record.

The “xvsamapi” function returns the isfd file handler on open file operation and the undefined value on all other operations.

The “xvsamapi” function sets the value of the iserrno variable to 0, to one of the C-ISAM errors or to one of Xvsam6 errors which are defined in xvsamapi.h include file.

The “xvsamapi” function prepares the string to pass to the next xvsamIO layer. This string has a machine-independent format so it can be send and received via socket by any other server. It gives the possibility to split application and data on two different physical devices, for increasing data security.

Linking libraries

It is possible to link any of the normal (libxvsam6.a) or shared (libxvsam6.so) versions of Xvsam6 library to user programs. The Xvsam6 utilities use the shared version, so the LD_LIBRARY_PATH environment variable should point also to the directory containing the libxvsam6.so library.
5.4.3 Debugging

A user program can dynamically manage the XVSAM log, by defining the XvsamFdbg external variable. XVSAM writes the call information and the error messages in xvsamFdbg stream, if the value of xvsamFdbg differs from NULL. It uses the variable xvsamLdbg to establish the level of debugging information: 0 - no log printing, 15 - maximum level of debugging information.

For example the program:

```c
external FILE *xvsamFdbg;
external int xvsamLdbg;
main () {
    xvsamFdbg = fopen ("xvsam.log", "w");
    xvsamLdbg=10;
    xvsamapi (...);
}
```

writes the Xvsam6 log information in the file xvsam.log with a debugging level of 10 - average information logging level for diagnose -.

**User debugging function**

*xvsam6* calls the function xvsamSdbg (if it's value isn't equal to NULL) any time it needs to print some debug information. The user can substitute the internal debug function with its own one. For example:

```c
#include <stdarg.h>
extern void (*xvsamSdbg)(int, char *, int, char *, char *, ...);
static void my_debug(int, char *, int, ...)
    va_start(marker, format);
    vfprintf(stdout, format, marker);
    va_end(marker);
    fflush(stdout);

main() {        /* change the debug print function for my one */
    xvsamFdbg = my_debug;
    xvsamapi (...);
}
static void my_debug(int lev,
    char *file,
    int line,
    char *id,
    char *format,
    ...)
{
    va list marker;
    if (level > 5) return;
    if (file != NULL) printf("\%s: \%d", file, line);
    if (id != NULL) printf("\%s: ", id);
    va_start(marker, format);
    vfprintf(stdout, format, marker);
    va_end(marker);
    fflush(stdout);

```
5.4.4 V2R - Database storage system

As described in the concepts section, XVSAM may store files on relational database tables. Each file is stored in a single table, and keys and indexed are managed by means of database features.

File access to database stored file is always performed using the usual XVSAM interfaces: EXEC CICS READ in XCICS/TS and normal I/O statement in COBOL programs (i.e. READ, WRITE).

The database access is achieved by means of file specific I/O programs (AKA progio). At generation time, XVSAM creates and compiles a COBOL program which is called for each I/O operation on the file and takes care to correctly manage it with a corresponding SQL command.

Table structure

The structure of the table containing the XVSAM file should match the file structure, containing one column for each single field contained in the file.

In addition to the file fields, XVSAM adds some system columns to allow direct access to table rows from legacy programs and easier access from new programs.

The name of the host table will be automatically composed by the logical name of the file prefixed by the tag XVSAM_: i.e.

| XVSAM_CLUST01 |

Configuration

XVSAM database interface is configured by means of the following environment variables:

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XVSAM_CONNECT</td>
<td>none</td>
<td>database connection string</td>
</tr>
<tr>
<td>XVSAM_PROGIO_PATH</td>
<td>COBPATH</td>
<td>path to PROGIO executables</td>
</tr>
</tbody>
</table>

Oracle users must be aware of the setting of NLS_LANG variable: it affects the data conversion between oracle client and server. XVSAM may store binary data into database fields, so no conversion should occur: because of that NLS_LANG variable must be set with the same language of the oracle server.

I.e.

```
export XVSAM_CONNECT=scott/tiger@ORCL
export XVSAM_PROGIO_PATH=$HOME/progio:$HOME/objs/int
export NLS_LANG=AMERICAN_AMERICA.WE8ISO8859P1
```

Environment generation

Of course the database must be properly setup before XVSAM may use it to store its files. Database user must be correctly setup with all the privileges required to create and access tables and with space enough to store data.

Additionally, some XVSAM specific functions must be defined for the oracle database user: the source for these functions is the file $XFRAMEHOME/etc/v2rfun_ora.sql.
Files generation

Whenever an XVSAM file must be stored on database, it must be properly created, supplying its record structure representation. XVSAM uses the XFRAME XML-based data structure descriptor.

This XML file may be obtained using “cpy2xml” utility (see XCONV Data Conversion Toolkit), and it contains information concerning the structure of the file.

The structure cannot contain REDEFINES clauses.

I.e.

```xml
<xml version="1.0" encoding="UTF-8"/>
<datadef source="FUNZIONI.cpy" language="COBOL" built="Wed May 24 12:43:01 MEST 2006">
    <datastruct>
        <dataitem name="FUNZIONI" level="1">
            <dataitem name="CAMPO-CHIAVE" level="2">
                <dataitem name="CAMPO1" level="3">
                    <dataitem name="CAMPO1" level="4" picture="S9(5)" length="5" usage="display" signed="true"/>
                    <dataitem name="CAMPO2" level="4" picture="S9(4)" length="2" usage="comp" signed="true"/>
                    <dataitem name="CAMPO3" level="4" picture="S9(4)" length="3" usage="comp-3" signed="true"/>
                </dataitem>
            </dataitem>
        </dataitem>
    </datastruct>
</datadef>
```

Once obtained the correct XML for record description, the command “xvsam” must be used to create the file, the database objects and the progio:

```
xvsam --create
    --catalog <CATALOG NAME>
    --type=ksds
    --record-length=<RECORD LENGTH>
    --key=<KEY INFO>
    --storage-engine=database
    --logical=<LOGICAL NAME>
    --xml-fd=<XML file>
    <CLUSTER NAME>
```

Alternate indexes are generated with a similar command.
6 Batch Environment

JCLs coming from mainframe system must be converted into UNIX/Linux shell scripts to run.

To minimize the changes in user programs and batch scheduling flow, the logic of the script must reflect the logic of the source JCL. Very often this logic is strictly dependant or based on some basic features of mainframe OSes (i.e. DISPOSITIONs) which are not available on UNIX/Linux systems: XFRAME supplies many of this missing features through its core components (i.e. XVSAM and XSORT) and some auxiliary utilities.

Furthermore, the XFRAME JCL converter (xmvscov) translates the source JCL codes into corresponding C or Korn shell statements, using, wherever is possible, system native functions or constructs to emulate the mainframe behaviour.

VSE JCLs can only be migrated using the C shell, while when migrating z/OS JCLs both C shell and Korn Shell may be used, depending on the presence of GOTO statements or not.

The conversion logic as well as the auxiliary utilities are described in the following pages.

Running JCL scripts

Obviously, converted JCLs are standard C or Korn shell scripts, therefore may be started as a standard script.

I.e.

```
  csh MYJCL01.csh
```
6.1 Scripts logic

JCL conversion is a task which may require an high grade of customization. Each user in fact has his specific way for coding JCLs and uses special tools or components. There is unfortunately no complete and unique rule and method to convert JCLs.

As a consequence XFRAME provides a programmable JCL converter, \texttt{xmvsconv}, which is composed by a core conversion system and a programmable configuration interface.

However this document is not intended to be a guide to \texttt{xmvsconv}, but only a comparative reference to JCL migration, explaining how mainframe component and logic can be converted.

\textbf{The scripting language}

JCL is the scripting language available on IBM mainframe and it is different between VSE and MVS. It can be replaced using standard UNIX shells.

VSE JCLs can be migrated using the C shell, while when migrating MVS JCLs both C shell and Korn Shell may be used, depending on the presence of GOTO statements or not.

All the samples in the following pages are written in C Shell.
6.1.1 EXEC

The EXEC statement causes, on a mainframe environment, a program, a JCL or a PROC to be started, depending on the supplier parameters.

In the XFRAME environment, the start method changes depending on what you intend to start.

Starting programs

The command xrun is used to start COBOL programs, according to the following syntax:

```bash
xrun <pgm-name>
```

Passing data cards

xrun enables programs to receive parameters cards, and to do it, it uses the standard input descriptor. Therefore if your parameter cards are enclosed in the script, an in-script redirection using the "<<" redirector is to be utilized.

For example:

```bash
#!/bin/csh
xrun PROG1 <<end_data_cards
PROG1 19991201 ***** *****
PROG1 19991202 00099 00145
PROG1 ***
end_data_cards
```

If the parameters cards are contained in a separate file, then the "<" redirector can be used:

```bash
xrun PROG1 < /some/where/cards.txt
```
6.1.2 DD & DLBL

To emulate this function, use the ASSIGN EXTERNAL directive when you submit your program to the COBOL compiler.

Before execution, you can associate the data file names, used in your program with the cluster name - in case of VSAM files - or physical names - in case of standard UNIX files -, via the setenv or export shell command.

Both DD (for MVS JCL) and DLBL (VSE JCLs) are substituted with their equivalent environment variables linked to the EXTERNAL names of COBOL programs.

Association between cluster name and physical file name whenever necessary will be provided by the xrun interface.

As described above, the physical name of a file is specified using an environment variable, therefore all corresponding environment variables for the referenced files must be exported / set before the program is executed.

xrun can handle both flat and XVSAM files, this has for consequence that two different kinds of environment variables have to be used.

The environment value referring to a file link will contain information about:

- cluster name
- file type
- catalog name (optional)
- disposition (optional)
- gdg (optional)
- fcc (optional)

Syntax

```bash
setenv dlbl_FILENAME "<cluster>[,cat=<catalog>,type=<flat|vsam>[, disp=<disposition>][, gdg=<version>]
```

For Example the MVS source

```
//EXEC   PGM=MERGE01
//INPUT1 DD DSN=FILE1.KSDS,DISP=(OLD,KEEP,KEEP)
//INPUT2 DD DSN=FILE2.ESDS(+2),DISP=(OLD,DELETE)
//OUTX   DD DSN=SORT.IN,DISP=(NEW,KEEP)
```

becomes

```
setenv dlbl_INPUT1 "FILE1.KSDS,disp=(OLD,KEEP,KEEP),type=vsam"
setenv dlbl_INPUT2 "FILE2.ESDS,disp=(OLD,DELETE), type=flat, gdg=+2"
setenv dlbl_OUTX "SORT.IN,disp=(NEW,KEEP),type=flat"
xrun MERGE01
```

<cluster> is the physical name of the file. <catalog> is the catalog name for VSAM files or the relative path to the location of the file and refers to the current $XVSAM environment variable.

Normally this path should be the name of the XVSAM catalog sheltering the file.

Dispositions are expressed in a similar way as in the VSE/MVS environment:

```
(,<start-state>,<exit-state>,<fault-state>)
```

where <start-state> indicates the initial state of the required file for further execution of the program:

- OLD file exists
- SHR file exists
- NEW  file doesn’t exist
- MOD  file exists and open for append

Both <exit-state> and <fault-state> indicate the states of file after execution, in case of normal and abnormal termination:

- KEEP leaves the file on the system
- DELETE removes it.

The default value for disposition is (OLD,KEEP).

**Example:**

COBOL code:

```cobol
INPUT-OUTPUT SECTION
FILE-CONTROL
    SELECT FILE1 ASSIGN TO EXTERNAL FILEIN
    ORGANIZATION IS INDEXED
    RECORD KEY IS R-KEY
    ACCESS IS DYNAMIC.
    SELECT FILE2 ASSIGN TO EXTERNAL FILEOUT
    ORGANIZATION IS SEQUENTIAL.
```

JCL code

```bash
#!/bin/csh
setenv dlbl_FILEOUT "FILESAM,cat=CATTEMP,disp=(NEW,KEEP)"
setenv dlbl_FILEIN  "CLUANAG,cat=CTUSR1"
xrun DOWNLD
```
6.1.3 Data Cards

In the original environment parameters cards may be passed to program from SYSIN or SYSREADER. They are normally specified in JCLs.

In the XFRAME environment, parameters cards are passed to a program via stdin (standard input) and they can be embedded directly in the script using standard features of the UNIX system.

In this case you have to inform xrun to read parameters cards from standard input, using all the possibility offered by the UNIX system to associate stdin.

Possible syntaxes are:

```
xrun <program-name> <<file_name
```

in case the parameters cards must be read from an external file or, if they follow the EXEC card as in the following VSE example:

```
// EXEC DIB900, SIZE=512K
DIB900 - ACIACIO DIB900 - FEBRUARY 99
DIB900 - UFF. 101 DIB900 - UDATE *
/*
```

they can be passed in XFRAME environment as:

```
xrun DIB900 <<end_data_cards
DIB900 - ACIACIO DIB900 - FEBRUARY 99
DIB900 - UFF. COI DIB900 - UDATE *
end_data_cards
```
6.1.4 PROC parameters substitution

JCLs coming from MVS (OS/390) can make use of dynamic PROC input, DD and parameters substitution. Cataloged
PROCs, as well as JCLs, are converted in C/Korn shell scripts. To take replicate the dynamic substitution feature the
XDSH utilities must be employed:

- xdinif
- xdadd
- xdsh

The first one must be called at the beginning of the caller script, to initialize the XDSH system. xdadd is the equivalent of a
DD statement: it defines what and where must be actualized in the called script (PROC). xdsh takes care to prepare and
execute the called script.

A sample

```
#!/bin/csh
#
source $HOME/etc/xjobinit.csh
xdinif
STEP01:
  EXEC PROC01, P1=X, P2=D
  FILEIN DD DSN=FILE01, CRTL=CHAR, DISP=SHR
  FILEOUT DD *
  01012003 31122003 ALL
  01012003 31122003 OPEN
*/
  SYSIN DD *
  LOAD SKED
  PRINT
*/
set STEPFNAME=STEP01
set STEP01="step $STEPNAME running..."
xdadd $STEP01 "setenv FILEIN=' $FILE01' CRTL=CHAR,disp=SHR"
xdadd --file $STEP01 FILEOUT=<<end_of_file
  01012003 31122003 ALL
  01012003 31122003 OPEN
end_of_file
xdadd --stdin $STEP01 <<end_of_file
  LOAD SKED
  PRINT
end_of_file
xdsh PROC01
set RC=$status
```

Requirements

XDSH tools require the following environments:

- **PID** containing the PID number of the main JCL.
- **PROCLIB** containing the path for PROCs.

Both them must be set and exported at the beginning of the main JCL.
6.1.5 SORT

Calls to the SORT or ICEMAN utilities are handled by means of XSORT. It is an high performance sort program, which accepts the same data cards of IBM DF/SORT. Please refer to XSORT documentation for further information.

XSORT is started using xun sort, as in the following example:

```
//STEP0060 EXEC GGOZ65Y,SPAZIO='(6225,(1000,100),,,,ROUND)' SORT0000
//SORTIN DD DSN=PCMDOS.EL22.SOWTR340.GC0,DISP=SHR
//SORTOUT DD DSN=PCMDOS.EL22.SGW15EF1.MA0,DISP=SHR
//UNIT( D, 85 ), SPACE=(CYL, (10,10), ELSE),
//DCB=(SYS1.DCB, RECFM=FB, LRECL=300, BLWSIZE=0)
//SYSIN DD *
//SORT FIELDD=(15,7,A22,7,A1,4,A4,FORMAT=HI)
//INCLUDE COND=(I,4,C4,Eq.C,nero',OR(I,4,C4,Eq.C,'NERO'))

//******************************************************

xunstep "STEP0060"
sset concatenated="$XMSAV/$TMP1/$JOBNAME $PID.STEP0060.SORTINL cat, tmp"
cat 'xqpd --full --get 0 $XMSAV/$TMP1/PCMDOS.EL22.SOWTR340.GC' >
$concatenated
cat 'xqpd --full --get 0 $XMSAV/$TMP1/PCMDOS.EL22.SGW15EF1.MA' >>
$concatenated
setenv dlib_SORTINL "$JOBNAME $PID.STEP0060.SORTINL cat, tmp=$TMP1/"
setenv dlib_SORTOUT "PCMDOS.EL22.SGW15EF1.MA, cat=$TMP1/Disp=(NB),
KEEP, type=flat, gid=1"
xrun xsort lrecl=300 <<end_data_cards
SRT FIELD=(15,7,A22,7,A1,4,A4,FORMAT=HI)
INCLUDE COND=(I,4,C4,Eq.C,'nero',OR(I,4,C4,Eq.C,'NERO'))
end_data_cards
xrun
set STEP0060 RC=$RC
set STEP0060 xsort RC=$RC
if ( $FLUSH_JOB ) then
  xdisplay "$STEPNAME flushing with RC=$RC"
xflush
endif
unsetenv dlib_SORTINL
unsetenv dlib_SORTOUT
```
6.1.6 FLAT & VSAM

A flat file is a file containing records that have no physical record organization: on Unix/Linux records are only a logical entity. XVSAM bypasses this limit and allows to have files with a record-based organization, which may be accessed either in sequential or keyed mode.

During the conversion process it is important to define if a file will be handled as flat or XVSAM, and, due to the different kind of organization, direct operations on file (i.e. creation, deleting, copy, etc) must be converted differently.

In the following paragraphs, some example of conversion will be shown.

**Define**

**vsam**

The source

```plaintext
// EXEC IDCAMS,SIZE=AUTO
DEFINE CL(NAME(VA.PMEDIQ.COS)                  //
   SPEED SHR 4) RECSZ(039 039) RECY(0100 0100) -
   KEYS(19 00) VOLL(VAVOLS) INDEXED) -
   CATALOG (VACATS) -
   DATA NAME(VA.PMEDIQ.COS.DATA) CISZ(2048)) -
   INDEX NAME(VA.PMEDIQ.COS.INDX) CISZ(512) )/
```

becomes

```plaintext
xvsam --create --catalog VACATS --record-length 39 --key 0,19 --type ksds VA PMEDI Q.COS
```

**flat**

The source

```plaintext
// EXEC IDCAMS,SIZE=AUTO
DEFINE CLUSTER(             //
   NAME VA.GDBT10 ) -
   SPEED -
   RECORDS (50 50) -
   NONINDEXED -
   RECORDSIZE (200 200) -
   CISZ (4096) -
   REUSE -
   VOLUMES (WAVOL1) -
 ) -
   DATA ( -
   NAME (VA.GDBT10.DATA) -
 )
```

becomes

```plaintext
touch $XVSAM/WKCAT1/VA.GDBT10
```
Delete

vsam
The source

```bash
// EXEC IDCAMS, SIZE=AUTO
DELETE (VA.PMEDIQ.COS) PURGE CAT(VACAT5)
/*
```

becomes

```bash
xvsam --delete --catalog VACAT5 VA.PMEDIQ.COS
```

flat
The source

```bash
// EXEC IDCAMS, SIZE=AUTO
DELETE (VA.GDBT10) PURGE NOERASE CLUSTER CAT(WKCAT1)
/*
```

becomes

```bash
rm -f $XVSAM/$IJSYSUC/VA.GDBT10
```

Alter

vsam
The source

```bash
* JOB ALTER
// EXEC IDCAMS, SIZE=AUTO
ALTER VA.ORD.S NEWNAME(VA.ORD.IM.S) CAT(VACAT5)
END
/*
```

becomes

```bash
xvsam rename VACAT5 VA.ORD.S VACAT5 VA.ORD.IM.S
```

flat
The source

```bash
* JOB ALTER
// EXEC IDCAMS, SIZE=AUTO
ALTER VA.GDBT10 NEWNAME(VA.GDBT10.OLD) CAT(WKCAT1)
END
```
The VSE source

```c
/*

becomes

mv $XVSAM/WKCAT1/VA.GDBT10 $XVSAM/WKCAT1/VA.GDBT10.OLD

Repro

vsam

The VSE source

```

```c
// DLBL INPU,'IPOST.ASSFAM.ANAG',,VSAM,CAT=IPUCT06
// DLBL OUTP,'IPOST.ASSFAM.ANAG.CALC',,VSAM,CAT=IPUCT02
// EXEC IDCAMS
// REPRO INFILE(INPU) OUTFILE(OUTP)
/*

becomes

xvsam --repro --input-catalog IPUCT06 --input-cluster IPOST.ASSFAM.ANAG 
  --output-catalog IPUCT02 --output-cluster IPOST.ASSFAM.ANAG

flat

The VSE source

```

```c
// DLBL IJSYSUC,'WKCAT1',,VSAM
// DLBL IN,'PPROD8J.TXT',,VSAM,CAT=WKCAT1
// DLBL OUT,'PPROD8J.TXT',,VSAM,CAT=VACAT5
// EXEC IDCAMS,SIZE=AUTO
// REPRO INFILE(IN) OUTFILE(OUT)
/*

becomes

cp $XVSAM/WKCAT1/PPROD8J.TXT $XVSAM/VACAT5/PPROD8J.TXT
```
6.1.7 GDG (Generation Data Group)

GDG stands for Generation Data Group and it is a method used on the mainframe to allow a group of related files to be created that can be referenced individually or as a group.

The name of the first GDG created file version is composed by the the base name plus "00" (number of version) added to the end.

When a new version of GDG file is created, all the previous versions are renamed increasing by 1 their number of version. The default number of maintained versions is 8. A different range can be indicated in xgdg.conf file, setting the group sizee (see Configuration).

The lower version is the last version.

Example

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov 8</td>
<td>16:24</td>
<td>/appl/akilo00/ki00/data/FLATCAT/TEST.E122.SCMET006.GC.02</td>
</tr>
<tr>
<td>Nov 8</td>
<td>16:24</td>
<td>/appl/akilo00/ki00/data/FLATCAT/TEST.E122.SCMET006.GC.01</td>
</tr>
<tr>
<td>Nov 8</td>
<td>17:23</td>
<td>/appl/akilo00/ki00/data/FLATCAT/TEST.E122.SCMET006.GC.00</td>
</tr>
</tbody>
</table>

Files within a GDG may be referenced using the physical name but they are more usually referenced using relative generation numbers.

Generation 0 (zero) (i.e. TEST.E122.SCMET006.GC(0)) represents the current generation within the GDG at the time that the current job began execution while -1 (minus one) represents the immediately previous version (TEST.E122.SCMET006.GC(-1)). Even earlier versions can be referenced using -2, -3 etc. up to the maximum number of generations held within the GDG. If a new generation is added to the GDG during execution of the job then it will be the +1 generation.

Referencing the files within a GDG by their relative names means that the actual physical filename does not need to be known.

GDG cataloging features are normally used inside a "GDG processing unit", that is script of commands. When the processing unit completes, the new version created during the execution unit are cataloged or removed depending on the results of the processing unit.

Usage

GDG files are used in the scripts running under the XFRAME environment.

Normally they are referenced in the file link declaration (i.e. dbbl_XXX) before the program invocation. In this case, the xgdg subsystem of the XFRAME runtime directly interprets the GDG information from the the "gdg" parameter contained in dbbl_[external_name] variable

Example

```bash
setenv dbbl_FILEIN "PKMS0, E122,SCMET040,GC,cat=FLATCAT,disp=OLD,type=flat,gdg=0"
```

Sometimes it may be necessary to get GDG information to let non XFRAME commands to use it. You may invoke the xgdg utility and get the information required. For example if our target is to know the correct name of GDG's version you can use

```bash
xgdg --full --get 0 SXVSAM/FLAT/FILE01
```

for more information refer to "xgdg".

For each XGDG processing unit (normally a script/JCL), the XJCLLOG environment value must be set to a unique file name, which name uniquely identifies the execution run.

I.e
When script is launched, the XFRAME batch runtime writes into the file addressed by $XJCLLOG all the script operational information (files used, GDG versions, return code of the steps, disposition, etc).

At the end of script, the command "xgdg --catalogue" must be invoked: it analyzes the information contained in $XJCLLOG, and performs the cataloging of necessary files.

Configuration

In order to have the XFRAME xgdg subsystem working, a configuration file must be provided. This file must be named xgdg.conf and must reside in the $HOME/etc directory.

This file contains both general GDG parameters as well as specific file settings.

I.e.

```bash
# xgdg.conf - configuration for xgdg
# set keep($RC < 8)
set verbose
define name=E104/PCKBOS.RA04.SCKB000.JA group=4
define name=E104/PCKBOS.RAXX.TSATSTTT.JA group=12
```

Sample run

```bash
--> step STEP0020 running
XRUN: redirecting stdout on /appl/akb000/kb0/data/FLATCAT/PCKBOS.E104.SCKB042.GS
XRUN: DD ISTBP010 SHR not exists
/appl/akb000/kb0/data/FLATCAT/GTBMLR.C000.ISTBP010.GA
XRUN: DD IF86630 SHR 670230
/appl/akb000/kb0/data/FLATCAT/PCKBOS.E104.SCKB042.GA
XRUN: DD WKRPT08 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT08
XRUN: DD WKRPT04 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT04
XRUN: DD ISTBP020 SHR not exists
/appl/akb000/kb0/data/FLATCAT/GTBMLR.C000.ISTBP020.GA
XRUN: DD WKRPT12 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT12
XRUN: DD ISTBP040 SHR not exists
/appl/akb000/kb0/data/FLATCAT/GTBMLR.C000.ISTBP040.GA
XRUN: DD WKRPT11 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT11
XRUN: DD ACKBX040 ,CATLG,DELETE 0
/appl/akb000/kb0/data/FLATCAT/PCKBOS.E104.SCKB040.GA tmp 1
XRUN: DD WKRPT15 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT15
XRUN: DD ISTBP060 SHR not exists
/appl/akb000/kb0/data/FLATCAT/GTBMLR.C000.ISTBP060.GA
XRUN: DD WKRPT17 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT17
XRUN: DD WKRPT16 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT16
XRUN: DD WKRPT06 NEW KEEP not exists
/appl/akb000/kb0/data/TMPGR/gkbe0418.sh.3666826.WKRPT06
XRUN: DD ISTBP070 SHR not exists
/appl/akb000/kb0/data/FLATCAT/GTBMLR.C000.ISTBP070.GA
```
XRUN: DD OCZZZ166 NEW KEEP not exists
 appl/akb000/kb0/data/SYSLST/gkbe0418.sh 366626, STB00X00.txt
XRUN: DD WKRPT020 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT020
XRUN: DD ACKBD048 , CATLG, DELETE 18120
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT0
XRUN: DD ACKBD048 , CATLG, DELETE 56625
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT0
XRUN: DD WKRPT03 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT03
XRUN: DD WKRPT03 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT03
XRUN: DD WKRPT10 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT10
XRUN: DD WKRPT09 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT09
XRUN: DD WKRPT05 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT05
XRUN: DD WKRPT19 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT19
XRUN: DD WKRPT11 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT11
XRUN: DD WKRPT18 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT18
XRUN: DD IZTBPO50 SHR not exists
 appl/akb000/kb0/data/FLATCAT/GTBMRA C000, IZTBPO50, GA
XRUN: DD WKRPT14 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT14
XRUN: DD ACKBD045 , CATLG, DELETE 2720
 appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8045, GA
XRUN: DD WKRPT02 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT02
XRUN: DD WKRPT07 NEW KEEP not exists
 appl/akb000/kb0/data/TMFCGR/gkbe0418.sh 366626, WKRPT07
XRUN: DD ACKBD047 , CATLG, DELETE 16600
 appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8047, GA
XRUN: DD ACKBD041 SHR 45602816
 appl/akb000/kb0/data/VSMCGR/GC3923, EL04, VSMCGR, GA
XRUN: DD OCZZZ2036 , CATLG, DELETE 221312
 appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8043, GA
XRUN: DD ACKBD049 , CATLG, DELETE 23840
 appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8049, GA
XRUN: DD IZTBPO30 SHR not exists
 appl/akb000/kb0/data/FLATCAT/GTBMRA C000, IZTBPO30, GA
XRUN: DD ICZZZ2166 OLD, DELETE 146
 appl/akb000/kb0/data/TMFCGR/IICZZZ166, 366626
XRUN: ACGBD042 erasing
 appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8040, GA trn 1
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8046, GA
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8048, GA
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8045, GA
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8047, GA
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8043, GA
XRUN: ACGBD042 erasing /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8049, GA
XRUN: STARTING ACGBD042
XRUN: DATA CARDS
  31 060731
XRUN: END OF DATA CARDS
XRUN: TERMINATED WITH CODE 0
XRUN: /appl/akb000/kb0/data/FLATCAT/GTBMRA C000, IZTBPO10, GA (not exists)
XRUN: /appl/akb000/kb0/data/FLATCAT/GC3923, EL04, SCB8042, GA (67023)
XRUN: /appl/akb000/kb0/data/TMFCGR/gkbe0418, sh 366626, WKRPT08 (not exists)
XRUN: /appl/akb000/kb0/data/TMFCGR/gkbe0418, sh 366626, WKRPT04 (not exists)
XRUN: /appl/akb000/kb0/data/FLATCAT/GTBMRA C000, IZTBPO20, GA (not exists)
At the end of script:

```bash
xgdg: checking GDG for /appl/akb000/kb0/data/FLATCAT/PCKBOS.E104.SCKBX040.GA
xgdg: ... not found in configuration: assuming default
xgdg: deleting PCKBOS.E104.SCKBX040.GA.08
xgdg: cataloging completed
```
6.1.8 JCL Example

Source lines are remarked by `#'. This character in Cshell marks line as comment.

The `xbmXX' keywords are XBM command.

If you submit jcl with XBM tool this keyword are necessary.

See XBM documentation for further information

### Heading

```csh
#!/bin/csh
#
# script converted as JOB by xmvsconv $Revision: 1.27 $ (VSE mode)
# on Mon Jan 27 11:41:48 MET 2003
#
source $XBMHOME/lib/xbmlib.csh
xbminit

# * $S$ JOB JNN=PFGECO3,CLASS=A,DISP=D,USER=IPFI,PRI=3
# * $S$ LST CLASS=R,DEST=(*,LAPLACA)
source $HOME/etc/xjobinit.csh
```

Is possible to include into jcl other cshell with `source' keyword. It to add other environment or global action without reconvert all jcls anytime

There are more Cshell inserted into jcl. You decide which insert into your jcl.

- `xjobinit.csh' at the beginning of job
- `xstepinit.csh' at the beginning of each step
- ...
- `xjobexit.csh' at the end of job

```csh
if ($#argv >= 1) then
goto $1
endif
```

Redirect job start to specific step. You pass stepname as argument in command line when job is submited. Those parameters are enclose in $#argv system array. If its value is grater than 1 then goto statement is processed.

### Step

```csh
# // DLBL IJSYSUC,'IPUCT04.UCAT',,VSAM
# // DLBL IPGECO3,'IPOST.FITTI.CONDOS',,VSAM
# // EXEC IDCAMS,SIZE=AUTO
# DELETE IPOST.FITTI.CONDOS
#   CLUSTER
#   FILE(IPGECO3)
#   PURGE
#   CAY(IPUCT04.UCAT)
# DEFINE CLUSTER
#   ( NAME(IPPOST.FITTI.CONDOS) FILE(IPGECO3) )
#   CYL(31) FOR(9999) KEYS(14) BCSZ(150 150)
#   INDEXED CISZ(2560) OWN(IPPOST) SHR(2)
# DATA ( NAME(IPPOST.FITTI.CONDOS.DATA) )
```
Each step is identified by a label STEPnnnn where nnnn is an incremental number.

setenv IJSYSUC "IPUCT04"

The job environment declaration of $IJSYSUC assigned value "IPUCT04"

xvasamDelete IPUCT04 IPOST.FITTI.CONDOS
xvasamCreate "<end_of_cards
C IPUCT04 IPOST.FITTI.CONDOS IPUCT04 K F 150 150 1 0 14 IPOST.FITTI.CONDOS N
end_of_cards
xmr
if ( $RC != $RCZERO ) then
  xdisplay "$XBM_STEP return code $RC"
  xdisplay "continuing execution."
endif

At the end of each step there is a return code test. $RC is last return code cathered. If it is different from zero, a message is displayed to standard output.

# /
# JOB PFGECO03 IPFI
# OPTION PARTDUMP
# DLBL SYS020,'IPOST.FITTI.CONDOM',,VSAM CAT=IPUCT04
# DLBL SYS021,'IPOST.FITTI.UNITA.IMM0P',,VSAM CAT=IPUCT04
# DLBL SYS022,'PATH.IPPOST.FITTI.PALAZ',,VSAM CAT=IPUCT04
# DLBL SYS023,'IPOST.FITTI.LOCAT',,VSAM CAT=IPUCT04
# DLBL SYS024,'IPPOST.FITTI.CONDOS',,VSAM CAT=IPUCT04
# DLBL SYS025,'IPPOST.FITTI.ANACON',,VSAM CAT=IPUCT04
# DLBL SYS026,'IPPOST.FITTI.MENSIL',,VSAM CAT=IPUCT04
# DLBL SYS027,'IPPOST.FITTI.CONDOM',,VSAM CAT=IPUCT04
# EXEC PFGECO03,SIZE= (AUTO,300K)
# P-06-1998-01
# /
# STEP0020:
# xhtmlp STEP0020
# xdisplay "step $XBM_STEP... running"
setenv dlbl_SYS025 "IPPOST.FITTI.ANACON cat=IPUCT04, type=vsam"
setenv dlbl_SYS026 "IPPOST.FITTI.MENSIL cat=IPUCT04, type=vsam"
setenv dlbl_SYS027 "IPPOST.FITTI.CONDOM cat=IPUCT04, type=vsam"
setenv dlbl_SYS028 "IPPOST.FITTI.UNITS.IMM0P cat=IPUCT04, type=vsam"
setenv dlbl_SYS022 "PATH.IPPOST.FITTI.PALAZ, cat=IPUCT04, type=vsam"
setenv dlbl_SYS023 "IPPOST.FITTI.LOCAT, cat=IPUCT04, type=vsam"
setenv dlbl_SYS024 "IPPOST.FITTI.CONDOS, cat=IPUCT04, type=vsam"
xml: IPUCT04/IPPOST.FITTI.ANACON
xml: IPUCT04/IPPOST.FITTI.MENSIL
xml: IPUCT04/IPPOST.FITTI.CONDOM
xml: IPUCT04/IPPOST.FITTI.UNITA.IMM0P
xml: IPUCT05/IPPOST.FITTI.PALAZ
xml: IPUCT05/IPPOST.FITTI.LOCAT
xml: IPUCT04/IPPOST.FITTI.CONDOS
xml:lock
EOJ: Label is the closing label. The job finish with $RC return code.

xjobinit.csh

This cshell is included into all job. It contains some global environment necessary to run job. Also is possible to add other environment if necessary. At the end, xdisplays print time job started.
environment

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COBPATH</td>
<td>program path (*.gnt or *.int version). You can insert more than one path, divided by &quot;.&quot;.</td>
</tr>
<tr>
<td>PROCPATH</td>
<td>proc path. When you include a proc with source statement you could insert PROCPATH variable instead full path</td>
</tr>
<tr>
<td>TMP</td>
<td>path temp dir</td>
</tr>
<tr>
<td>PARMLIB</td>
<td>path parmlib dir</td>
</tr>
<tr>
<td>TMPCAT</td>
<td>Default catalog.</td>
</tr>
<tr>
<td>VSAMCAT</td>
<td></td>
</tr>
<tr>
<td>FLATCAT</td>
<td></td>
</tr>
<tr>
<td>XSPOOLOPT</td>
<td>Contained Xspool parameters. See Xspool documentation for further information</td>
</tr>
</tbody>
</table>

xjobexit.csh

At the end, xdispaly print time job finished.

```bash
#!/usr/bin/tcsh
#
# xjobexit.csh
#
# xdisplay "Job $XBM_JOBNAME terminated at `date`"
```
6.2 Auxiliary utilities

6.2.1 xdinit

`xdinit` initializes the substitution data for a Job. It should be issued only once at job beginning.

**Syntax**

```
xdinit
```

**Environment variables**

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process identifier (PID) of the main caller script.</td>
</tr>
<tr>
<td>PROCLIB</td>
<td>Path for called scripts (PATH)</td>
</tr>
<tr>
<td>XJCLLOG</td>
<td>environment value MUST be set to a unique file identifier for any JOB using xlist (see xrun).</td>
</tr>
</tbody>
</table>
6.2.2 xdadd

xdadd adds parameters to the substitution phase for the following PROC called.

Both standard input and declared DD (dbl_ environments) may be substituted. If options --lineseq or --blocked are specified a temporary file is created and given to the called PROC.

The --blocked option provides file in blocked mode (without newline characters), for the specified record size. The -- lineseq simply provides a standard line sequential file.

Syntax

    xdadd [options] <stepname> [<ddname>] [<args>]

Options

    -p, --pid <PID> specifies PID number
    -i, --stdin prepares data for step standard input substitution (SYSIN/stdin)
    -l, --lineseq prepares temporary line sequential file for DD/DLBL substitution
    -b, --blocked <rs> prepares temporary sequential file, blocked <RS>, for DD/DLBL substitution

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepname</td>
<td>is the name of the step for which substitution is to be operated</td>
</tr>
<tr>
<td>ddname</td>
<td>is the name of the DD entry in the called PROC for which substitution is to be operated</td>
</tr>
<tr>
<td>args</td>
<td>are the arguments for substitution</td>
</tr>
</tbody>
</table>

Environment variables

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process identifier (PID) of the main caller script.</td>
</tr>
<tr>
<td>PROCLIB</td>
<td>Path for called scripts (PATH)</td>
</tr>
<tr>
<td>XJCLLOG</td>
<td>environment value MUST be set to a unique file identifier for any JOB using xlist (see xrun).</td>
</tr>
</tbody>
</table>
6.2.3 xdsh

**xdsh** is the PROC runner. It invokes the specified PROC, substitutes parameters and runs it.

**Syntax**

```
xdsh [options] <procname>
```

**Options**

- `-p`, `--pid<pid>` forces xdadd to use another pid

**Parameters**

```
procname
```

is the PROC to invoke

**Environment variables**

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PID</td>
<td>Process identifier (PID) of the main caller script.</td>
</tr>
<tr>
<td>PROCLIB</td>
<td>Path for called scripts (PATH)</td>
</tr>
<tr>
<td>XJCLLOG</td>
<td>environment value MUST be set to a unique file identifier for any JOB using xlist (see xrun).</td>
</tr>
</tbody>
</table>
6.2.4 xgdg

The command line tool xgdg is used to interface the XGDG subsystem.

Syntax

xgdg [options] [<args>]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--catalogue, -c</td>
<td>catalogues GDG files used in the job</td>
</tr>
<tr>
<td>--uncatalogue, -u &lt;n&gt;</td>
<td>uncatalogues file number &lt;n&gt;</td>
</tr>
<tr>
<td>--dlbl, -d</td>
<td>returns the real name of the GDG file specified in the argument in a dlbl_format. Argument must be specified in a dlbl_format.</td>
</tr>
<tr>
<td>--get&lt;gdg&gt;, -g&lt;gdg&gt;</td>
<td>returns the real name of the &lt;gdg&gt; GDG file specified in the argument.</td>
</tr>
<tr>
<td>--full, -f</td>
<td>returns the full path name of the &lt;gdg&gt; GDG file specified in the argument.</td>
</tr>
<tr>
<td>--info, -i</td>
<td>shows information about configuration for all files or for the specified one.</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>runs in verbose mode</td>
</tr>
</tbody>
</table>

Environment variables

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XJCLLOG</td>
<td>environment value MUST be set to a unique file identifier for any JOB using xlst (see xrun).</td>
</tr>
</tbody>
</table>

Notes

- xgdg command line is not required when files are accessed from a program run using "xrun".
- xgdg --catalogue MUST be issued at the end of scripts using GDG files
- xgdg --uncatalogue may be used only on catalogued versions (negative or zero)

Example

```bash
#!/bin/ish
# export XCLLOG=${$.Log}
rm -f $XCLLOG
filein=$(xgdg --full --get 0 $XSAM/FLAT/FILE01)
fileout=$(xgdg --full --get +1 $XSAM/FLAT/FILE01)
sort < $filein > $fileout
xgdg --catalogue
```
**Path name resolution**

Any reference (in configuration or in commands) to non-absolute path name, causes XGDG to consider the file under the $XVSAM path. Therefore to define a file like the following

```
define name=FLATCAT/FILE01.DAT
```

is the same as specifying the full path name including $XVSAM ($XVSAM/FLATCAT/FILE01.DAT)

The same appens using commands:

```
filein=$(xgdg --full --get 0 FLAT/FILE01)
```

is the same as:

```
filein=$(xgdg --full --get 0 $XVSAM/FLAT/FILE01)
```

Obviously, absolute paths are left unchanged:

```
filein=$(xgdg --full --get 0 /home/data/FLAT/FILE01)
```

**Configuration file**

This is the file which controls the behaviour of xgdg and it named xgdg.conf and contained in $HOME/etc. Configuration is specified using the following commands

- set
- define

**Parameter definition**

General parameters are defined using the set directive

**Syntax**

```
set <parameter>[=<value>]
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>If specified xgd runs in verbose mode</td>
</tr>
<tr>
<td>keep(&lt;condition&gt;)</td>
<td>Defines the test to be verified when deciding if a GDG file must be kept or deleted</td>
</tr>
</tbody>
</table>

**Files definition**

In order to have GDG working properly on a certain file it must be defined in xgdg.conf. If a file is not defined but GDG handling is requested for it, default values are assumed.

**Syntax**

```
define name=<fileid> [group=<gdg>] [record_length=<recl>]
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name=&lt;fileid&gt;</code></td>
<td>This parameter is mandatory. It defines the file identifier, which can be:</td>
</tr>
<tr>
<td></td>
<td>- an absolute path</td>
</tr>
<tr>
<td></td>
<td>- a path relative to $XVSAM</td>
</tr>
<tr>
<td><code>group=&lt;gdg&gt;</code></td>
<td>Specifies the GDG width. Default is 8 (eight)</td>
</tr>
<tr>
<td><code>record_length=&lt;recl&gt;</code></td>
<td>Specifies the record length for the file. If xrun runs in EBM mode, the parameter specifies the ddbname_LRECL parm (in lrecl is not specified in the dlb)</td>
</tr>
</tbody>
</table>

### An example

```bash
# configuration for xgdg
set keep($RC < 8)
set verbose
define name=CATALOG1/FILE01.DATA group=4
define name=CATALOG1/EXXP.BCKJHS group=4
define name=CATALOG2/EYYP.ADDRSF group=4 record_length=500
define name=CATALOG2/EZPO.PEXZXQ group=4 record_length=506
```
6.2.5 xlst

xlst manages printers in batch process in a VSE like flavour.

During batch executions xlst stores all printer commands in the JCL log file and at the end provides to associate each printer to assigned destination and issue print commands.

It can be called with different parameters to perform corresponding action.

XJCLLOG environment value MUST be set to a unique file identifier for any JOB using xlst (see xrun).

Syntax

- `--lst <list options>`
  - read printer options such as destination class disposition ...

- `--assgn <dd_name>`
  - assign printer file to specific destination

- `--alias <alias> <device>`
  - define an alias for device

- `--spool`
  - execute print commands

Environment variables

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XJCLLOG</td>
<td>environment value MUST be set to a unique file identifier for any JOB using xlst (see xrun).</td>
</tr>
</tbody>
</table>

Configuration File

Configuration file has to be placed in $HOME/etc and named xlst.conf. Configuration is specified using the following commands :

- set
- ignore
- map

Parameter definition

General parameters are defined using the set directive

Syntax

```
set <parameter>[ =<value>]  
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>verbose</td>
<td>if specified xlst runs in verbose mode</td>
</tr>
<tr>
<td>off</td>
<td></td>
</tr>
<tr>
<td>default_printer</td>
<td>if xlst cannot resolve destination device for printer it assign this value. If no printer is defined, spool command aborts.</td>
</tr>
<tr>
<td>no_default</td>
<td></td>
</tr>
</tbody>
</table>
spool_command | command performed for all printers
---|---
xspooladd -N

## Options definition

Using map or ignore settings you can customize your spool options command.

### Syntax

map <parameter>=<value>

ignore <parameter>

### Example : configure xlst to work with xspool

This is the xlst.conf to place in $HOME/etc.

```bash
set default_printer=01E
set spool_command=xspooladd -N
ignore LST
set verbose
map COPY=n
map CLASS=c
map DEST=u
map FND=f
```

### Example : VSE jcl

**Original VSE step**

```vse
* $$ LST CLASS=A,DISP=L,COPY=1,LST=FEF                                    */ DLBL ...     // DLBL FIL.MOVIN',,VSAM,CAT=VSAMCAT
// DLBL GEPIN,'FL200.W',,VSAM,CAT=FLATCAT
// EXEC PROGRAM,SIZE=200KAAMMGG=021227
AAMMG=021227
```

**Step converted**

```vse
# * $$ LST CLASS=A,DISP=L,COPY=1,LST=FEF                                    */
# xlst --lst "CLASS=A,DISP=L,COPY=1,LST=FEF"
# // DLBL FIL.MOVIN',,VSAM,CAT=VSAMCAT
# // DLBL GEPIN,'FL200.W',,VSAM,CAT=FLATCAT
# // EXEC PROGRAM,SIZE=200KAAMMGG=021227
# AAMMG=021227
# /*
```

© 2007 H.T.W.C. Srl
At EOJ step:

```bash
# /$ $S$ EOJ
#
# EOJ
xlist --spool
source $HOME/etc/xjobedit.csh
exit $RC
```
6.2.6 xrun

Since version 3.2 of XFRAME, a new program launcher, called xrun (lowercase), is deployed. This new launcher is more flexible than XRUN (uppercase) launcher.

Even if XRUN is still enclosed in XFRAME distribution, we strongly suggest to use xrun.

This command will start execution of the indicated program using the appropriate runtime system, if needed.

**Functionalities**

| Library loading                  | xrun may pre-load shared libraries by specifying the environment XRUN_LIBRARIES. I.e.
|                                | export XRUN_LIBRARIES=
|                                | "libone.so:/usr/lib/libtwo.so:$HOME/lib/libthree.so"
| Program preloading              | xrun may call program using a user defined loader. SQLLOADER is the default for Oracle programs, SOLDB2 for DB2 programs and SQLODBC for ODBC programs
| RDBMS connection                | preloading using SQLLOADER/SOLDB2/SQLODBC, xrun ensure the connection to the RDBMS
| GDG handling                    | xrun grants the GDG (Generation Data Group) functionalities
| EBM/MBM compatibility           | xrun also grants the compatibility with programs coming from an EBM(tm) (aka MBM(tm)) environment
| CATLG like function             | xrun may take care to simulate a CATLG disposition, creating an empty file according to the DD disposition

**Syntax:**

```
xrun [options] <program-name>
```

**Options:**

- `--sqlid<userid>, -s<userid>` specifies RDBMS userid and ask to connect
- `--connect,-c` explicitly ask to perform RDBMS connection
- `--noconnect,-n` explicitly ask to ignore connecting phase
- `--verbose,-v` runs in verbose mode
- `--help,-h` shows help screen

**Environment variables**

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XRUN_LIBRARIES</td>
<td>Specifies, separated by ':', paths and name of shared libraries to pre-load.</td>
</tr>
</tbody>
</table>
### XRUN_SQL_RDBMS
Specifies the type of RDBMS: oracle, db2 or odbc

### XRUN_SQL_DBID
If one of this variables is set XRUN starts SQLLOADER/SQLDB2/SQLODBC which connects the RDBMS using the userid specified by the environment.

i.e.

```bash
export XRUN_SQL_DBID= "scott/tiger@ORCL"
```

### XRUN_SQL_DATE_FORMAT
If this variable is set, the SQLLOADER asks the RDBMS to return date fields in the specified format.

i.e.

```bash
export XRUN_SQL_DATE_FORMAT= 'dd/mm/yyyy'
```

### XRUN_SQL_TIMESTAMP_FORMAT
If this variable is set, the SQLLOADER asks the RDBMS to return timestamp fields in the specified format.

i.e.

```bash
export XRUN_SQL_TIMESTAMP_FORMAT= 'yyyyy-mm-dd hh24.mi.ssxff'
```

### XRUN_MODE
If set to "EBM" xrun supplies to the program run, EBM compatible environments are set.

### XJCLLOG
Must be set to unique file identifier in order to keep track of JCL and use GDG

### XRUN_STORE_CLIENT_INFO
If set to 'YES' the SQLLOADER stores additional execution info in the table V$SESSION, in the format:

```bash
()
```

### XRUN_SHOW_DD
If set to 'YES' or 'FULL', xrun show all the dlbl_ environments (DD/DLBL) set up for the program

### XRUN_REDIRECT_SYSPRINT
If set to 'YES' xrun redirects the STDOUT on the SYSPRINT file specified by dlbl_SYSPRINT

### XRUN_SHOW_CARDS
If set to 'YES' xrun shows the data cards (SYSIN) provided to the program

### XRUN_SQL_LOADER
If set, xrun uses the specified SQL loader

i.e.

```bash
export XRUN_SQL_LOADER= "$HOME/objs/int/USERLOADER.int"
```

### XRUN_CONF
Path to the optional configuration file

---

**The configuration file**

Further customization may be done through the configuration file. XRUN looks for the file pathed by $XRUN_CONF or for a file named $HOME/etc/xrun.conf.

The configuration file supports the following instructions:
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set rdbms=&lt;db&gt;</code></td>
<td>Specifies the kind of database to be connected:</td>
</tr>
<tr>
<td></td>
<td>• oracle</td>
</tr>
<tr>
<td></td>
<td>• db2</td>
</tr>
<tr>
<td></td>
<td>• odbc</td>
</tr>
<tr>
<td></td>
<td><strong>set abend threshold=&lt;value&gt;</strong> defines the abend return</td>
</tr>
<tr>
<td></td>
<td>code threshold. When the return code is equal or greater</td>
</tr>
<tr>
<td></td>
<td>to the specified value, the program handled as abnormally</td>
</tr>
<tr>
<td></td>
<td>terminated, and CATLG dispositions, if active, are properly</td>
</tr>
<tr>
<td></td>
<td>handled. The default value is 12.</td>
</tr>
<tr>
<td></td>
<td><strong>set sql_loader=&lt;pgm&gt;</strong> if set xrun uses the specified SQL</td>
</tr>
<tr>
<td></td>
<td>loader program &lt;pgm&gt;</td>
</tr>
<tr>
<td></td>
<td><strong>set redirect_sysprint</strong> if set xrun redirects the STDOUT</td>
</tr>
<tr>
<td></td>
<td>on the SYSPRINT file specified by dlbl_SYSPRINT</td>
</tr>
<tr>
<td></td>
<td><strong>set show_dd</strong> if set xrun show all the dlbl_ environments</td>
</tr>
<tr>
<td></td>
<td>(DD/DLBL) set up for the program</td>
</tr>
<tr>
<td></td>
<td><strong>set show_cards</strong> if set xrun shows the data cards (SYSIN)</td>
</tr>
<tr>
<td></td>
<td>provided to the program</td>
</tr>
<tr>
<td></td>
<td><strong>set handle_catlg</strong> if set xrun handles the CATLG</td>
</tr>
<tr>
<td></td>
<td>disposition, creating an empty file wherever required</td>
</tr>
<tr>
<td></td>
<td><strong>define standalone &lt;pgm&gt;</strong> defines &lt;pgm&gt; as a standalone</td>
</tr>
<tr>
<td></td>
<td>program (no RTS or loader is invoked)</td>
</tr>
<tr>
<td></td>
<td><strong>set default_object_type=&lt;value&gt;</strong> defines the default</td>
</tr>
<tr>
<td></td>
<td>object type. Admitable values:</td>
</tr>
<tr>
<td></td>
<td>• &lt;none&gt;</td>
</tr>
<tr>
<td></td>
<td>• exe</td>
</tr>
<tr>
<td></td>
<td>If no value is specified, objects are run by the COBOL</td>
</tr>
<tr>
<td></td>
<td>runtime system</td>
</tr>
<tr>
<td></td>
<td>If value is set to &quot;exe&quot; programs are directly executed,</td>
</tr>
<tr>
<td></td>
<td>without any RTS</td>
</tr>
</tbody>
</table>

I.e.

```bash
# xrun.conf
# set rdbms=oracle
set redirect_sysprint
set show_dd
set show_cards
set sql_loader=belando
define standalone IEBGENER
define standalone RUNFTP
```
6.2.7 xvsamRts

xvsamRts is an embedded runtime system to execute batch COBOL programs, providing direct support to the XVSAM file system, through the standard COBOL I/O instructions (i.e., READ, WRITE, etc).

Depending on the XFRAME setup, xvsamRts provides runtime support for Microfocus COBOL or for ACUCOBOL.

**Configuration**

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XVSAM</td>
<td>path to the XVSAM repository root</td>
</tr>
<tr>
<td>XVSAMRTSDBG</td>
<td>if set to &quot;YES&quot; xvsamRts will produce a trace file named xvsamRts.log</td>
</tr>
<tr>
<td>XVSAMRTS0D</td>
<td>if set to &quot;YES&quot;, xvsaRts will add a carriage return char (\r - 0x0d) at the end of the line of printing files</td>
</tr>
<tr>
<td>XVSAM_VAR_FMT</td>
<td>specifies the format of variable record length files. The admitted values are: IBM (default) XFRAME</td>
</tr>
<tr>
<td>XVSAMRTS_CHECK_FLUSH</td>
<td>if set to &quot;YES&quot; xvsamRts will perform addition error checking on file flushing</td>
</tr>
<tr>
<td>XVSAMRTSFCC</td>
<td>it specifies the global FCC behaviour. Please look at the FCC section below.</td>
</tr>
<tr>
<td>COBLPFORM</td>
<td>it defines up to 12 printing channels in the form: setenv COBLPFORM &quot;c1:c2:......:c12&quot; see the your Microfocus ServerExpress documentation for further details</td>
</tr>
<tr>
<td>XVSAMRTSSIGHUP</td>
<td>if set to &quot;YES&quot;, xvsamRts will additionally handle SIGHUP signal.</td>
</tr>
<tr>
<td>ACUDBGSRV</td>
<td>address of the ACUCOBOL debugger. The format is depending in the mode set in ACUDBGMODE</td>
</tr>
<tr>
<td>ACUDBGMODE</td>
<td>mode for ACUCOBOL debugger. Admitted value: display (default) tref thin</td>
</tr>
</tbody>
</table>

**Files specification**

In order to provide to the programs running on the XVSAM RTS, all the information about files and their characteristics from the scripts, environments have to used.

For each file specified in the program to be run, an environment named `dlbl_FILENAME` must be provided. This environment will contain all the information required (path, catalog, disp, etc)
Feed Control Character (FCC) management

It is possible to set FCC at the global level or for each file separately.

Global setting is performed by defining the environment variable named `XVSAMRTSFCC`.

Individual settings for specified files may be done in different ways:

- setting environments named `XVSAMRTSFCC_FILE`. i.e.
  ```
  setenv XVSAMRTSFCC_FILE01 "ASA"
  ```

- by specifying the "fcc=" in the `dlbl_` environment. i.e.
  ```
  setenv `dlbl_FILE01 "FILE.01,cat=SYSLST,disp=(NEW,KEEP),fcc=ASA"
  ```

- forcing `XVSAMRTSFCC_FILE` directly in the batch COBOL program. i.e.
  ```
  DISPLAY 'XVSAMRTSFCC_FILE01' UPON ENVIRONMENT-NAMEDISPLAY 'ASA' UPON
  ENVIRONMENT-VALUE
  ```

Admitted FCC values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASA</td>
<td>add an ASA control character before each line</td>
</tr>
<tr>
<td>ASN</td>
<td>as ASA, but add newline (0a) after each line</td>
</tr>
<tr>
<td>AMA</td>
<td>as ASA, but put the control character in the first line position instead of adding it before</td>
</tr>
<tr>
<td>AMN</td>
<td>as AMA, but add newline (0a) after each line</td>
</tr>
<tr>
<td>AFP</td>
<td>print HTAFPHT as the first line, add newline (0a) after each line</td>
</tr>
<tr>
<td>FORCE</td>
<td>force the ASA management, wait the control character in the first byte of each line</td>
</tr>
<tr>
<td>PRINT</td>
<td>force the file be LINE SEQUENTIAL: remove trailing spaces and put newline (0a) after each line</td>
</tr>
</tbody>
</table>
6.2.8 Miscellaneous data utilities

**dbunload**

**Description**

The dbunload command line utility unloads data from one relational table to one flat file. With dbunload, you can unload all rows from an entire table or select columns and records by using PL/SQL language. The output records written by the dbunload utility are compatible as input to the SQLLDR oracle utility by using control file generated from xddiconv XFRAME utility, and to LOAD udb utility.

**Synopsis**

```
dbunload <table> [ <file>]```

**Parameters**

- **table** is the name of the table to download
- **file** is the flat file name, if omitted data are written to STDOUT

**Configuration file**

Configuration file is necessary to run dbunload utility. It must to be placed in $HOME/etc, be named dbunload.conf and has to contain oracle connection parameters, as following.

For Oracle

```
user=user
password=password
database_url=jdbc:oracle:thin:@linux01:1521:htora
structure_inquiry=[yes|no]
```

For UDB (DB2):

```
user=user
password=password
database_url=jdbc:db2://linux02:50000/htdb2
```

where:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>user for RDBMS</td>
</tr>
<tr>
<td>password</td>
<td>password for RDBMS</td>
</tr>
<tr>
<td>rdbms</td>
<td>default is oracle, for to db2 for UDB connection. Provide to setting correct jdbc driver for remote connection.</td>
</tr>
<tr>
<td>database_url</td>
<td>URL of database connection</td>
</tr>
<tr>
<td>structure_inquiry</td>
<td>(optional) enables override of columns data type. New datatype is read from UNLOADER_TABLE.</td>
</tr>
</tbody>
</table>

© 2007 H.T.W.C. Srl
Environment variables

UNLOADER_SELECT

It specifies the text file containing the SQL SELECT instruction to perform in place of the entire table unloading. I.e.

```
export UNLOADER_SELECT="/home/etc/unload_table01.sql"
```

where the file contains:

```
SELECT FIELD1, FIELD2, FIELD3
FROM MYTABLE
WHERE FIELD4 = 'AAAA'
OR FIELD3 = 01
```

The UNLOADER_TABLE definition

The UNLOADER_TABLE may be created to contain columns redefinition datatype. User have to generate this table in the RDBMS, if inquiry_structure is set to "yes" in the dbunload.conf file.

Table creation

In order to create UNLOADER_TABLE, the following DDL must be used

```
CREATE TABLE UNLOADER_TABLE
    (TABLE_NAME CHAR(60) NOT NULL,
     FIELD_NAME CHAR(60) NOT NULL,
     FIELD_TYPE CHAR(20),
     FIELD_BYTES NUMBER DEFAULT 0 NOT NULL
     PRIMARY KEY (TABLE_NAME, FIELD_NAME)
    )
```

Populating the table

In order to redefine some of the fields of one or more table, rows must be inserted in to the table. For each field to be redefined a row must be created specifying:

- table name
- field name
- new type
- length in bytes

I.e.

```
INSERT INTO UNLOADER_TABLE VALUES ('TB1824', 'RICZ_NU_EDIZ', 'SMALLINT', 2);
```

or DATE field managed as TIME in ORACLE database

```
INSERT INTO UNLOADER_TABLE VALUES ('TB1824', 'START_TIME', 'TIME', 8);
```
**DBUNLOAD data type**

The dbunload utility follows the general rules and conventions of DB2 on the data type attributes and the compatibility among the data types.

### Data type managed

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>The data field contains character data. The length is the specified field's length.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>A varchar field is a length-value datatype. It consists of a binary length subfield followed by a character string of the specified length</td>
</tr>
<tr>
<td>INTEGER</td>
<td>The data is a full-word binary integer. The length, in bytes, is based on the size of a LONG INT (4 bytes).</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>The data is a full-word binary integer. The length, in bytes, is based on the size of a INT. N.B. Oracle change SMALLINT in INTEGER datatype so it becomes 4 Bytes. SMALLINT columns have to be inserted into UNLOADER_TABLE table with 2 bytes as FIELD_BYTES to unload as 2 bytes.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>Data is in packed decimal format: two digits per byte, except for the last byte, which contains a digit and sign. It's length is calculated by field length ( / 2 + 1 )</td>
</tr>
<tr>
<td>DATE</td>
<td>Field contains character data that should be converted using the specified date mask: DD/MM/YYYY</td>
</tr>
<tr>
<td>TIME</td>
<td>Field contains character data that should be converted using the specified time mask: hh:mm:ss. TIME datatype is valid only for DB2 rdbms. If you have an Oracle database that contains DATE field used as TIME, you can manage this field as exception using UNLOADER_TABLE and settings columns as TIME length 8.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Datatype is an extension of the DATE datatype. It stores the year, month, and day of the DATE datatype, plus the hour, minute, and second values of the TIME datatype. Format: YYYY-MM-DD-HH24.MI.SS.SSXXFF</td>
</tr>
<tr>
<td>NULL</td>
<td>Column with NULL value clause has 1 byte with X'00' value if field has a valid contents instead has X'3F' (?) if field contents null value.</td>
</tr>
</tbody>
</table>

### An example

Supposing the following table description:

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE</td>
<td>NOT NULL</td>
<td>CHAR(4)</td>
</tr>
<tr>
<td>DATE.MOD</td>
<td>NOT NULL</td>
<td>DATE</td>
</tr>
<tr>
<td>SUB_CODE</td>
<td>NUMBER(7,2)</td>
<td></td>
</tr>
<tr>
<td>REF</td>
<td>NUMBER(5)</td>
<td></td>
</tr>
</tbody>
</table>

When the dbunload utility is run, user gets the following output:

```
# unloader TABLE01 $HOME/some/where/flat_file.dat
inquiry_structure=yes
Open Connection to : jdbc:oracle:thin:@localhost:1521:dbora
Record length 23
Download 122948 records
```
Close connection

Record length nnn is the length of each row in bytes.

Download nnnn records is the number of records unloaded

If you multiply record length for number of records unloaded obtain file size. The record produced will look as follows:

| pos | 1234| 5678|9|012|3 |
|-----|-----|-----| |   |   |
| rec | 0516|24/06/2004| G | | |
| hex | 3333|33232333333| 0042|0|001|0|
| hex | 0516|24F06F2004|007C0|00C0|0|
7 XSORT

XSORT 2 is a component of the XFRAME family, whose functionality is equivalent to that of the IBM DFSORT utility. It maintains the same syntax and offers the same services as its mainframe counterpart.

It offers all the capabilities of its predecessor (XSORT version 1 release 6), plus new options. Furthermore, new functionalities to handle UNIX native file in an easier way have been deployed with XSORT 2.

Files accessed by XSORT 2 can be flat UNIX files, as sequential and line sequential data files, and files managed by the XVSAM product, as KSDS, ESDS and RRDS data files.

The product itself is composed by the executable xsort.

Requirements

XSORT 2 is deployed since XFRAME 3.3.18.

What's new

The most important new features that XSORT 2 provides are:

- completely rewritten
- new multiple SORTIN handling algorithm
- new data cards parsing algorithm
- support for fields comparison
- support for field edit
- support for alternate collating sequences
- support for line sequential files
- support for file format conversion
7.1 Usage

XSORT 2 may be invoked from command line or from scripts using:

```
xsort [options]
```

or

```
xrun xsort [options]
```

Options

The following options may be specified:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>none</td>
<td>-</td>
<td>shows the XSORT version and exits</td>
</tr>
<tr>
<td>verbose</td>
<td>none</td>
<td>-</td>
<td>sets the level of XSORT messaging to VERBOISE</td>
</tr>
<tr>
<td>trace</td>
<td>none</td>
<td>-</td>
<td>sets the level of XSORT messaging to TRACE</td>
</tr>
<tr>
<td>mode</td>
<td>mvs</td>
<td>vse mvs</td>
<td>sets the compatibility mode</td>
</tr>
<tr>
<td>y2k</td>
<td>50</td>
<td>any valid number</td>
<td>sets the Y2K threshold</td>
</tr>
<tr>
<td>type</td>
<td>flat</td>
<td>flat vsam</td>
<td>sets the file type for the following file(s)</td>
</tr>
<tr>
<td>record</td>
<td>fixed</td>
<td>fixed variable lineseq</td>
<td>sets the record type for the following file(s)</td>
</tr>
<tr>
<td>lrecl</td>
<td>none</td>
<td>any valid record length</td>
<td>sets the record length for the following file(s)</td>
</tr>
<tr>
<td>sortin</td>
<td>none</td>
<td>any valid path name</td>
<td>adds a SORTIN file</td>
</tr>
<tr>
<td>sortout</td>
<td>none</td>
<td>any valid path name</td>
<td>adds a SORTOUT file</td>
</tr>
<tr>
<td>varfmt</td>
<td>IBM</td>
<td>ibm xframe</td>
<td>forces variable record length format</td>
</tr>
</tbody>
</table>

All options may be specified with or without "--" prefix.

I.e.

```
xsort version
```

or

```
xsort --version
```

Files specification

SORTIN/SORTOUT files may be specified on command line using the parameters `sortin` and `sortout`, or through the environment variables `dlbl_SORTIN`, `dlbl_SORTINn` (where `n` is a number from 1 to 9) and `dlbl_SORTOUT`.

The value of the `dlbl_` variables may be a pathname or an XFRAME like file definition.

The disposition handling is granted only by specifying files through `dlbl_` environments in the XFRAME format.
680

XFRAME User's Guide

PARMs specifications
Other parameters may be provided provided through the environment variable named XRUN_PARM (the same as
mainframe PARM value):
Parameter

Description

CENTWIN

sets the Y2KPAST limit

Examples
Example 1
Two SORTIN and the SORTOUT files are provided using the command line parameters:

xsort --type=fixed lrecl=100
\
--sortin=$HOME/data/FILE01.dat \
--sortin=$HOME/data/FILE02.dat \
--sortout=$HOME/data/FILEOUT.dat <</*
SORT FIELDS=COPY
/*

Example 2
2 SORTIN files are provided using dlbl_ variables, another SORTIN and the SORTOUT are provided using the command
line parameters:

export dlbl_SORTIN1="$HOME/data/FILE01.dat"
export dlbl_SORTIN2="FILE02.dat,cat=FLATCAT,disp=(OLD,KEEP),type=flat"
xsort type=fixed lrecl=100
\
sortin=$HOME/data/FILE03.dat \
sortout=$HOME/data/FILEOUT.dat <</*
SORT FIELDS=COPY
/*

Example 3
A typical script usage:

setenv dlbl_SORTIN1 "FILE01.IDX,cat=VSAMCAT,disp=(OLD,KEEP),type=vsam"
setenv dlbl_SORTIN2 "FILE02.TMP,cat=FLATCAT,disp=(OLD,DELETE),type=flat"
setenv dlbl_SORTOUT "FILEXX.OUT,cat=FLATCAT,disp=(NEW,KEEP),type=flat,gdg=+1"
xsort lrecl=100 <</*
SORT FIELDS=COPY
/*

© 2007 H.T.W.C. Srl


## 7.2 Configuration

XSORT 2 may accept some options as default by providing using the following environment variables:

<table>
<thead>
<tr>
<th>Environment name</th>
<th>Default</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSORT_LOG</td>
<td>none</td>
<td>NONE VERBOSE TRACE</td>
<td>controls the level of XSORT messaging</td>
</tr>
<tr>
<td>XSORT_MODE</td>
<td>MVS</td>
<td>MVS VSE</td>
<td>sets the compatibility mode</td>
</tr>
<tr>
<td>XSORT_ALTSEQ_CARDS</td>
<td>ASCII</td>
<td>ASCII EBCDIC</td>
<td>sets the encoding of ALTSEQ codes recognition</td>
</tr>
<tr>
<td>XSORT_TMPDIR</td>
<td>-</td>
<td>Any valid path</td>
<td>sets the temporary path. If not set, XSORT looks, in the order reported, for $TMP, $TMPDIR and $PWD</td>
</tr>
<tr>
<td>XSORT_Y2PAST</td>
<td>50</td>
<td>two digits number</td>
<td>sets the default Y2PAST limit</td>
</tr>
<tr>
<td>XVSAM_VAR_FMT</td>
<td>IBM</td>
<td>IBM XFRAME</td>
<td>sets the variable record length format. If XVSAM_VAR_FMT is not set, xsort looks for XSORT_VAR_FMT. If it is not set it assumes IBM mode.</td>
</tr>
<tr>
<td>XSORT_VAR_FMT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSORT_IGNORE_RDW_ON_VSAM</td>
<td>NO</td>
<td>YES NO</td>
<td>if set to YES, XSORT handles VSAM variable length records, without taking care of the 4 byte RDW (Record Descr. Word).</td>
</tr>
</tbody>
</table>
7.3 Control statements

XSORT execution is controlled by means of its control statements. These statements are read on standard input (STDIN).

Control cards have been kept equal to the control cards accepted by IBM DFSORT(TM) due to compatibility reason, but, XSORT 2 has no need for line continuation char at column 72, and cards may be specified in both uppercase or lowercase characters.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SORT</td>
<td>Sorts the input file(s) according to the specified fields</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>define the rules to define which records must be written in the output file</td>
</tr>
<tr>
<td>OMIT</td>
<td>define the rules to define which records must be omitted in the output file</td>
</tr>
<tr>
<td>OUTREC</td>
<td>defines the record structure of the output file</td>
</tr>
<tr>
<td>SUM</td>
<td>instructs XSORT to perform summarization functions on some fields</td>
</tr>
<tr>
<td>OPTION</td>
<td>sets XSORT option</td>
</tr>
<tr>
<td>RECORD</td>
<td>defines the type and length of the records being processed</td>
</tr>
<tr>
<td>ALTSEQ</td>
<td>changes the alternate translation table</td>
</tr>
</tbody>
</table>
7.3.1 SORT control statement

The SORT control statement must be used when a sorting application is performed; this statement describes the control fields in the input records on which the program sorts. A SORT statement can also be used to specify a copy application.

**FIELDS parameter**

Describes the control fields for which records must be sorted. Requires four facts about each control field in the input records:

<table>
<thead>
<tr>
<th>Facts</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>the position of the field within the record</td>
</tr>
<tr>
<td>l</td>
<td>the length of the field</td>
</tr>
<tr>
<td>f</td>
<td>the format of the data in the field (may be omitted if FORMAT is specified). It may assume any supported format. See the FORMAT parameter for the list of supported formats.</td>
</tr>
<tr>
<td>s</td>
<td>sequence into which the field is to be sorted (A or D)</td>
</tr>
</tbody>
</table>

The ordering sequence s may be:

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ascending</td>
</tr>
<tr>
<td>D</td>
<td>descending</td>
</tr>
</tbody>
</table>

**FORMAT parameter**

Defines the format for the control fields. This option may be used only if all the fields have the same format.

Supported formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Length in bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>1-4092</td>
<td>unsigned binary</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>1-4092 character</td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>1-4096 EBCDIC character</td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td>1-4096 character with alternate collating sequence</td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td>1-32 signed packed decimal</td>
<td></td>
</tr>
<tr>
<td>ZD</td>
<td>1-32 signed zoned decimal</td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td>1-256 signed fixed-point</td>
<td></td>
</tr>
<tr>
<td>Y2C or Y2Z</td>
<td>2 two digit char or zoned year</td>
<td></td>
</tr>
<tr>
<td>Y2B</td>
<td>1 two digit binary year</td>
<td></td>
</tr>
<tr>
<td>Y2P</td>
<td>2 two digit packed year</td>
<td></td>
</tr>
</tbody>
</table>

**COPY parameter**

When the COPY parameter is specified no SORT occurs on input records.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[NO]EQUALS</td>
<td>only for compatibility reason: XSORT 2 always preserve the original sequence</td>
</tr>
<tr>
<td>SKIPREC=k</td>
<td>skips the first k input records while processing</td>
</tr>
<tr>
<td>STOPAFT=a</td>
<td>stop reading input after a records</td>
</tr>
<tr>
<td>Y2PAST=y</td>
<td>sets the Y2K threshold to y</td>
</tr>
<tr>
<td>FILES=y</td>
<td>sets number of SORTIN files (VSE mode) y</td>
</tr>
</tbody>
</table>
7.3.2 INCLUDE control statement

Use an INCLUDE statement if you want only certain records to appear in the output data set. The INCLUDE statement selects the records you want to include.

You can specify either an INCLUDE statement or an OMIT statement in the same DFSORT run, but not both.

**COND parameter**

The parameter COND may assume 3 values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>all the records are included</td>
</tr>
<tr>
<td>NONE</td>
<td>no record is included</td>
</tr>
<tr>
<td>logical expression</td>
<td>If the logical expression is true for a given record, the record is included in the output data set. A logical expression consists of one or more relational conditions.</td>
</tr>
</tbody>
</table>

**Relational conditions**

```
p1, m1, fi, (EQ | NE | GT | GE | LT | LE), (p2, m2, f2 | constant)
```

Or, if the FORMAT=f operand is used:

```
p1, m2, (EQ | NE | GT | GE | LT | LE), (p2, m2 | constant)
```

Relational conditions can be logically combined, with AND or OR, to form a logical expression.

If they are combined, the following rules apply:

- AND statements are evaluated before OR statements unless parentheses are used to change the order of evaluation; expressions inside parentheses are always evaluated first. (Nesting of parentheses is limited only by the amount of storage available.)
- The symbols & (AND) and | (OR) can be used instead of the words.

**Field definition**

Comparisons may be grouped in:

- field-to-field
- field-to-constant

The first field in the comparison is specified with:
Facts | Description
---|---
p1 | the position of the field
m1 | the length of the field
f1 | the format of the data in the field (may be omitted if FORMAT is specified)

If a comparison file-to-field is required, the second field is specified using:

Facts | Description
---|---
p2 | the position of the field
mm | the length of the field
f2 | the format of the data in the field (may be omitted if FORMAT is specified)

otherwise, if a field-to-constant comparison is required, the constant may be specified using:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>C'string'</td>
<td>character string</td>
<td>C'ABC'</td>
</tr>
<tr>
<td>X'hex value'</td>
<td>hexadecimal string</td>
<td>H'F1F2'</td>
</tr>
<tr>
<td>Y'year'</td>
<td>year</td>
<td>Y'97'</td>
</tr>
<tr>
<td>+</td>
<td>-number</td>
<td>decimal number</td>
</tr>
</tbody>
</table>

Padding and truncation

In a field-to-field comparison, the shorter compare field is padded appropriately. In a field-to-constant comparison, the constant is padded or truncated to the length of the compare field.

Character and hexadecimal strings are truncated and padded on the right.

The padding characters are:

- X'40' For a character string
- X'00' For a hexadecimal string.

Decimal constants are padded and truncated on the left. Padding is done with zeros in the proper format.

Supported comparisons

The supported field-to-constant comparison are:

<table>
<thead>
<tr>
<th>Field format</th>
<th>Decimal number</th>
<th>Character string</th>
<th>Hexadecimal String</th>
<th>Year string</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
</tbody>
</table>
### The supported field-to-field comparison are:

<table>
<thead>
<tr>
<th>Field format</th>
<th>BI</th>
<th>CH</th>
<th>CE</th>
<th>AQ</th>
<th>ZD</th>
<th>PD</th>
<th>FI</th>
<th>Y2C</th>
<th>Y2B</th>
<th>Y2P</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CE</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AQ</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZD</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PD</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FI</td>
<td></td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2C</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2B</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y2P</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.3.3 OMIT control statement

Use an OMIT statement if you do not want certain records to appear in the output data set. The OMIT statement selects the records you want to exclude.

You can specify either an INCLUDE statement or an OMIT statement in the same DFSORT run, but not both.

**COND parameter**

The parameter COND may assume 3 values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>all the records are omitted</td>
</tr>
<tr>
<td>NONE</td>
<td>no record is omitted</td>
</tr>
<tr>
<td>logical expression</td>
<td>If the logical expression is true for a given record, the record is omitted from the output dataset. If the logical expression is a conjunctive condition, it is treated as an OR. A logical expression consists of one or more relational conditions.</td>
</tr>
</tbody>
</table>

**Details**

For details about relation conditions, field definition, padding and truncation and supported comparisons, please refer to the INCLUDE control statement.
7.3.4 OUTREC control statement

```
OUTREC   FIELDS=(
    [c:]s | p, m[,a] | p |
    p, m, HEX | p, m, f[,edit] | p, m, f, to |
    [, CONVERT | VTOF | FTOLS ]
```

The OUTREC control statement allows you to reformat the input records before they are output. That is, to define which parts of the input record are included in the reformatted output record, in what order they are to appear, and how they are to be aligned.

You do this by defining one or more fields from the input record. The reformatted output record consists of those fields only, in the order in which you have specified them, and aligned on the boundaries or in the columns you have indicated.

Furthermore, it is possible to insert constants, separators and to perform edit and/or conversions on data fields

**Column indicator**

The column indicator `c:` specifies the offset to place the following data field. It must specify a position greater than the sum of the length of previously declared field.

**Constants and separators**

Constants may be declared as follows:

<table>
<thead>
<tr>
<th>Declarative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>'string'</code></td>
<td>specifies a string constant</td>
</tr>
<tr>
<td><code>X'hex string'</code></td>
<td>specifies an hexadecimal constant</td>
</tr>
<tr>
<td><code>X</code></td>
<td>specifies a NULL separator (binary 0x00)</td>
</tr>
<tr>
<td><code>Z</code></td>
<td>specifies a blank separator (space character, binary 0x20)</td>
</tr>
</tbody>
</table>

Each constant may be prefixed by one or more digits, to indicate a multiplier. I.e. "20Z" means 20 spaces, "2'CABC" produces "ABCABC"

**Field declaration**

Each input field to be reported on the output record must be identified by its position `p`, by its length `m`. Optionally its format `f` and an eventual edit mask `edit` may be specified.

Valid edit formats:

- ZD
- PD
- BI
- FI
Format mask specifiers:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR</td>
<td>CR is printed if the value is negative; otherwise, two blanks are printed</td>
</tr>
<tr>
<td>S</td>
<td>sign</td>
</tr>
<tr>
<td>I</td>
<td>a blank is printed when the digit is an insignificant zero</td>
</tr>
<tr>
<td>T</td>
<td>the digit is always printed</td>
</tr>
</tbody>
</table>

Any other character outside the table above is shown at the specified position.

**Record format conversion**

If the CONVERT clause or the VTOF clause is specified, output record is converted from variable length to fixed length record.

If the FTOLS clause is specified, output record is converted into line sequential.
7.3.5 SUM control statement

```
SUM     FIELDS=[(p,m,f) | (p,m),FORMAT=f | NONE | (NONE) ]
```

The SUM control statement specifies that, whenever two records are found with equal sort or merge control fields, the contents of their summary fields are to be added, the sum is to be placed in the first record, and the other record is to be deleted. The EQUALS or NOEQUALS have no effect, because always the first record is used to summarize.

**Field definition**

Each field to summarize must be defined providing its position p in the input record, its length m and its format f. If all the fields have the same format, the parameter FORMAT may be used, omitting the format specification for each field.

Possible formats:

<table>
<thead>
<tr>
<th>Format</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BI</td>
<td>2, 4, or 8 bytes</td>
<td>Unsigned binary</td>
</tr>
<tr>
<td>FI</td>
<td>2, 4, or 8 bytes</td>
<td>Signed fixed-point</td>
</tr>
<tr>
<td>PD</td>
<td>1 to 16 bytes</td>
<td>Signed packed decimal</td>
</tr>
<tr>
<td>ZD</td>
<td>1 to 18 bytes</td>
<td>Signed zoned decimal</td>
</tr>
</tbody>
</table>
7.3.6 OPTION control statement

The OPTION control statement allows you to override some of the options provided in XSORT 2 configuration.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[NO]LIST</td>
<td>specifies whether or not XSORT must print out the control cards provided</td>
</tr>
<tr>
<td>[NO]EQUALS</td>
<td>only for compatibility reason: XSORT 2 always preserve the original sequence</td>
</tr>
<tr>
<td>SKIPREC=k</td>
<td>skips the first k input records while processing</td>
</tr>
<tr>
<td>STOPAFT=a</td>
<td>stop reading input after a records</td>
</tr>
<tr>
<td>Y2PAST=y</td>
<td>sets the Y2K threshold to y</td>
</tr>
<tr>
<td>FILNM=y</td>
<td>sets file names (VSE mode) y</td>
</tr>
</tbody>
</table>
7.3.7 RECORD control statement

The RECORD control statement can be used to specify the type and lengths of the records being processed, and the minimum and average record lengths for a variable-length sort. Format may assume the following values:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>fixed length</td>
</tr>
<tr>
<td>FB</td>
<td>fixed length</td>
</tr>
<tr>
<td>V</td>
<td>variable length</td>
</tr>
<tr>
<td>VB</td>
<td>variable length</td>
</tr>
<tr>
<td>LS</td>
<td>line sequential</td>
</tr>
</tbody>
</table>
7.3.8 ALTSEQ control statement

The ALTSEQ control statement can be used to change the alternate translation table (ALTSEQ table). Any modifications you specify are applied to the standard ASCII translation table.

The ALTSEQ table can be used to apply an alternate collating sequence for SORT, MERGE, INCLUDE or OMIT fields with format AQ (or format CH with CHALT in effect). In this case, the ALTSEQ table is used to change only the order in which data is collated, not the data itself. If you specify AQ (or CH with CHALT) without specifying an ALTSEQ control statement, DFSORT uses the installation default ALTSEQ table.

**CODE encoding**

By default, or specifying the parameter ASCII, XSORT 2 recognize the ff and tt as ASCII values.

So, for example, if user wants to specify that the character ‘A’ (ASCII X’41’) is to be placed after the character ‘B’ (ASCII X’42’) he should specify:

```
ALTSEQ CODE=(4142)
```

In order to save the user to convert his existing DFSORT(TM) control cards, the parameter EBCDIC may be used. In this case user may leave ff and tt encoded with EBCDIC values, XSORT 2 will automatically convert the codes in ASCII values. So, for example, if in the original control cards was specified:

```
ALTSEQ CODE=(C1C2)
```

in order to have ‘A’ after ‘B’, control cards may remain unchanged by specifying

```
ALTSEQ CODE=(C1C2), EBCDIC
```

The EBCDIC handling may be globally set up, by means of the environment variable XSORT_ALTSEQ_CARDS (see Configuration)

The following sample, shows how to sort, in ASCII ordering, inverting the collating sequence of ‘1’ with ‘2’, and ‘A’ with ‘B’, providing mainframe-like altseq encoding:

```plaintext
setenv XSORT_ALTSEQ_CARD "EBCDIC"
setenv dlbl_SORTIN "FILE01.IDX,cat=VSAMCAT,disp=(OLD,KEEP),type=vsam"
setenv dlbl_SORTOUT "FILE02.TXT,cat=FLATCAT,disp=(OLD,KEEP),type=flat"
setenv dlbl_SORTOUT "FILE03.OUT,cat=FLAT,disp=(NEW,KEEP),type=flat,grp=1"
xsort lrec=100 <<*/
 SORT FIELDS=FIELDS(1,8,AQ)
 ALTSEQ CODE=(F1F2,C1C2)
/*
```
7.3.9 END control statement

```
END
```

The END control statement allows DFSORT to discontinue reading the control cards.
7.4 VSE mode specific control statements

**INPFIL control statement**

```
INPFIL   BLKSIZE=ss
```

Does nothing. This option, in VSE more, is supported only for compatibility reason.

**OUTFIL control statement**

```
OUTFIL   BLKSIZE=ss
```

Does nothing. This option, in VSE more, is supported only for compatibility reason.
8 XBM

8.1 Introduction

This book is intended to help you to operate with XBM 4.2.0.

XBM (the XFRAME Batch Manager) is a cross-platform software solution to control, schedule and supply facilities to scripts execution.

Both UNIX an Windows systems can run user batch scripts to control execution of batch programs, and they supply a very flexible environment with just few facilities to control batch execution.

Normally these features supplied by this system are good enough for native applications, which have been designed for that systems, and take advantage of the system structure, but sometimes the flexibility of these architecture can be really dangerous for batch scripts, coming from Mainframe systems.

In fact, mainframes (as z/OS or VSE) let their jobs run in a really rigid environment, in order to grant some security and integrity features. Therefore JCLs coming from mainframe system, even if migrated into a new language (i.e. shell script, windows scripting language, java etc.), have the logical structure of mainframe JCL, and running them without a driver can be really dangerous.

XBM is born to avoid conflicts in JCLs executions in the new systems, and to supply the some features, that system cannot supply without a big programming effort in scripts.

XBM core and consoles are a pure Java coding, therefore they can be run on each system supporting the Java platform.

The script interface is obtained through the standard system features.
8.1.1 Features

XBM supplies the following features:

- configurable execution engines
- centralized console log
- command line based remote operation control
- GUI based remote console
- step level execution monitoring
- priority based FIFO stand by queue (SBQ)
- job name based concurrency handling
- static and dynamic virtual resources handling
- dynamic SBQ handling
- SBQ backup and restore
- SYSLST handling
- RDBMS based jcl tracking
- user handling
- Java based pluggable reports
- cross engine persistent JOB VARIABLES
- operator driven accept/reply system

Since 1.1.0:

- on demand engines
- operator reserved engines

Since 2.0

- task scheduler
- improved GUI interface
- message localization

Since 3.0

- chains
- calendar based tasks
- data source manager

Since 3.3

- stand by queue mirroring
- job-queue advanced editing
Since 3.4
  • licensing support

Since 4.0
  • Event handling
  • FTP browsing
8.2 Concepts

XBM is a daemon process always running in order to control the execution of JCLs submitted. Obviously it can control only those JCLs scheduled to it.

In order to understand the XBM behaviour some concepts must be explained.

Data source

The datasource is the database schema containing all the data for each session defined in the xbm network.

Session

A session is an entry in the datasource identifying the single daemon process.

Core

XBM core in the main thread controlling each resource in the XBM. Even if user has no direct interaction with this object, it is the most significative part of the product: in fact, it controls and synchronizes all the objects in XBM.

NetServer

This is the subsystem dedicated to remote consoles communication: all the input for XBM passes through it.

Consoles

Both GUI based remote console and command line based tools are treated as remote consoles. All these tools allow access to the XBM system, in order to submit and control JCLs execution.

JCL

With the term JCL, job or script we refer to standard UNIX/Linux shell scripts (C/Bourne/Korn shell) running under the control of XBM.

Requestes

Each operation in XBM is a request, which is identified by a unique request ID. Therefore each JCL submit is identified by a single request ID, which identifies it for its life cycle.

Chains

A chain is concatenation of steps. Each steps is composed by one or more task running on different sessions (systems). Each step may be related to others in order to decide when to execute it.

Task

Each entry in the scheduler module is called "Task". A task is a submit request delay and/or iterated in time.

Task may be executed:

- only once at a predefined time
- on a minute basis interval (i.e. every 10 minutes)
- once to the hour
- once to the day
- once to the month
– once to the year
– on a calendar base

Depending on execution mode above mentioned, date and/or time must be supplied.

When defining task user may exclude from execution:

– certain hours of the day
– certain days of the week
– certain days of month
– certain months

When defining a calendar based task a calendar must be supplied.

**Engines thread**

XBM has a configurable number of threads (engines) monitoring JCLs execution: each engine monitors a single JCL. Therefore the number of JCLs running under its control is equal to the number of defined and enabled engines.

Engines can be enabled and disabled, so it is possible to determine how many concurrent JCLs can run in a certain moment on the system.

Engines are identified by a progressive number from 0 (zero) to N, where N is the number of engines defined.

Engines may also be referenced with an alias. An alias is a case-sensitive alphanumerical string. Alias cannot be defined as "all".

An engine may be defined "on demand". It means that an "on demand" engine is never assigned by default. In order to use it, request must specify its number or alias when submitted.

An engines may be reserved for an operator. More than one engine may be reserved for one operator.

When an operator has one or more reserved engine, his own requests are assigned by default to one of his reserved engines. He can use a non reserved engine by specifying its name or alias when submitting a request.

A reserved engine cannot be used by other operators.

Reserved engines may be also defined as "on demand", that means that owning operator can use them only by request.

Moreover, on UNIX systems, each engine is identified by a nice level, that means that all JCLs started by it are run with the specified nice level.

**SBQ - the stand by queue**

When all enabled engines (or the requested engine) are busy (or disabled), or static resources (see below) for JCL are in use, execution is delayed. In this situation the execution request is put into the stand by queue (SBQ), that is a FIFO queue where JCLs wait for their execution time. Each request in the queue is identified by a priority level (1 up to 10, 1 is the highest), that affects the order of de-queing.

When XBM detects the a request must be de-queued from SBQ and all requested resources are free, and an engine (or the requested engine) is available to process, it removes the request from SBQ and passes it the elected engine in order to execute it.

Requests collected in SBQ can be pause, resumed and purged. Moreover requests may be moved up and down in the queue and changed.

SBQ is mirrored on a file named xbm.session_name.s bq.dat where (session_name is the name of XBM instance) in the directory defined core.s bq.backup.path in the configuration file. Please refer to the configuration section of this guide to get more information.

When XBM restarts alway try to recover existing SBQ mirrors, in order to resume previously unresolved requestes.
Resources

As XBM cannot directly integrate the host system to check real system resources such as files or RDBMS tables, the concept of virtual resource has been developed. A virtual resource is just a lockable identifier maintained by XBM: each JCL can make use of this identifier, defining and locking it. For example, if the JCL "A" is locking the resource named "FILESET1", all JCLs which want to use the same resource must wait to run until the can gain lock on that resource.

Each resource is identified by its name.

Resources can be declared statically or dynamically. Statically declared resources are defined and locked using static tags on JCL; that XBM can detect at submission time. Dynamically declared resources are defined and locked by JCL using the interface commands: XBM can detect the use only during run time.

The difference is that a JCL statically defining a resource is not de-queued and executed until the resource is available for it (no engine is occupied by the JCL waiting for the resource). When a JCL uses dynamically a resource, occupies an engine while waiting for the resource.

Events

Events are logical conditions that may cause a JCL to be enabled to start. When submitted, JCLs requiring one or more event to occur, are automatically put on hold. Once all the required events are detected, the JCL is enabled to start.
8.2.1 The database

XBM store information concerning JCLs execution in a relational database. All data types used by XBM are SQL ANSI standard, and access to the RDBMS is made trough JDBC, so each ANSI DB supporting JDBC can be used.

More sessions of XBM may access the same database, that means that all the information concerning the job execution of several XBM instances in the network can be tracked in one single database.

Furthermore, even if you have more than one XBM instance, you only need to define users once, and simply assign the desired one to the required instance, with the necessary authorization level.

A sample scenario:

The machine PROD serves the production for the applications A01 and A02. The machine DEV serves both development and test environments.

On PROD machine two instances of XBM run to serve application A01 and A02. On DEV machine two XBM instances run to serve the development environment and the test environment. All these instance may access the same database and store their information on a single destination.

The XBM users are define only once: someone is assigned only to one instance (i.e. A01 operators); someone is assigned to multiple instances (i.e. developers are assigned to both development and test instances); other ones are assigned to all instances (i.e. administrators are assigned to all instances).

Pluggable reports

Data stored in database can be extracted and printed using the features offered by the RDBMS or using the pluggable report mechanism provided by XBM remote console.

Pluggable reports are Java objects which are subclasses of abstract class com.hite.xbmReport.

For more information concerning writing pluggable reports please refer to Plugins chapter.
8.2.2 Database drivers and URL

XBM uses the JDBC technology to connect the database system.

Following table is a reference of drivers and URL samples:

<table>
<thead>
<tr>
<th>Database</th>
<th>Driver class name</th>
<th>URL sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>oracle.jdbc.driver.OracleDriver</td>
<td>jdbc:oracle:thin:@hpux01:1521:XBM</td>
</tr>
<tr>
<td>IBM UDB</td>
<td>com.ibm.db2.jcc.DB2Driver</td>
<td>jdbc:db2://linux01:50000/XBM</td>
</tr>
<tr>
<td>MySQL</td>
<td>com.mysql.jdbc.Driver</td>
<td>jdbc:mysql://linux01:3306/xbm</td>
</tr>
</tbody>
</table>

For further information on JDBC URL definition, please refer to your JDBC provider documentation.
8.2.3 Users management

XBM has handles users that can access the resources of one or more sessions. Each user is identified by:

- name
- password
- role

Obviously name and password allow XBM to recognize who the user is.

Once defined, the users should be assigned to one or more sessions, with the desired role. This means that the same user may have different roles on different sessions (i.e. to be administrator for session A and operator for session B).

The role has been provided to identify what user can do and what user cannot do. Two roles are actually recognized:

- administrator
- operator

Users with administrator role can do everything on the XBM instances.

Users defined as operator can only:

- submit jobs
- pause, resume, purge, reply and kill requests issued by themselves
- report XBM status
- perform reports

and cannot:

- shutdown XBM
- purge, pause, resume, kill and reply to requests issued by other users

Users definition

Users can be defined by administrator using XBM Datasource Manager.

Once a user has been define, he must be assigned to the desired session(s).
8.2.4 Events

Events provide to the user the possibility to enable the execution of a job only and when a certain condition occurs.

Therefore, when a job is submitted, it may require the presence of certain events to run. If at the submission time, the events required by the job, are not verified, the job is paused (put on hold) until all the events requested occur.

XBM supports the following categories of events:

- Logic events
- Job events
- File events

Different event categories may be mixed in the definition of a job.

Requiring events

Events are required by specifying XBM static tags inside the job script. For further information please refer to the JCL Interface section.
8.2.4.1 Logic events

Logic events are a kind of XBM internal variables. Depending on the definition, and eventually on the value, of one or more variables, a job may be conditioned.

In order to require logic event conditioning for a certain job the wait_event item tag have to be used. I.e.

```
#!/bin/csh
#
# @xbm wait event VAR1
# @xbm wait event VAR2 value = 0 or value = 10
# @xbm wait event VAR3 value eq "OK" or value ne "KO"
#
```

Using the tag specified in the sample above, will cause the job to be kept on hold, until all the conditions are verified:

- the event variable VAR1 is defined, with any value
- the event variable VAR2 is defined and its value is numerically equal to 0 (zero) or 10 (ten)
- the event variable VAR3 is defined and its value is equal to the string "OK" and not equal to the string "KO"

Defining event variables

Event variables are set using the `xbmc event set`, `event unset`, and `event reset` command. For further detail, please refer to the Command line section.
8.2.4.2 Job events

Job events are caused by the execution of jobs and by their return codes. A job execution may be conditioned depending of the execution and eventually on the result, of other jobs.

When requiring job events a time range must be provided. The time range specifies the time interval for the condition: only if the condition specified is always true in the time range provided, the job is executed.

In order to require job event conditioning for a certain job the wait_event job tag have to be used. I.e.

Time range may be specified requiring:

<table>
<thead>
<tr>
<th>Time Range Type</th>
<th>Time Range Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>submit</td>
<td>time range starts from the job submission time</td>
</tr>
<tr>
<td>today</td>
<td>time range starts from the 0:00 AM of the current day</td>
</tr>
<tr>
<td>reset</td>
<td>time range starts from the last issue of xbmc reset command</td>
</tr>
<tr>
<td>hh:mm</td>
<td>time range starts from specified hour</td>
</tr>
</tbody>
</table>

I.e.:

```csh
#!/bin/csh
# @xbm wait job JOB01.csh since reset
# @xbm wait job JOB02.csh since today rc = 0
# @xbm wait job JOB03.csh since 15:30 max <= 12 and min >=4
```

In the sample above, job will start if:

- JOB01.csh has run one or more time, since last reset command
- JOB02.csh has run one or more time, always returning a return code equal 0 (zero), since the 0:00 AM
- JOB03.csh has run one or more time, always returning a maximum return code less equal 12 and a minimum return code greater equal to 4, since 15:30
8.2.4.3 File events

File events may be used to condition the execution of a job depending on the presence and eventually on the properties of a specified file. XBM may simply test the availability or check for the size and/or the permission of a file, using the follow event modifies:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notempty</td>
<td>File length must be greater than zero</td>
</tr>
<tr>
<td>read</td>
<td>File must be readable</td>
</tr>
<tr>
<td>write</td>
<td>File must be writable</td>
</tr>
</tbody>
</table>

I.e.

```bash
#!/bin/csh
@xbm wait file /var/tmp/upload.txt read
@xbm wait file /etc/passwd notempty readable
```

In the sample above, job waits for:

- file `/var/tmp/upload.txt` to be present on file system
- file `/etc/passwd` to be not empty and readable
8.2.4.4 Diagnostic

In order to diagnose why a certain request is kept on hold, the `describe` function may be used.

Furthermore, the `force` command may be issued to reset all the events requested by a job, and force it to run.
8.2.5 Chains

In order to understand chain behaviour some concepts must be explained. The items composing a chain are:

- Chain
- Steps
- Jobs

Once run each item may assume two states:

- (normally) terminated
- fault

When an item ends in fault state NO return code is available. Instead, when an item ends "normally" a return code is provided.

Chains and steps have a set of actions to be performed on fault or normal end.
8.2.5.1 Chains

A chain is a sequence of steps. When running a chain, XBM executes all the defined steps in the defined order.

A chain is identified by:

- name
- sequence of steps
- default set of step start conditions (see below)
- default set of step fault rules (see below)
- default set of step termination actions (see below)
- default set of step fault actions (see below)
- set of fault rules
- set of fault actions
- set of termination actions
- return code calculation mode

Defined steps must have different names.

Fault rules

Fault rules are a set of conditions tested to determine if a step is fault or normally terminated. These conditions are tested in order the have been supplied.

If one of supplied rules is verified, chain is considered fault.

See below for rule types.

Fault action

Fault actions are a set of action to be performed when chain faults. All the defined actions are executed.

See below for action types.

Termination action

Termination actions are a set of action to be performed when chain terminates normally. All the defined actions are executed.

See below for action types.

Return code calculation

Chain return code may be calculated using:

- minimum step return code caught
- average step return code caught
- maximum step return code caught
- return code of last step executed
8.2.5.2 Steps

A Step is a set of jobs. When running a step, XBM submits all the defined jobs at the same time.

A Step is identified by:

- name
- set of Jobs
- set of start conditions
- set of fault rules
- set of fault actions
- set of termination actions
- return code calculation mode

Steps must not be empty.

Start conditions

Start conditions are a set of conditions tested to determine if a step may be run. These conditions are tested in order the have been supplied.

If no rule is supplied, chain default step fault rules are used.
If one of supplied rules is verified, step is run.
See below for rule types.

Fault rules

Fault rules are a set of conditions tested to determine if a step is fault or normally terminated. These conditions are tested in order the have been supplied.

If no rule is supplied, chain default step fault rules are used.
If one of supplied rules is verified, step is considered fault.
See below for rule types.

Fault action

Fault actions are a set of action to be performed when step faults. All the defined actions are executed.

If no action is supplied, chain default step fault actions are used.
See below for action types.

Termination action

Termination actions are a set of action to be performed when step terminates normally. All the defined actions are executed.

If no action is supplied, chain default step fault actions are used.
See below for action types.

Return code calculation

Step return code may be calculated using:

- minimum job return code caught
• average job return code caught
• maximum job return code caught
• return code of last job terminated
8.2.5.3 Jobs

A Job is the single unit of work, and it is identified by:

- command line
- priority
- engine
- session name

If no session is specified, the job runs in the session running chain. When a session is specified, XBM tries to start it on the specified remote session.

Fault state

Job faults when XBM cannot start it. Possible fault cause may be:

- Job non found in JOBPATH
- Remote session not defined in datasource
- Remote session is not active
- Remote session cannot found specified Job in its JOBPATH
8.2.5.4 Rules

A rule is an entity tested to determine when a certain condition is verified.

Rules are Java objects subclassed from the abstract class com.hite.xbm.chains.Rule and user may define its own rules (see plugin chapter).

Following rules are supplied by default with XBM:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Verified if</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.hite.xbm.chains.RuleTrue</td>
<td>Always</td>
<td></td>
</tr>
<tr>
<td>com.hite.xbm.chains.RuleFalse</td>
<td>Never (causes rules checking to stop)</td>
<td></td>
</tr>
<tr>
<td>com.hite.xbm.chains.RuleIsFault</td>
<td>an item faults</td>
<td>item name(1)</td>
</tr>
<tr>
<td>com.hite.xbm.chains.RuleIsNormallyTerminated</td>
<td>an item terminates normally</td>
<td>item name(1)</td>
</tr>
<tr>
<td>com.hite.xbm.chains.RuleReturnCode</td>
<td>an item RC matches condition</td>
<td>item name(1) condition (&lt;, &gt;, =, !=, IN) value</td>
</tr>
</tbody>
</table>

(1) if "*" is specified any item matches selection.

IN condition allows to check more than one possible value, simply specifying one or more values delimited by "," (comma) or ranges, delimited by "-" (minus).

i.e.

| IN 3,8 | return true if RC = 3 or RC = 8 |
| IN 3-8 | return true if RC >= 3 and RC <=8 |
| IN 5,8-12,20 | return true if RC = 5 or (RC>=8 and RC<=12) or RC=20 |
### 8.2.5.5 Actions

An action is an entity run on item termination (normal or fault).

Actions are Java objects subclassed from the abstract class `com.hite.xbm.chains.Action` and user may define its own actions (see plugin chapter).

Following actions are supplied by default with XBM:

<table>
<thead>
<tr>
<th>Class name</th>
<th>Action</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.hite.xbm.chains.EMailAction</code></td>
<td>Sends an email message to specified recipient</td>
<td>to subject message</td>
</tr>
<tr>
<td><code>com.hite.xbm.chains.GoToStepAction</code></td>
<td>Causes chain to skip forward to specified step</td>
<td>item name</td>
</tr>
<tr>
<td><code>com.hite.xbm.chains.FaultChainAction</code></td>
<td>Causes chain to stop execution, terminating chain with fault</td>
<td></td>
</tr>
<tr>
<td><code>com.hite.xbm.chains.StopChainAction</code></td>
<td>Causes chain to stop execution, terminating chain normally</td>
<td>return code</td>
</tr>
<tr>
<td><code>com.hite.xbm.chains.RunChainAction</code></td>
<td>Submit a chain execution to specified session</td>
<td>chain name session name</td>
</tr>
<tr>
<td><code>com.hite.xbm.chains.RunJclAction</code></td>
<td>Submit a jcl execution to specified session</td>
<td>jcl args session name engine priority</td>
</tr>
</tbody>
</table>
8.3 **Installation**

In order to setup XBM, the following components must are required on the server side:

- Java 2 Standard edition 1.4.0 (JRE or JDK 1.4)
- RDBMS
- a working FTP daemon

and on the client side (where remote consoles run):

- Java 2 Standard edition 1.4.0 (JRE or JDK 1.4) or higher

**Licensing**

Since version 3.4 XBM is protected by a license. The XBM is contained in a text file, that can be located wherever on the host system (normally under $XBMHOME).

Each XBM session must include in its configuration file (xbm.conf) a parameter named license.file which paths to the absolute name of the license file.

I.e.

```
license.file=/home/xbm/license.dat
```

To obtaining the license file for your system(s), please contact High Technology, supplying, for each system where you want XBM to run, the output of the command hostname.

**Running on Java 2 version 1.3.x**

On some systems, like AIX 4.x, Java 1.4 is not available. In order to have XBM running on these systems user must include the following like in the profile of userid running xbmd:

```
CLASSPATH=$XBMHOME/lib/classes12.jar:$CLASSPATH
```

However some functionalities may not be supported on Java version 1.3.x.
8.3.1 Installation and upgrade

To install or upgrade XBM on a UNIX/Linux system:

1. Copy the XBM distribution file (xbm-version.tgz) from the distribution media (cd or HTWC website) in temporary directory
2. unzip the file using "gzip -d xbm-version.tgz"
3. uncompress the tarball using "tar xvf xbm-version.tar"
4. run "sh install.sh" and follow the instructions.

Once the procedure is completed, XBM is setup

**Database setup**

XBM uses a relational database to store its data. The first time XBM is installed, a database schema must be created and setup.

The database schema must be created using the native utilities of the RDBMS itself (i.e. sqlplus).

These are some samples to initialize the schema, for different database types:

**Oracle**

```sql
create tablespace "XBM" logging
datafile '/some/where/xbm.dbf'
size 1024m
extent management local
segment space management auto;
create user xbm
identified by "xbm"
default database "XBM"
profile default
account unlock;
grant connect to xbm;
grant resource to xbm;
grant unlimited tablespace to xbm;
alter user xbm default role all;
```

**DB2**

```sql
create schema xbm authorization xbm;
```

**MySQL**

```sql
create database if not exists xbm
grant all privileges on xbm.* to 'xbm@localhost' identified by 'xbm';
grant all privileges on xbm.* to 'xbm@%' identified by 'xbm';
```

Once the database schema is defined, the XBM tables must be created.

Tables creation may be performed automatically by means of XBM Datasource Manager (refer to the XBM Datasource Manager section), or manually using the scripts contained in $XBMBHOME/scripts.

When tables have been created, XBM sessions and users must be defined: XBM Datasource Manager can be used to achieve this.
Shell environment setup

To make use of XBM commands, the following instructions must be added to the environment definition (.profile):

```
export XBMHOME=<xbm setup directory>
export PATH=$XBMHOME/bin:$PATH
export XBMOPID=<default operator definition>
```

The XBMOPID environment describes the default connection for "xbmc" utility, and it is described in the "Command line interface" section.

Upgrading XBM database to version 4

In order to upgrade an existing XBM installation to version 4, XBM database must be modified.

To do this, follow these steps:

- shut down running instances of XBM,
- run the script `$XBMHOME/scripts/upgrade.3.to.4.oracle.sql`:
  ```
  # sqlplus xbmuser/password @$XBMHOME/scripts/upgrade.3.to.4.oracle.sql
  ```
- install XBM version 4
- restart XBM
### 8.4 Configuration

XBM is configured using an ASCII text file called xbm.conf. This file is a standard Java property file: every entry in the file is a pair item/value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Defaults (if any)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session.name</td>
<td>default</td>
<td>session name</td>
</tr>
<tr>
<td>license.file</td>
<td>&lt;none&gt;</td>
<td>absolute path to XBM license file</td>
</tr>
<tr>
<td>core.engines</td>
<td>4</td>
<td>number of defined engines</td>
</tr>
<tr>
<td>core.name.locks</td>
<td>1</td>
<td>if 1 execution of the same JCL (with the same name) are serialized</td>
</tr>
<tr>
<td>core.async.parsing</td>
<td>0</td>
<td>if 1 async JCL parsing is enabled</td>
</tr>
<tr>
<td>core.jobqueue.maxlength</td>
<td>0</td>
<td>if &gt; 0, XBM notifies on console each time the number of pending jobs exceeds the specified value</td>
</tr>
<tr>
<td>core.run.next.on.timer</td>
<td>false</td>
<td>if set to true XBM checks for the stand by queue on each timer request.</td>
</tr>
<tr>
<td>core.remove.tmp.script</td>
<td>true</td>
<td>if set to false, scripts temporary files are not deleted</td>
</tr>
<tr>
<td>super.user.mode</td>
<td>false</td>
<td>set to true if XBM is run as root user</td>
</tr>
<tr>
<td>net.console.timeout</td>
<td>3000</td>
<td>timeout for remote consoles sockets</td>
</tr>
<tr>
<td>net.port</td>
<td>8200</td>
<td>TCP/IP port for the netserver</td>
</tr>
<tr>
<td>core.tmp.path</td>
<td>$HOME/tmp</td>
<td>path for temporary files</td>
</tr>
<tr>
<td>core.log.path</td>
<td>$HOME/logs</td>
<td>path for logs</td>
</tr>
<tr>
<td>core.chain.path</td>
<td>$HOME/chains</td>
<td>path for chains</td>
</tr>
<tr>
<td>core.timer.interval</td>
<td>300</td>
<td>interval in seconds for the timer</td>
</tr>
<tr>
<td>core.auto.gc</td>
<td>0</td>
<td>if 1 garbage collector is managed by XBM</td>
</tr>
<tr>
<td>core.sbq.backup.path</td>
<td>$HOME</td>
<td>path for stand by queue mirrors and backup/restore</td>
</tr>
<tr>
<td>core.service.port</td>
<td>8999</td>
<td>port number for special service entry</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>core.trace.level</td>
<td>0</td>
<td>level for tracing (0/1/2)</td>
</tr>
<tr>
<td>user.timezone</td>
<td>&lt;none&gt;</td>
<td>overrides the JVM user.timezone parameters</td>
</tr>
<tr>
<td>script.use.jmvs</td>
<td>0</td>
<td>if 1 uses JVMS interface</td>
</tr>
<tr>
<td>script.libpath</td>
<td>$PWD</td>
<td>multiple paths for JCLs (column &quot;*&quot; separated)</td>
</tr>
<tr>
<td>script.procpath</td>
<td>&lt;none&gt;</td>
<td>path for PROCs</td>
</tr>
<tr>
<td>script.syslst</td>
<td>&lt;none&gt;</td>
<td>path for SYSLSTs</td>
</tr>
<tr>
<td>script.join.stdout.stderr</td>
<td>false</td>
<td>if true script's STDOUT and STDERR are joined together via 2&gt;&amp;1</td>
</tr>
<tr>
<td>jdbc.driver</td>
<td>&lt;none&gt;</td>
<td>class name of the JDBC driver</td>
</tr>
<tr>
<td>jdbc.url</td>
<td>&lt;none&gt;</td>
<td>URL of database connection</td>
</tr>
<tr>
<td>jdbc.user</td>
<td>&lt;none&gt;</td>
<td>user for RDBMS</td>
</tr>
<tr>
<td>jdbc.password</td>
<td>&lt;none&gt;</td>
<td>password for RDBMS</td>
</tr>
<tr>
<td>mail.host</td>
<td>localhost</td>
<td>SMTP host (mail server)</td>
</tr>
<tr>
<td>mail.from</td>
<td>xbm</td>
<td>xbm email address</td>
</tr>
<tr>
<td>remote.jdbc.driver</td>
<td>&lt;jdbc.driver&gt;</td>
<td>The JDBC driver used by remote consoles</td>
</tr>
<tr>
<td>remote.jdbc.url</td>
<td>&lt;jdbc.url&gt;</td>
<td>The JDBC database URL used by remote consoles</td>
</tr>
<tr>
<td>remote.jdbc.user</td>
<td>&lt;jdbc.user&gt;</td>
<td>The JDBC database users used by remote consoles</td>
</tr>
<tr>
<td>remote.ftp.user</td>
<td>xbm</td>
<td>The FTP users used by remote consoles</td>
</tr>
<tr>
<td>remote.ftp.password</td>
<td>xbm</td>
<td>The FTP password used by remote consoles</td>
</tr>
<tr>
<td>rmijdbc.server.enabled</td>
<td>false</td>
<td>If true XBM starts an RMI-JDBC bridge</td>
</tr>
<tr>
<td>rmijdbc.server.port</td>
<td>0</td>
<td>If &gt; 0 the RMI-JDBC bridge works on specified port</td>
</tr>
<tr>
<td>rmijdbc.server.external.registry</td>
<td>false</td>
<td>If true XBM starts an RMI-JDBC bridge</td>
</tr>
<tr>
<td>remote.jdbc.password</td>
<td>&lt;jdbc.password&gt;</td>
<td>The JDBC database password used by remote console</td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>admin.address</td>
<td>root</td>
<td>root email address</td>
</tr>
<tr>
<td>enginexx.nice</td>
<td>0</td>
<td>nice level for engine xx</td>
</tr>
<tr>
<td>enginexx.alias</td>
<td>&lt;none&gt;</td>
<td>alias name for engine xx</td>
</tr>
<tr>
<td>enginexx.enabled</td>
<td>true</td>
<td>status for engine xx</td>
</tr>
<tr>
<td>enginexx.ondemand</td>
<td>false</td>
<td>if &quot;true&quot; engine is on demand</td>
</tr>
<tr>
<td>enginexx.reservedto</td>
<td>&lt;none&gt;</td>
<td>if set, only specified operator can use the engine</td>
</tr>
<tr>
<td>admin.only.chain.editing</td>
<td>false</td>
<td>if &quot;true&quot; only administrator users may edit chains</td>
</tr>
<tr>
<td>ftp.navigation.restrictions</td>
<td>none</td>
<td>if set, non-administrator users may perform only specified operations via the FTP browser. Possible operations are: navigation, submit and view</td>
</tr>
</tbody>
</table>
8.4.1 A sample configuration

```
session.name=TESTSESSION
core.engines=4
core.name.locks=1
core.async.parsing=0
core.tmp.path=/var/tmp
core.log.path=/var/logs
core.trace.level=0
core.use.timer=1
core.service.port=6998
licences.file=/home/xbm/licenses.dat
net.port=8200
script.use.jme=0
script.library=/home/porting/jclsls/users/local/jcls
script.procpath=/home/porting/jclsls/procs
script.syslist=/usr/porting/spcl/syslist
jdbc.driver=oracle.jdbc.driver.OracleDriver
jdbc.url=jdbc:oracle:thin:@hpux01:1521:HITORAL
jdbc.user=xbm
jdbc.password=xbm
remote.jdbc.driver=oracle.jdbc.driver.OracleDriver
remote.jdbc.url=jdbc:oracle:thin:@hpux01:1521:HITORAL
remote.jdbc.user=xbm
remote.jdbc.password=xbm
remote.ftp.user=user1
remote.ftp.password=clisja
admin.address=calabretta@hite.it
mail.host=mail.ht.net
mail.from=xbm@hpux01.ht.net
ftp.navigation.restrictions=navigation,submit
engine00.nice=0
engine01.nice=0
engine02.nice=10
engine03.nice=10
engine00.enabled=true
engine01.enabled=true
engine02.enabled=false
engine03.enabled=false
engine00.alias=fasttp
engine01.alias=fasttp
engine02.alias=slothtp
engine03.alias=slothtp
engine03.ondemand=true
engine02.reservedto=cics01
```
### 8.5 Starting and stopping

XBM daemon can be started up using the command "xbmd" from the command line of the user who will hold the XBM resources.

If properly configured, more than one instances of XBM can be run on a single system.

#### Usage

```
xmd [options] [command]
```

#### Options

- `-d` java vm debug mode
- `-f` run in foreground
- `-n` do not perform session recovery
- `-c<file>` xbm start with specified configuration file
- `-y` forces YES reply in shutdown
- `-i` immediate shutdown

#### Command

- `start` starts XBM daemon (default)
- `stop` stops XBM daemon
- `dbm` start XBM Datasync Manager

If the foreground mode is specified on command line, XBM daemon starts as a foreground process. If no parameter is specified, XBM daemon works in the background.

#### Shutdown

Only administrator users can shutdown XBM daemon. Shutdown can be requested using command line or GUI console.

From command line shutdown may invoke issuing:

```
xbmc shutdown
```

or

```
xbmd stop
```

When shutdown is invoked, XBM stops accepting further JCL submission, and waits for all scheduled requests to be completed.

If the immediate shutdown is requested, XBM backups the stand by queue and kills all running jobs.
8.6 Command line interface

XBM operations can be controlled using system command line using xbmc.

Configuration

xbmc connects it XBM using identifying the operator with settings supplied by the environment XBMOPID. This environment is set as follow:

```
XBMOPID=user/password@hostname[:port]
```

Usage

```
xbmc [options] command <args>
```

Global options

```
-O<opid> overrides XBMOPID settings.
```

Commands

There are many commands to control different aspects of XBM and each command has its own options. These commands may be grouped as follows and they are documented in the following pages.

Job control:

- submit
- move
- pause
- resume
- change
- force
- purge
- kill

Chain control:

- chain run
- chain kill

Event control:

- event set
- event reset
- event unset

Report and inquiry:

- report
- query
- describe
XBM administration:

- shutdown
- disable engine / enable engine
- disable submit / enable submit
- service
- sbq store / sbq restore
8.6.1 chain kill

Kills the specified chain request id

**Usage**

```
xbmc chain kill <reqid>
```

**Options**

```
none
```
8.6.2 chain run

Submits the specified chain

**Usage**

```
xbmc chain run <chain>
```

**Options**

```
-s<step>  starts chain from specified step
```
8.6.3 change

Changes submission information for the specified request in stand by queue. Both requested engine and priority may be changed. Issuing change without options priority is reset to default and engine requested is set to "any".

Usage

```
xbmc [otions] change <reqid|all [filter]>
```

Options

- `-p<priority>` specifies priority
- `-e<engine>` requests a specific engine (number or alias)

Filters

<table>
<thead>
<tr>
<th>Filter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner=&lt;ownername&gt;</td>
<td>filters for the specified ownername</td>
</tr>
<tr>
<td>engine=&lt;engine alias&gt;</td>
<td>filters for the specified engine alias</td>
</tr>
<tr>
<td>jobname=&lt;jobname regexp&gt;</td>
<td>filters jobname against the specified regular expression</td>
</tr>
</tbody>
</table>
8.6.4 describe

Describes the specified request providing all information available.

Usage

xbmc describe <reqid>

Options

none
8.6.5 disable engine

Disables specified engine. If all is specified, all engines are disabled.

**Usage**

```
xbmc disable engine <n|alias|all>
```

**Options**

```
none
```
8.6.6 disable submit
Disables JCL submission.

Usage
xbmc enable submit

Options
none
8.6.7 enable engine

Enables specified engine. If all is specified, all engines are enabled.

Usage

```
xbmc enable engine <n|alias|all>
```

Options

- none
8.6.8 enable submit
Enables JCL submission.

Usage
xbmc enable submit

Options
none
8.6.9 event reset
Reset all event variables.

Usage
xbmc event reset

Options
none
8.6.10 event set

Sets an event variable.

**Usage**

xbmc event set <name> <value>

**Options**

none
8.6.11 event unset

Unsets an event variable.

Usage

xbmc event unset <name>

Options

none
8.6.12 force

Forces the execution of the specified request in the stand by queue, removing all the events eventually waited.

**Usage**

xbmc force <reqid>

**Options**

none
8.6.13 move

Move the specified request at the specified position. If requested position is greater than size of queue, request is moved at last position.

Usage

xbmc move <reqid> <position>

Options

none
8.6.14 kill

Kills the specified request running, sending SIGKILL to all subprocesses started by the request.

Usage

```
xbmc kill <reqid>
```

Options

- none
8.6.15 log

Logs specified message on the xbmc central console.

Usage

xbmc log <message>

Options

-l<level> specifies the message level

0 information
1 warning
2 alert
3 urgent
8.6.16pause

Pauses the specified request in stand by queue. If all is specified all requestes are paused.

Usage
xbmc pause <reqid|all [filter]>

Options

none

Filters

| owner=<ownername>     | filters for the specified owenname  |
| engine=<engine alias> | filters for the specified engine alias |
| jobname=<jobname regexp> | filters jobname against the specified regular expression |
8.6.17 purge

Purges the specified request from stand by queue. If all is specified all requestes are purged.

Usage

xbmc purge <reqid|all [filter]>

Options

none

Filters

<table>
<thead>
<tr>
<th>owner=&lt;ownername&gt;</th>
<th>filters for the specified ownername</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine=&lt;engine alias&gt;</td>
<td>filters for the specified engine alias</td>
</tr>
<tr>
<td>jobname=&lt;jobname regexp&gt;</td>
<td>filters jobname against the specified regular expression</td>
</tr>
</tbody>
</table>
8.6.18 query

Reports about the execution of a chain or a job.

Range Values:

- last reports the last specified item run
- after <date> reports the first item run after the date provided
- before <date> reports the last item run before the date provided

Usage

xbmc query (job|chain) {<id>} <range>

Parameters

{id} is the name of the jcl or chain request id
{range} is the selection value, that may be:
  - last
  - after <date>
  - before <date>
{date} must be specified in the following format: yyyy.MM.dd.hh.mm.ss
8.6.19 reply
Replies the specified request if waiting for operator input.

Usage
xbmc reply <reqid> <response>

Options
none
8.6.20 report

Produces different kind of reports about the status of XBM. If no report type is specified a "full" report is produced.

Report types:
- full
- jobqueue
- engines
- version
- parameters
- resources
- chains
- events
- pids

Usage
xbmc report [report_type]

Options
none
8.6.21 resume

Resumes the specified request in stand by queue. If all is specified all requestes are resumed.

Usage

xbmc resume <reqid|all [filter]>

Options

none

Filters

<table>
<thead>
<tr>
<th>owner=&lt;ownernam&gt;</th>
<th>filters for the specified ownernam</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine=&lt;engine alias&gt;</td>
<td>filters for the specified engine alias</td>
</tr>
<tr>
<td>jobname=&lt;jobname regexp&gt;</td>
<td>filters jobname against the specified regular expression</td>
</tr>
</tbody>
</table>
8.6.22 sbq restore

Restores stand by queue from a sequential file. The SBQ is restored from the file name xbm.session_name.sbq.backup.dat located in the directory defined core.sbq.backup.path in the configuration file. Please refer to the configuration section of this guide to get more information.

Usage

    xbmc sbq restore

Options

    none
8.6.23 sbq store

Stores stand by queue in a sequential file. The SBQ is stored from the file name `xbm.session_name.sbq.backup.dat` located in the directory defined `core.sbq.backup.path` in the configuration file. Please refer to the configuration section of this guide to get more information.

Usage

```
xbmc sbq store
```

Options

```
one
```
8.6.24 service
Access XBM internal services

**Usage**
```
xmbc service port <command>
```

**Parameters**
- `trace none` set tracing to none
- `trace log` set tracing to log level
- `trace debug` set tracing to debug level
- `dump` dumps diagnostic information on file
8.6.25 shutdown
Shuts down XBM.

Usage
xbmc shutdown

Options
- i  immediate shutdown
- y  suppress confirmation
8.6.26 submit

Submits the specified JCL for execution with optionally specified parms. If an absolute pathname is given for the job no JOBPATH search is done by XBM, otherwise specified job is searched in the specified JOBPATH.

Usage

xbmc submit <job> [args]

Options

- `p<priority>` specifies priority
- `e<engine>` requests a specific engine (number or alias)
- `-w` wait mode: xbmc waits for JCL terminination and exits
- `-m` runs the JCL even if another JCL with the same name is already running
- `-h` submits the JCL in pause status (hold)
- `-E<environment>` submits the JCL, passing one or more environment definitions as follows:

  NAME=value

  I.e.

  xbmc submit -E"VAR1=XX" -E"VAR2='A B'"

- `-t<time>` specifies when to submit specified job. `<time>` must be specified in the following format:

  yyyy.MM dd.hh.mm

  Time units separator may be a '.' (dot), a ':' (column) or a '/' (slash).

  If a smaller number of elements is provided xbm will assume the closest time value.

  I.e.

  xbmc -t 2002.1.10.10.10 submit JOB1.csh

  will submit JOB1.csh on January, 10th 2002 at 10:10 AM.

  xbmc -t 10:10 submit JOB1.csh

  will submit JOB1.csh at 10:10 AM of current day. If current time is

  after 10:10 the script will be submitted on the next day.

  xbmc -t 27.10.10 submit JOB1.csh

  will submit JOB1.csh at 10:10 AM of 27th of current
month.
8.7 XBM Remote Console

In order to provide an easy access to XBM resources a GUI based remote console has been provided. The remote console is pure Java 2 coding, therefore it can be run on each system supporting the Java 2 platform.

To start XBM Remote Console just start xbmcw.cmd for Windows systems, or xbmcw for UNIX systems.

The menu bar
All the functions available in the remote console can be accessed through the menu bar.

The toolbar
The toolbar offers a shortcut for some of the most frequently used functions: a tooltip will explain the function for the selected icon.

The log window
The log windows shows the central console log, since connected. Each log entry is identified by

- level (Info, Warning, Alert, Urgent)
- date and time
- message text

Different colors identify different message level:

- magenta for information
- green for warnings
- yellow for alerts
- red for urgent

The log window may be cleaned up using the appropriate menu item.

The status window
The status window shows the current XBM status, and it divided in two areas:

- the engine status area
- the stand by queue

The engine status area show information about the status of the engines. For each engine the following information are shown:

- engine name (id)
- engine alias
- thread status (dead or alive)
- engine status (standby, busy or waiting)
- request assigned
- PID
- job name
- current step
- job owner
- job startup time

In the standby queue area the pending requests are shown:
- progressive number
- request ID
- job name
- status (paused or waiting)
- priority
- owner
- requested engine

By selecting a row and clicking right mouse button a popup menu is brought up offering the available choices for the selected resource.

**The active chains window**

The active chains window shows the chains running at the moment. By right clicking mouse a monitor may be activated.

**The connect dialog**

The connect dialog shows the session defined in the XBM database. Selecting the desired session, all the setting for that session are loaded from DB.

Supplying user name and password, and clicking Ok, the remote console connects the XBM session.

**The direct connection dialog**

By using this dialog, it is possible to connect an XBM session, specifying all the session parameters. Clicking Ok, remote console tries to connect the specified session, using the parameter supplied.

**The options dialog**

Through this dialog it is possible to configure the XBM remote console. Configurable parameter are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default user</td>
<td>User prompted by default in the connect dialog</td>
</tr>
<tr>
<td>External editor</td>
<td>Path to the external editor (if any)</td>
</tr>
</tbody>
</table>
Print Settings | Specifies some parameter to setup printings

The options set will be activate the next time remote console starts.

**The submit dialog**

Through this dialog it is possible to start JOBs, specifying the eventually required engine, priority level, hold and/or parallel mode.

**The broadcast message dialog**

Using this dialog, it is possible to send a message on the central console log, also specifying the message level.

**The SYSLSTs form**

The SYSLSTs window shows the SYSLSTs (JCL output) available.

- If Step details is clicked the Step Details form is shown.
- By clicking Delete the selected SYSLST is deleted.
- By clicking Refresh, the SYSLSTs content is updated.
- Selecting one item and clicking Show, the SYSLST is shown in another dialog.
- By clicking Filter, the filters dialog appear, in order to define filters (range, name, owner, etc.) and sort order on the items shown in the SYSLSTs form.
- Using the Show script button, the source of run script may be visualized. This option works only if XBM has been configured to retain temporary files.

**The step detail form**

This form shows all return code for each step of a Job execution. A filter allow the selection of non zero return code steps.

**The scheduler form**

The scheduler window shows available tasks.

- By using Add, Remove and Edit buttons tasks may be defined, edited and removed.
- Using Enable/Disable buttons a task may be enabled or disabled.
- By clicking Refresh, the Scheduler content is updated.

**The calendar form**

This form shows the available calendars.

- By using Add, Remove and Edit buttons calendars may be defined, edited and removed.

**The chains form**

This form shows the available chains.
By using Add, Remove and Edit buttons chains may be defined, edited and removed.
By using Execute chains may be started, eventually from the selected step.
By using Copy chains may be copied into a new one.
By using Rename chains may be renamed.
By using Print chains may be printed.

**The chain editor form**

This form allows user to edit chains.

**The chain log form**

The chain log window shows the result of chains available.

If Details is clicked the Chain Details form is shown.
By clicking Delete the selected chain log is deleted.
By clicking Refresh, the chain logs content is updated.
By clicking Filter, the filters dialog appear, in order to define filters (range, name, owner, etc.) and sort order on the items shown in the chain logs form.

**The report form**

This form shows installed pluggable report. Selecting a report and running it, eventually required user input is prompted, and the report is executed. A new window will show the result of the query, in a table format.

**The FTP browsing form**

This windows enables the browsing of the remote site, using the FTP protocol. XBM administrators may define the level of interaction that the user may have with the remote site.

By default standard user may freely navigate through directories, view files and submit selected scripts. Administrators may reduce their privileges, limiting these features.

See the ftp.navigation.restrictions options for the XBM configuration.
8.7.1 Session definition

The session represents the set of information to locate and connected an XBM instance. Once defined, the session may be used at any time, together with username and password, to connect the console to the desired instance.

The session must contain:

- a mnemonic name
- hostname or IP address of the machine running the XBM instance
- the port number where XBM is listening for console connection

**SSH Tunnel**

Sometimes, due to network configuration or for security reasons, it may be necessary to connect the XBM instance using an SSH tunnel. In this situation, all the XBM communication streams are tunnelled into an SSH connection.

Therefore it is necessary to have a valid SSH connection to a system that can reach the XBM instance and database: normally this system is the same one that runs XBM, but also different systems may be used as “bridge”.

To enable the tunnelling, the SSH tunnel check-box must be flagged and hostname, port and user name for the SSH connection must be defined.

XBM supports SSH2 protocol with password and certificate authentication methods. At least one authentication method must be selected.

Note: If you use the password authentication mechanism and you want to store the SSH password in the session definition, remember that XBM will store that password in a clear text file. If the password is not stored, XBM will ask for it at each connection.

Please note that the XBM user and SSH use are two different entities: the first one is defined in the XBM database and it is known only to the XBM instances; the second one is defined to the system connect via SSH. Also the XBM hostname/port and SSH hostname/port are different entities: the first one is the location of the XBM instance (as seen by the SSH host) and the second is the location of the SSH service as seen by the local workstation.
8.8 XBM Datasource Manager

XBM DataSource Manager is a tool that allows:

- XBM tables creation
- session definition
- user maintenance
- user assignment
- logs cleaning

XBM datasource may be run stand-alone using the command "xbmd dbm" or in the same VM of XBM Remote Console using the appropriate menu item.
8.8.1 Usage

XBM Datasource Manager (DBM since now on) can manage more than one XBM database.

Each connection to a different datasource must be defined providing:

- name
- JDBC driver class
- JDBC url
- user
- password

For more information about JDBC, please refer to the "JDBC drivers and URLs" section.

Once the connection is defined, by clicking on the corresponding tree item, connection is activated and data contained in the database can be accessed.

**First setup**

As described in the "Installation and Upgrade" section, the first time XBM is setup, an empty database schema must be created.

After that, XBM DBM can be used to correctly create tables: if the connected database has not been yet initialized for XBM, the DBM will generate the tables and basic data.

Now, it is possible to define users and sessions.
8.9 **Scripting interface**

In order to take advantage of the features offered by XBM, JCL (scripts) must use the XBM JCL interface, which is a set of commands and tags provided to access the XBM resources.

**Static tags**

Static tags are provided to statically define and set some resource and property for a JCL. These settings can be detected by XBM at submission time, when XBM analyzes the script.

All XBM static tags are prefixed with the string 


**use_resource**

The use_resource tag statically defines a resource. When this resource is defined XBM understands the XBM will use it during execution.

**Usage**

```bash
#@xbm use_resource <resource_identifier>
```

**run_on_engine**

This tag request XBM to run the JCL on the specified engine.

**Usage**

```bash
#@xbm run_on_engine <engine_id|engine_alias>
```

**run_with_priority**

This tag request XBM to run the JCL with specified priority

**Usage**

```bash
#@xbm run_with_priority <priority>
```

**allow_multiple**

If this tag is specified more copy of the JCL can run concurrently.

**Usage**

```bash
#@xbm allow_multiple
```

**start_paused**

If this tag is specified JCL is set in pause mode (hold) when submitted.

**Usage**

```bash
#@xbm start_paused
```
**mail_to**

This tag asks to mail the specified address, when JCL completes.

**Usage**

```csh
#@xbm mail_to <email_address>
```

**wait event**

This tag asks requires XBM to wait for the specified event variable before to start the job. One or more condition may be specified on the tag according to this syntax:

```
<value> <operator> <value>
```

where operator is one of the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq</td>
<td>string equal</td>
</tr>
<tr>
<td>ne</td>
<td>string not equal</td>
</tr>
<tr>
<td>lt</td>
<td>string less than</td>
</tr>
<tr>
<td>le</td>
<td>string less equal</td>
</tr>
<tr>
<td>ge</td>
<td>string greater equal</td>
</tr>
<tr>
<td>gt</td>
<td>string greater than</td>
</tr>
<tr>
<td>=</td>
<td>numeric equal</td>
</tr>
<tr>
<td>!=</td>
<td>numeric not equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>numeric less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>numeric less equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>numeric greater equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>numeric greater</td>
</tr>
</tbody>
</table>

**Usage**

```csh
#@xbm wait event event_name [ condition [ (and|or) condition ] ]
```

**Example**

```csh
#!/bin/csh
#@xbm wait event VAR1
#@xbm wait event VAR2 value = 0 or value = 10
#@xbm wait event VAR3 value eq "OK" or value ne "KO"
```
**wait job**

This tag asks requires XBM to submit the job depending on the execution of another job (parent). XBM will check if the parent has already been submitted and executed within the time range specified and, if it is not yet started, XBM will wait for the parent execution, before to activate the job. The execution of the job may be also conditioned by the return code of the parent job.

The time range may be one of the following:

<table>
<thead>
<tr>
<th>Submit</th>
<th>Time range starts from the job submission time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today</td>
<td>Time range starts from the 0:00 AM of the current day</td>
</tr>
<tr>
<td>Reset</td>
<td>Time range starts from the last issue of xbmc reset command</td>
</tr>
<tr>
<td>Hh:mm</td>
<td>Time range specified hour</td>
</tr>
</tbody>
</table>

Conditions are defined as follows:

\[
[ rc | max | min ] <operator> <value>
\]

where operator is one of the following:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Numeric equal</td>
</tr>
<tr>
<td>!=</td>
<td>Numeric not equal</td>
</tr>
<tr>
<td>&lt;</td>
<td>Numeric less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Numeric less equal</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Numeric greater equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Numeric greater</td>
</tr>
</tbody>
</table>

**Usage**

```csh
@xbm wait job job_name since time_range [ condition [ (and|or) condition ] ]
```

**Example**

```csh
#!/bin/csh
@xbm wait job JOB01.csh since reset
@xbm wait job JOB02.csh since today rc = 0
@xbm wait job JOB03.csh since 15:30 max <= 12 and min >=4
```

**wait file**

This tag asks requires XBM to wait for the presence of the specified file before to start the job. The status of the file may be tested with these modifiers:
notempty  file length must be greater than zero
read     file must be readable
write    file must be writable

Usage

```bash
#@xbm wait file path_name [ modifier [ modifier ] ]
```

Example

```bash
#!/bin/csh
#
#@xbm wait file /var/tmp/upload.txt
#@xbm wait file /etc/passwd notempty readable
#```
8.9.1 Commands

A set of command and alias is provided to grant access to XBM features from JCLs. All the scripts using XBM must include the xbmlib.

Korn/Bourne shell scripts must include:

```
. $XBHOME/lib/xbmlib.sh
```

C shell scripts must include:

```
source $XBMHOME/lib/xbmlic.csh
```

**xbminit**

Initializes the data exchange to XBM. It must be issued at the beginning of each script using XBM

**Usage**

```
xbminit
```

**xbmstep**

Defines a new step and communicates XBM that the new step is running

**Usage**

```
xbmstep <step_name>
```

**xbmdisplay**

Displays text on XBM console log.

**Usage**

```
xbmdisplay <text>
```

**xbmdr**

Defines a logical resource.

**Usage**

```
xbmdr <resource_id>
```

**xbmur**

Undefines a logical resource

**Usage**

```
xbmur <resource_id>
```
**xbmlock**

Tries to locks the resources dynamically currently defined using xbmdr and xbmur. It holds the script execution until all the defined resources are available.

**Usage**

xbmlock

**xbmset**

Sets a Job variable and store the value in an environment variable.

**Usage**

xbmset <variable_name>=<variable_value>

**xbmget**

Gets the value for a Job variable and stores in an environment variable.

**Usage**

xbmget <variable_name>

**xbmaccept**

Asks operator input for the specified environment variable.

**Usage**

xbmaccept <variable_name>

**xbmrc**

Communicates the step return code to XBM.

**Usage**

xbmrc
### 8.9.2 Environment variables

XBM defines the following environment variables during JCL execution:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XBM_REQID</td>
<td>Request ID of running job</td>
</tr>
<tr>
<td>XBM_USER</td>
<td>Unix user alias</td>
</tr>
<tr>
<td>XBM_OPERATOR XBM</td>
<td>Unix user alias</td>
</tr>
<tr>
<td>XBM_JOBNAME</td>
<td>Name of current job (just the first token until the first &quot;.&quot;)</td>
</tr>
<tr>
<td>XBM_FULL_JOBNAME</td>
<td>Full name of the job</td>
</tr>
<tr>
<td>XBM_ENGINE</td>
<td>Current engine name</td>
</tr>
<tr>
<td>XBM_ENGINE_ALIAS</td>
<td>Current engine alias (if defined)</td>
</tr>
<tr>
<td>XBM_NICELEVEL</td>
<td>Nice level of running job</td>
</tr>
<tr>
<td>XBM_SESSION_ID</td>
<td>Identifier of the current XBM session</td>
</tr>
<tr>
<td>XBM_SESSION_NAME</td>
<td>Name of the current XBM session</td>
</tr>
<tr>
<td>XBM_SYSLST</td>
<td>Full path to the current SYSLST output</td>
</tr>
</tbody>
</table>
### 8.9.3 An example

```bash
#/bin/csh
#
# example_joh_csh
#
# static tags
#
# @xvm use_resource ANAGSET
# @xvm use_resource FILEANAG2
# @xvm run_on_engine 1
#
source /XFRAME/lib/xmlih.csh

xminit

source $HOME/user/include.csh

STEP1:
xmstep STEP1
xmsset MPID $$
xmsget VAR2
xmdisplay "The first step is running with PID $MPID"
xmdisplay "VAR2 has been set to $VAR2 by someone else"
xmdr $HOME/files/FILE1MPL.txt
xmdr $HOME/files/FGH50L.dat
xmlock
userpgm01
xmcc

STEP2:
xmstep STEP2
xmdisplay "Please enter value for lock mode"
xmaccept LOCK
xmrz $HOME/files/FILE1MPL.txt
xmr $HOME/files/FGH50L.dat
if ( "$LOCK" = "YES" ) then
    xmdr $HOME/files/OTHERL.txt
    xmdr $HOME/files/OTHER2.dat
endif
xmlock
xrun USERPGM02 <<end_of_cards
0010** 0099**
end_of_cards
xmcc

END
exit 0
```
8.10 APIs interface

XBM provides some of the JCL interface features to user programs by means of its C interface library: libxbm. This library must be linked with or dynamically loaded by those programs which are going to use the XBM C APIs.

One or two libraries are provided for each operating system/architecture: libxbm (32bit library) and libxbm64 (64bit library). These libraries are located in:

$XBMMHOME/lib/<os>/<arch>

Where <os> is the operating system and <arch> is the cpu architecture.

I.e, to link on an IA32 Linux system:

```
c -o mypgm mypgm c -L $XBMMHOME/lib/linux/ia32 -lxbm
```

I.e, to dynamically load the library in the XFRAME environment and use it from a COBOL program on an AIX system:

```
setenv XRUN_LIBRARIES "$XBMMHOME/lib/aix/ppc/libxbm.so"
xrun COBPGM
```
8.10.1 Programming Interface

All the APIs are written in C language, so, when calling from other languages, i.e. COBOL, programmer must take care about correct data type conversion.

xbm_display

This routine logs an information message on the XBM console. Upon successful completion, it returns 0 (zero) otherwise a non zero value is returned.

Prototype

```c
int xbm_display(char *message, int length);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>C type</th>
<th>COBOL type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>char *</td>
<td>PIC X</td>
<td>message to be logged</td>
</tr>
<tr>
<td>length</td>
<td>int</td>
<td>PIC S9(8) COMP-5</td>
<td>length of the message</td>
</tr>
</tbody>
</table>

COBOL call sample

```cobol
call "xbm_display" using by reference message                         
          by value     length of message
```

xbm_log

This routine logs a message with an user-defined level on the XBM console. Upon successful completion, it returns 0 (zero) otherwise a non zero value is returned.

Prototype

```c
int xbm_log(char *message, int length, int level);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>C type</th>
<th>COBOL type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>char *</td>
<td>PIC X</td>
<td>message to be logged</td>
</tr>
<tr>
<td>length</td>
<td>int</td>
<td>PIC S9(8) COMP-5</td>
<td>length of the message</td>
</tr>
</tbody>
</table>
| level     | int     | PIC S9(8) COMP-5    | level of the message:  
                          0 - information  
                          1 - warning       
                          2 - error        
                          3 - urgent
**COBOL call sample**

```cobol
01  level pic s9(4) comp-5

call "xbm_log" using by reference message                  
    by value       length of message                  
    by value       level
```

**xbm_accept**

This routine instructs XBM to ask for operator input and return the value supplied by the operator. Upon successful completion, it returns the length of the data received otherwise a negative value is returned.

**Prototype**

```cobol
int xbm_accept(char *message, int length, char *destination, int destlength);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>C type</th>
<th>COBOL type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>char *</td>
<td>PIC X(n)</td>
<td>message to be logged</td>
</tr>
<tr>
<td>length</td>
<td>int</td>
<td>PIC S9(8) COMP-5</td>
<td>length of the message</td>
</tr>
<tr>
<td>destination</td>
<td>char *</td>
<td>PIC X(n)</td>
<td>data-area in which XBM returns the value</td>
</tr>
<tr>
<td>destlength</td>
<td>int</td>
<td>PIC S9(8) COMP-5</td>
<td>length of data-area</td>
</tr>
</tbody>
</table>

**COBOL call sample**

```cobol
01  level pic s9(4) comp-5

call "xbm_accept" using by reference message                  
    by value       length of message                  
    by reference   data-area                        
    by value       length of data-area              
```
8.11 Plug-ins

A Plugin is a Java class that user may create to perform his own tasks. Actually 5 types of Plugin are handled:

- reports
- step.rule
- chain.rule
- step.action
- chain.action
8.11.1 Developing report plugins

Report plugins must be subclasses from com.hite.xbm.XBMReport and must define at least getSQLStatement(): This function must return the SQL query to obtain report.

User fields may be requested to operator by overriding addDataInput() and retrieving them by using getXXXXInput() methods.

I.e.

```java
package com.user;

public class UserReport extends XBMReport {
    public UserReport() {
        super();
    }

    public void addDataInput() {
        addStringInput("JCL", "Jcl name");
        addNumberInput("RC", "Return code");
    }

    public String getSQLStatement() {
        String jcl = getStringInput("JCL");
        int rc = getNumberInput("RC");
        String SQL = "select * from execution"
                     + " where jobname = "+jcl+" and return_code > "+rc;
        SQL += " order by started_at";
        return SQL;
    }
}
```
8.11.2 Developing Step.rule and Chain.rule plugins

Step and chain rules plugins must be subclasses from com.hite.xbm.chain.Rule and must define at least check(): This function must return true when rule is verified.

By using set() or get() functions addition parameters may be set.

I.e.

```java
package com.user;

import com.hite.xbm.*;
import com.hite.xbm.chains.*/

public class UserRule extends Rule { 
  public UserRule() { 
    super(); 
    set("RC", new Integer(99)); 
  } 
  
  public boolean check(ChainItem op) { 
    Integer myRc=Integer.get("RC"); 
    ListIterator li=op.getSubItem().listIterator(); 
    while (li.hasNext()) { 
      ChainItem i=(ChainItem) li.next(); 
      if (i.hasPar() && i.isNormallyTerminated()) { 
        if (i.getReturnCode()==myRc.intValue()) { 
          return true; 
        } 
      } 
    } 
    return false;
  }
}
```
Developing Step.action and Chain.action plugins

Step and chain action plugins must be subclasses from com.hite.xbm.chain.Action and must define at least execute():
This function must perform the desired tasks;

By using set() or get() functions addition parameters may be set.
I.e.

```java
package com.user;

import com.hite.xbm.*;
import com.hite.chains.*;
import java.util.*;

public class RunChainAction extends Action {
    public final static String NAME = "chain.name";
    public final static String SESSION = "chain.session";

    public RunChainAction() {
        setName("");
        setSession(null);
    }
    public void setName(String s) {
        set(NAME, s);
    }
    public String getName() {
        return ((String) get(NAME));
    }
    public void setSession(String s) {
        set(SESSION, s);
    }
    public String getSession() {
        return ((String) get(SESSION));
    }
    public void execute(ChainManager cm) {
        Chain c=new Chain();
        c.setName(getName());
        cm.log("action starts chain "+c.getName());
        cm.execute(c);
    }
}
```
8.11.4 Deploying plugins

In order to deploy plugins following operation must be performed:

1. A text file must be created containing the list of full class names and plugin types. I.E.

   com.user.UserReport=plugin
   com.user.RunChainAction=step.action, rule.action
   com.user.UserRule=step.rule
   com.user.UserRule2=step.rule, chain.rule

2. CLASSPATH must be set properly to path to defined plugins

3. The VM must be started using -Dplugins.properties=<URL> where URL paths to the file created before.
8.12 Using Java WebStart

XBM Remote Console is deployed in a format compatible with the Java WebStart Technology (tm).

With Java Web Start, user launches applications simply by clicking on a Web page link. If the application is not present on your computer, Java Web Start automatically downloads all necessary files. It then caches the files on local computer so the application is always ready to be re-launched anytime you want — either from an icon on your desktop or from the browser link.

Moreover if application files are out-of-date, Java Web Start automatically provides application updating by downloading newer files.

Requirements

The client machine requires support for the Java Runtime Environment (JRE), version 1.3 or later. Java Web Start (tm) is available for Windows 95/98/NT/2000/XP, Solaris (SPARC & Intel editions), and Linux.

Applications can be deployed from any standard Web server. In order to use Java Web Start (tm), the Web server must be configured with support for a new MIME type as explained below.
8.12.1 Installing on a J2EE Web Server

XBM is deployed with a J2EE Web Application that can be used to deploy the XBM Remote Console via Java Webstart, without having to configure anything. The only thing to do is to create a context in the J2EE WebServer hosting the web application located in $XBMHOME/webapp.

Installing on Tomcat 4.1

To install the deployment application on Tomcat 4.1 simply add in the Tomcat webapps directory ($CATALINA_HOME/webapps by default) an XML file named as the context you are going to use (i.e. xbm.xml) containing:

```xml
<Context path="<CONTEXT_NAME>" docBase="<XBMHOME>/webapp" debug="0" reloadable="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
       prefix="localhost.xbm.log."
       suffix=".txt"
       timestamp="true"
  />
</Context>
```

replacing the CONTEXT_NAME with the real context name, and XBMHOME with the real XBM installation path ($XBMHOME).

Once configured the XML file and restarted the web server simply connect to http://your.tomcat.web.server/CONTEXT_NAME/xbmcw.jsp (where CONTEXT_NAME is the name specified above) to start the remote console.
8.12.2 Installing on a standard Web Server

To make use of Java Web Start (tm) technology with XBM Remote Console simply follow this step:

- configure the Web server so that all files with the .jnlp file extension are set to the application/x-java-jnlp-file MIME type
- update the supplied JNLP files (x4j.jnlp, jh.jnlp, freeutils.jnlp, orajdbc.jnlp) with the correct codebase information
- make the application accessible on the Web server by uploading application's JARs, JNLP and GIF files, at the URLs listed in the JNLP file.
- Create a link from the Web page to the JNLP file. For example:

  `<a href="XBM.jnlp">Launch XBM Remote Console</a>`

This first step is critical for making sure that a Web browser will launch Java Web Start when downloading the file.

A link to the Java Web Start installer should also be provided on the Web page, so users who do not already have Java Web Start installed can download and install the software.
8.12.3 Further information

For further information on Java Web Start(tm) please refer to: http://java.sun.com/products/javawebstart .
Part IX

XCONV Data Conversion Toolkit
During the migration process from mainframe platform to the UNIX/Linux one, data conversion is an obliged step. Data conversion means to translate correctly data, from EBCDIC to ASCII character-set, taking care about different data formats contained in the file.

The XCONV Data Conversion Toolkit provides a quick and smart system to convert data from EBCDIC to ASCII and vice-versa.

Data coming for an RDBMS such IBM DB2 does not require the same handling of files: the conversion of the table data is performed by means of "xddiconv" utility.

### Conversion steps

The conversion of each file always follows this flow:

1. translation of the COBOL record into an XML description using "cpy2xml"
2. creation of the conversion code using "xmlconverter", which automatically generates a COBOL program to convert the file, according to the structure described by the previously generated XML file
3. (optional) coding of a COBOL copybook, containing the structure and/or redefines branch identification rules
4. compiling and running the conversion program to translate the file

Basically, you may distinguish four different categories of files:

- single record structure without critical redefines
- multiple record structures without critical redefines
- single record structures with critical redefines
- multiple record structures with critical redefines

Critical redefines are such data redefinition that lead to an uncompatible conversion, like shown in the following table:

<table>
<thead>
<tr>
<th>picture usage</th>
<th>X DISPLAY</th>
<th>9 DISPLAY</th>
<th>S9 DISPLAY</th>
<th>S9 COMP</th>
<th>S9 COMP-3</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>X DISPLAY</td>
<td>-</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>9 DISPLAY</td>
<td>-</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>S9 DISPLAY</td>
<td>critical</td>
<td>critical</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>S9 COMP</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
</tr>
</tbody>
</table>

### Example Table

<table>
<thead>
<tr>
<th>picture usage</th>
<th>X DISPLAY</th>
<th>9 DISPLAY</th>
<th>S9 DISPLAY</th>
<th>S9 COMP</th>
<th>S9 COMP-3</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>X DISPLAY</td>
<td>-</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>9 DISPLAY</td>
<td>-</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>S9 DISPLAY</td>
<td>critical</td>
<td>critical</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
</tr>
<tr>
<td>S9 COMP</td>
<td>critical</td>
<td>critical</td>
<td>critical</td>
<td>-</td>
<td>critical</td>
<td>critical</td>
</tr>
</tbody>
</table>
Single record structures without critical redefines

The file contains only one record structure without critical redefines. I.e.

```
01 FILE-REC.
  03 NAME.
  05 SURNAME PIC X(40).
  05 FIRST-NAME PIC X(20).
  03 ADDRESS.
  05 CITY PIC X(20).
  05 STREET PIC X(20).
  03 INFO.
  05 CODE PIC S9(8) COMP.
  05 SALARY PIC S9(8) COMP.
  05 DATE-OF-BIRTH PIC X(8).
  05 FILLER REDEFINES DATE-OF-BIRTH.
  07 DATE-YYYY PIC X(4).
  07 DATE-MM PIC X(2).
  07 DATE-DD PIC X(2).
```

This is the simplest file type to convert: XCONV toolkit is able to generate everything is required for the conversion.

Multiple record structure without critical redefines

The file contains more than one well-defined record structure each one without critical redefines. Programs accessing the file, recognize the proper structure according to information contained in the record itself. I.e.

```
01 FILE-REC.
  03 RECORD-TYPE PIC X.
  03 NAME.
  05 SURNAME PIC X(40).
  05 FIRST-NAME PIC X(20).
  03 ADDRESS REDEFINES NAME.
  05 CITY PIC X(20).
  05 STREET PIC X(20).
  03 INFO REDEFINES NAME.
  05 CODE PIC S9(8) COMP.
  05 SALARY PIC S9(8) COMP.
```

In this case the record structure should be divided in different record structures, one for each sub-type (3 in the example above).

The generated converter will contain the code to convert each different record structure. You have to code a copybook containing the rules to recognize, for each record processed, the correct structure to convert it.
Single record structures with critical redefines

The file contains only one record structure with critical redefines.

I.e.

```asciidoc
01 FILE-REC.
  03 SURNAMES           PIC X(40).
  03 FIRST-NAME         PIC X(20).
  03 CITY               PIC X(20).
  03 STREET             PIC X(20).
  03 CODE               PIC S9(8) COMP.
  03 OTHER REDEFINES   CODE PIC X(8).
```

In this situation, the record contains one single record structure, but contains one or more redefines clauses that lead to a critical situation.

The record may be left as is: the converter program will contain the code to handle all the different branches (caused by redefines clauses) of the record. You have to provide a copybook with the rules to recognize, for each record processed, the correct conversion branch to use.

Multiple record structures with critical redefines

The file contains only more than one record structures with critical redefines.

I.e.

```asciidoc
01 FILE-REC.
  03 RECORD-TYPE    PIC X.
  03 NAME.
    05 SURNAMES        PIC X(40).
    05 FIRST-NAME      PIC X(20).
    03 ADDRESS REDEFINES NAME.
    05 CITY            PIC X(20).
    05 STREET          PIC X(20).
    03 INFO REDEFINES  NAME.
    05 CODE            PIC X(8) COMP.
    05 SALARY REDEFINES CODE PIC S9(8) COMP.
```

The record structure should be divided in different record structures, one for each sub-type (3 in the example above).

The generated converter will contain the code to convert each different record structure and, for each one of these, it will contain the code to handle the different branches of the structures.

You have to code a copybook containing the rules to recognize, for each record processed, both the correct structure and the correct branches to convert it.
9.1 XML generation

As described before, the first step of a file conversion, is the generation of an XML definition of the record structures. The XML definition contains the properties of all the fields in the structure.

The tool "cpy2xml" analyzes the COBOL description of the record structure and converts it into XML.

To build the description of a single file, it can analyze one or more COBOL copybooks even containing one or more level 01 items.

I.e.

```
# cpy2xml -o FILEREC.xml FILEREC.cpy
```
9.2 Converter generation

Once built the XML describing the structure(s) of the file, the file's conversion program must be generated.

This task is achieved by means of "xmlconverter", which analyzes the XML description and generates the corresponding COBOL program for the conversion of the file.

By default, the program generated, provides EBCDIC to ASCII conversion.

I.e.

```
xmlconverter -o FILE01.cbl FILE01.xml
```

The converter is COBOL program that, for the whole source file:

- reads the input record
- converts the record
- writes the output record

If a files has been described with multiple structures, the program contains the code to properly convert the different structure.

For each structure defined, it contains the statements to convert every field in the structure and, if critical redefines have been found, it also contains all the possible conversion branches for all the critical situation.

The program does not contain the logic to recognize which structure to use and which redefine branch to apply: the programmer must take care to code such logic.

At generation time it also possible to instruct the xmlconverter to add in the generated code some data checks (i.e. packed field or numeric checks), to get error messages whenever the data being converted is not compatible with the expected data type.

**Data type handling**

When generating the converter program, xmlconverter automatically chooses the default conversion function for each field in the record. To force different conversion method for specific fields, you may insert in the XML descriptor file the "conversion" attribute on the field descriptions (dataitem) you want to handle. The conversion attribute may assume one of the following values:

- auto
- binary
- packed
- char
- dbcs
- float

I.e.

```
<dataitem name="HOUR-PRICE"
  level="5"
  picture="S9(8)"
  length="5"
  usage="comp-3"
  signed="true"
  conversion="char"/>
```
The explanation of different data type is the following:

**binary**

Binary fields are not converted. Their content is the same in both ASCII and EBCDIC character-set.

**packed**

Packed fields are not converted. Only numeric test is possible to validate their content. If a field is not numeric it can be initialized to zero. To force this test it is necessary to run xmlconverter with -p command-line option.

**numeric**

Numeric fields are converted by XCONVZONED if it is signed otherwise it is converted by XCONVCHAR function. It’s also possible to add a validation on its content with -n option. If numeric test is required, function ZERO2NOTNUMERIC is called. If a non-numeric field is detected during check, the conversion process is stopped.

**char**

Char fields are converted by XCONVCHAR function.

**Conversion direction**

By default the generated programs handles EBCDIC to ASCII conversion. To generate a program which converts from ASCII to EBCDIC, force -a parameter on command line when running xmlconverter.
9.3 Rules definition

Once the converter has been generated, the programmer must code the rules to choose the correct structure and branches for those files that requires it.

These rules must be coded in COBOL and stored in a copybook that is automatically referenced, with a COPY statement, at generation time by xmlconverter.

With this code the programmer must identify each record and instructs the converter on which structure to use and, within the structure itself, which branch to follow.

The programmer code is invoked for each record red, and it must identify the structure to use and, within the structure itself, which branch to follow. Of course if only one record structure is defined or if there are no redefines branches there is no need to identify them.

The field RECORD-TYPE identifies the structure to use: the programmer must set this filed to the correct structure number (starting from zero).

Each redefines branch is identified by a field name RDFBRCH-field where field is a the redefined field name. Its value must be set to the corresponding branch to use (starting from zero).

To simplify programmer's life, xmlconverter may generate a skeleton for the rules copybook: the programmer will only have to define the recognition code.

Remember that, for EBCDIC to ASCII conversions, the content of the record red is code in EBCDIC.

Rules samples

Multiple structure without critical redefines

The source copybook (employee.cpy):

```
01 EMPLOYEE.
   03 INFO PIC X.
     88 TYPE-1 VALUE '1'.
     88 TYPE-2 VALUE '2'.
   03 ANAG-INFO.
     05 SURNAME PIC X(20).
     05 NAME  PIC X(20).
   03 SALARY-INFO REDEFINES ANAG-INFO.
     05 HOUR-PRICE PIC S9(8) COMP-3.
     05 TAX-RATE  PIC S9(8) COMP-3.
```

The copybook after structure manual separation (employee_split.cpy):

```
01 EMPLOYEE-1.
   03 INFO PIC X.
   03 ANAG-INFO.
     05 SURNAME PIC X(20).
     05 NAME  PIC X(20).
01 EMPLOYEE-2.
   03 INFO PIC X.
   03 SALARY-INFO.
     05 HOUR-PRICE PIC S9(8) COMP-3.
     05 TAX-RATE  PIC S9(8) COMP-3.
```

Commands:

```
# cpy2xml -o employee.xml employee_split.cpy
```
The rules copybook (employee_rules.cpy) obtained editing manually (bold text) the template generated by xmlconverter:

```
* * template rule for *
* EVALUATE TRUE *
* record type: EMPLOYEE-1 *
   WHEN INFO OF EMPLOYEE-1 = X"F1"
   MOVE 0 TO RECORD-TYPE *
* record type: EMPLOYEE-2 *
   WHEN INFO OF EMPLOYEE-1 = X"F2"
   MOVE 1 TO RECORD-TYPE
   WHEN OTHER
   PERFORM ERROR-MEG
END-EVALUATE
CONTINUE.
```

**Single structure with critical redefines**

The source copybook (employee.cpy):

```
01  EMPLOYEE.           03  SURNAME        PIC X(20).           03  NAME           PIC X(20).           03  CALC-CLASS     PIC X(2).           03  HOUR-PRICE     PIC S9(8) COMP.           03  BILL-RATE REDEFINES HOUR-PRICEx  PIC X(4).
```

Commands:

```
# cpy2xml -o employee.xml employee.cpy
# xmlconverter -R -o employee.cbl -c employee_rules.cpy -t employee_rules.cpy employee.xml
```

The rules copybook (employee_rules.cpy) obtained editing manually (bold text) the template generated by xmlconverter:

```
EVALUATE TRUE
WHEN CALC-CLASS = LOW-VALUES
   MOVE 0 TO RDFBRCH-HOUR-PRICE
WHEN CALC-CLASS = X"F0F0"
   MOVE 1 TO RDFBRCH-HOUR-PRICE
WHEN OTHER
   PERFORM ERROR-MEG
END-EVALUATE
```
Multiple structure with critical redefines

The source copybook (employee.cpy):

```
01 EMPLOYEE.
  03 SURNAME    PIC X(20).
  03 NAME       PIC X(20).
  03 CALC-CLASS PIC X(2).
  03 HOUR-PRICE PIC S9(8) COMP.
  03 BILL-RATE REDEFINES HOUR-PRICE PIC X(4).
```

The copybook after structure manual separation (employee_split.cpy):

```
01 EMPLOYEE-1.
  03 RECORD-TYPE PIC X.
  03 ANAG-INFO.
    05 SURNAME    PIC X(20).
    05 NAME       PIC X(20).
  01 EMPLOYEE-2.
    03 RECORD-TYPE PIC X.
    03 BILLING-INFO.
      05 CALC-CLASS PIC X(2).
      05 ADDR      PIC X(40).
      07 BILL-RATE REDEFINES ADDR.
      07 IND       PIC X.
      07 HOUR-PRICE PIC S9(8) COMP.
      07 DISC-RATE REDEFINES HOUR-PRICE PIC X(4).
```

Commands:

```
# cpy2xml -o employee.xml employee_split.cpy
# xmlconverter -R -o employee.cbl -c employee_rules.cpy -t employee_rules.cpy employee.xml
```

The rules copybook (employee_rules.cpy) obtained editing manually (bold text) the template generated by xmlconverter:

```
EVALUATE TRUE
  * record type: EMPLOYEE-1
  * WHEN INFO OF EMPLOYEE-1 = X"F1"
    MOVE 0 TO RECORD-TYPE
  *
  * record type: EMPLOYEE-2
  *
  WHEN INFO OF EMPLOYEE-1 = X"F2"
    MOVE 1 TO RECORD-TYPE
  * redefines for field ADDR
  EVALUATE TRUE
    WHEN CALC-CLASS = LOW-VALUES
      MOVE 0 TO RDFBRCH-BILLING-INFO
```
WHEN CALC-CLASS = X’FOF0’
    MOVE 1 TO RDFSRC-HBILLING-INFO * redefines for field: HOUR-PRICE
    EVALUATE TRUE
    WHEN IND NOT = X’00’
        MOVE 0 TO RDFSRC-HBILL-RATE
    WHEN IND = X’00’
        MOVE 1 TO RDFSRC-HBILL-RATE
    WHEN OTHER
        PERFORM ERROR-MSG
    END-EVALUATE
    END-EVALUATE
    PERFORM ERROR-MSG
    WHEN OTHER
    END-EVALUATE
    PERFORM ERROR-MSG
    END-EVALUATE
    PERFORM ERROR-MSG
    CONTINUE
9.4 Converter compiling

Once the converter is created and the eventual rules copybook has been defined, it must be compiled with your COBOL compiler.

For Microfocus COBOL:

    cob <program>.cbl

For ACUCOBOL:

    ccbl <program>.cbl
9.5 Converter execution

To run the converter program, you have to use xvsamRts. On the command line, you have to provide input filename and output filename, eventually specifying paths.

\[
\text{xvsamRts <program> <input> <output>}
\]

I.e.:

\[
\text{xvsamRts FILE01 FILE01.EBCDIC ../ascii/FILE01.ASCII}
\]

During its execution, the program reports:

- input and output paths
- record size
- record converted number
- last record read, if an error

Program will issue an error and stop if:

- it can't convert a record because no rule for that record type is available
- file size and record size are not compatible (this check is not applied for variable length record file)

For ACUCOBOL only:

Before to execute the program, you must instruct the xvsamRts to load the XCONV conversion library, using the \text{XRUN\_LIBRARIES} environment variable. The XCONV library is $\text{XFRAMEHOME/lib/libxconv.so} \,(.\text{sl} \text{for HPUX 11.11})$.

\[
\text{export XRUN\_LIBRARIES=}\$\text{XFRAMEHOME/lib/libxconv.so}
\]

**Overriding Codepage**

Alternate codepages may be loaded by the XCONV library by exporting the \text{XCONVCODEPAGE} environment variable. This environment must contain an absolute path name to an XFRAME codepage file.

I.e.:

\[
\text{export XCONV\_CODEPAGE=}$\text{XFRAMEHOME/etc/IBM930.tbl}$
\]

**Setting a DBCS conversion table**

If the conversion requires a DBCS conversion table, the environment variable \text{XCONVDDBCSTABLE} must be exported and set to the full path of an XFRAME codepage table file.

\[
\text{export XCONV\_DBCSTABLE=}$\text{XFRAMEHOME/etc/IBM-Kanji.tbl}$
\]

For further information about XFRAME codepage and DBCS tables, please refer to the XCICS/TS configuration documentation.
9.6 The XML descriptor

The XCONV toolkit handles record structure description with XML. The XML file contains the following elements:

**datadef**

It describes the whole structures set

**datastruct**

It describes one structure. In other words it represents each 01 level COBOL field.

**dataitem**

It describes each COBOL field. Its attributes describe the field. If the field is an elementary one, the dataitem's attributes will also contain attributes describing its memory representation. If the field is a structured one, the element will contain child elements (of the same type).

The possible attribute for dataitem elements are:

<table>
<thead>
<tr>
<th>attribute</th>
<th>required</th>
<th>description</th>
<th>Admissible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>field name</td>
<td></td>
</tr>
<tr>
<td>level</td>
<td>yes</td>
<td>nested item level</td>
<td></td>
</tr>
<tr>
<td>redefines</td>
<td></td>
<td>name of the field redefined</td>
<td></td>
</tr>
<tr>
<td>occurs</td>
<td></td>
<td>max value indexed structure (default 1)</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td></td>
<td>field size in bytes</td>
<td></td>
</tr>
<tr>
<td>picture</td>
<td></td>
<td>COBOL picture clause</td>
<td></td>
</tr>
<tr>
<td>usage</td>
<td></td>
<td>field usage</td>
<td>display comp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>binary</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>comp-4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>comp-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>comp-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>comp-3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>packed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>comp-x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nchar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>japanese</td>
</tr>
<tr>
<td>signed</td>
<td></td>
<td>boolean indicator: true if field is signed</td>
<td>implicit</td>
</tr>
<tr>
<td>sign</td>
<td></td>
<td>sign type for signed fields</td>
<td>trailing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>leading</td>
</tr>
<tr>
<td>conversion</td>
<td>conversion type. Auto default. You could force different type of conversion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>discardable</td>
<td>boolean indicator: false if field is a critical redefines</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9.7 The conversion library

The XCONV conversion library (libxconv.so) contains functions invoked by the converter programs to properly handle the conversion of different data types.

**XCONV**

It must be the first called function, which provides to check file size and stores conversion direction.

**Prototype**

```
int XCONV(unsigned int direction, BYTE *filename, unsigned int record_length)
```

**Parameters**

- **direction** - conversion direction
  - 0 ASCII --> EBCDIC
  - 1 EBCDIC --> ASCII
- **filename** - name of the file to be converted
- **record_length** - record length

**XCONVCHAR**

This function converts character fields

**Prototype**

```
int XCONVCHAR(unsigned char *area, int len)
```

**Parameters**

- **area** - address of the area to be converted
- **len** - size of the area

**XCONVZONED**

This function converts numeric signed display fields

**Prototype**

```
int XCONVZONED(unsigned char *area, int len)
```

**Parameters**

- **area** - address of the area to be converted
- **len** - size of the area
**XCONVDBCS**

This function converts field through a specific codepage table, previously loaded by XCONV.

**Prototype**

```c
int XCONVDBCS(unsigned char *area, int len)
```

**Parameters**

- **area**: address of the area to be converted
- **len**: size of the area

**ZERO2NOTNUMERIC**

This function checks and changes any blank or binary zero (X"00") value into zero char (X"30" ASCII) on numeric signed display fields.

**Prototype**

```c
int ZERO2NOTNUMERIC(unsigned char *area, int len)
```

**Parameters**

- **area**: address of the area to be converted
- **len**: size of the area

**XCONVDUMP**

This function is called automatically on error to display an hexadecimal dump of a wrong record. The program dumps record and stops execution.

**Prototype**

```c
int XCONVDUMP(unsigned char *area, int len)
```

**Parameters**

- **area**: address of the area to be converted
- **len**: size of the area
9.8 cpy2xml

This tool analyzes a COBOL description of a record and generates XML view of the structure.

Each structure is described by the following XML elements:

- datadef
- datastruct
- dataitem

Datadef elements represent the whole structures set. Each structure is described by a datastruct element. The dataitem elements describe both elementary and structured fields.

Syntax

cpy2xml [-options] <file>

Parameters

- -v, --version shows version
- -o, --output <file> specifies output file
- -d, --store-dtd generates DTD (Document Type Definition) file
- -r, --report verbose report
- -i, --ignore-uncritical-redef ignores non critical redefines. If set, the tool does not produce elements for non critical redefines.
- -t, --tag-uncritical-redef tags non critical redefines. If set each critical redefines element is tagged with the attribute discardable = "false"
- -n, --rename-filler renames filler fields with a unique ID
- -h, --help shows help

XML File

The file generated by cpy2xml, contains the following elements and attributes

Datadef

The datadef attribute defines a data structure corrisponding to 01 level in Cobol language.

Its attributes are:

<table>
<thead>
<tr>
<th>attribute</th>
<th>required</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>built</td>
<td></td>
<td>contains the date of creation</td>
</tr>
<tr>
<td>language</td>
<td></td>
<td>language used</td>
</tr>
<tr>
<td>source</td>
<td></td>
<td>the name of source used to create xml file</td>
</tr>
</tbody>
</table>
Datastruct

The datastruct element defines a data structure with elementary subitem or other nested structure.

Dataitem element

The dataitem element defines the elementary field items.

Its attributes are:

<table>
<thead>
<tr>
<th>attribute</th>
<th>required</th>
<th>description</th>
<th>Admittable values</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>field name</td>
<td></td>
</tr>
<tr>
<td>level</td>
<td>yes</td>
<td>nested item level</td>
<td></td>
</tr>
<tr>
<td>redefines</td>
<td></td>
<td>name of the field redefined</td>
<td></td>
</tr>
<tr>
<td>occurs</td>
<td></td>
<td>max value indexed structure (default 1)</td>
<td></td>
</tr>
<tr>
<td>length</td>
<td></td>
<td>field size in bytes</td>
<td></td>
</tr>
<tr>
<td>picture</td>
<td></td>
<td>COBOL picture clause</td>
<td></td>
</tr>
<tr>
<td>usage</td>
<td></td>
<td>field usage</td>
<td>display comp binary comp-4 comp-1 comp-2 comp-3 packed comp-5 comp-x index pointer nchar japanese</td>
</tr>
<tr>
<td>signed</td>
<td></td>
<td>boolean indicator: true if field is signed</td>
<td></td>
</tr>
<tr>
<td>sign</td>
<td></td>
<td>sign type for signed fields</td>
<td>implicit trailing leading</td>
</tr>
</tbody>
</table>

© 2007 H.T.W.C. Srl
Example

```xml
<xml version="1.0" encoding="UTF-8"/>
<datastruct>
    <dataitem level="1" name="X-ACKID0015">
        <dataitem length="4" level="5" name="X-BEGINDATUM" picture="X(04)" signed="false" usage="display"/>
        <dataitem length="8" level="5" name="X-RECDATUM" picture="X(08)" signed="false" usage="display"/>
        <dataitem length="8" level="10" name="X-RECDATUMP" picture="X(08)" signed="false" usage="display"/>
    </dataitem>
    <dataitem level="2" name="X-DUBBEL-IND" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="3" name="X-DUBBEL-IND" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="4" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="5" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="6" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="7" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="8" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="9" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
    <dataitem level="10" name="X-TEKEN" picture="X(01)" signed="false" usage="display"/>
</datastruct>
```
<dataitem length="7" level="5" name="X-REAL-AKTIEKODE" picture="X(07)"
signed="false" usage="display"/>
<dataitem length="5" level="5" name="X-REAL-VANNDUMMER" picture="X(05)"
signed="false" usage="display"/>
<dataitem length="1" level="5" name="X-REAL-BJ7" picture="X(01)"
signed="false" usage="display"/>
<dataitem length="30" level="5" name="X-REAL-RESERVE" picture="X(30)"
signed="false" usage="display"/>
</dataitem>
</datastruct>
9.9 xmlconverter

This tool produces conversion routines from the XML data structure description, created by cpy2xml utility. It may generate several converter types:

- COBOL programs for data conversion
- C language routines for XCICS Online Dynamic Conversion System (ODCS)
- COBOL programs for H2R data conversion
- SQL statements to load H2R fields

**COBOL programs for data conversion**

This is the default output type of xmlconverter. By default the converter will handle a fixed record length file. If file has variable record length, you need to run xmlconverter with `-V` option: The input and output file will be managed as variable size file.

**C language routines for XCICS Online Conversion System**

With option `-C`, xmlconverter produces a C language function source, to be used with the XCICS Online Conversion System.

To create conversion program for DL/I data download, you have to use `-H` option. DL/I data are organized with 16 bytes at the beginning of each record, where DBD and PSB name are stored. The DBD name is normally equal for all records because each DBD is downloaded by a single program into the same file.

The PSB name is variable and the segment name define a particular record description used to translate the record. You have as many record descriptions as DL/I segments.

The segment name is before translated and then tested to identify the right record description to convert the area.

**SQL statements to load H2R fields**

With option `-d`, it generates a SQL script to load the H2R system tables, with the data structure information

**Syntax**

```
xmlconverter [-options] <file>
```

**Parameters**

- `-v`, `--version` show version
- `-a`, `--ascii-to-ebcdic` changes conversion direction from ASCII to EBCDIC
- `-o`, `--output <filename>` redirects standard output to filename
- `-t`, `--write-template <template>` writes a template for the rules coding
- `-c`, `--include-copy <copy>` includes copy before testing record type field
-p, --test-packed          enables test numeric validation on packed fields
-n, --test-numeric         enables test numeric validation on numeric fields
-l, --test-length          disables check on variable file length size (valid only with -var option)
-R, --handle-critical-redefines includes critical redefines handling code
-V, --variable, -var      generates program for variable record files
-C, --c-area               generates C conversion routine (for XCICS Online Conversion System)
-H, --h2r-converter, -h2r generates standard H2R conversion program
-d, --h2r-fields, -db     creates file to load H2R fields info into RDBMS
-h, --help                 shows help
## 9.10 xddlconv

XDDLCONV is a useful utility help you migrate relational database data from mainframe RDBMS to Unix RDBMS. It analyzes DDLs objects and extracts 3 different source:

1. DDL written in Oracle or UDB dialect
2. Copybook data structure to generate program conversion through XCONV toolkit
3. Control File to upload data through sqldi oracle utility or load to upload with load utility

### Syntax

```bash
usage: xddlconv <[options]> [file]
```

### Parameters

- `-h, --help`
  - show command options

- `-D, --docs`
  - show command options and configuration file parameters

- `-v, --verbose`
  - verbose mode

- `-c, --customize <file>`
  - force alternative customization file. Default $HOME/etc/xddlconv.conf

- `-s, --vse`
  - generate COPY and CONTROL FILE for data downloaded by ARIDBS utility. (VSE utility)

- `-d, --db2`
  - generate DDL, COPY and LOAD for UDB destination RDBMS. Default ORACLE.

### Environment

In order to use XDDLCONV utility is mandatory defines 3 environments. One for each kind of generated object. It is recommended to export those environments in xframelocal.conf file.

```bash
# XDDLCONV setting (sql conversion)
export XDDLCONTRL=$HOME/db2/ctl  # Control file path
export XDDLCOPY=$HOME/db2/import/diction  # Copybook path
export XDDLDDL=$HOME/db2/sql   # Ddl path
```

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDDLCONTRL</td>
<td>Path where XDDLCONV generates control file or load (DB2) modules.</td>
</tr>
<tr>
<td>XDDLCOPY</td>
<td>Path where XDDLCONV generates copybook to convert data. Through those copy you can perform data conversion process generating xml and program conversion objects.</td>
</tr>
<tr>
<td>XDDLDDL</td>
<td>Path where XDDLCONV generates ddls for target RDBMS. It divide table and index into 2 different scripts in order to create table, load data and at the end create indexes.</td>
</tr>
</tbody>
</table>
Configuration file

XDDLCONV has a useful customization file that permits to customize some download or upload parameters. Entire help (--docs) shows and resums all this options. XDDLCONV looking for customization file in $HOME/etc named xddlconv.conf, if you use -c|--customize option followed file name you can force alternative file. All date and time option are valid only for ORACLE RDBMS.

Notes

Rows prefixed by '#' are handled as remarks. All commands in metalanguage are prefixed by '@' and must be in lowercase. When code is used, it is PERL 5 code.

Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@set db2 = on/off</td>
<td>set on if final rehosting DB is UDB. Default is ORACLE (equal to submit xddlconv with -d</td>
</tr>
<tr>
<td>@set vse = on/off</td>
<td>set on if data downloaded with ARIDBS utility (VSE system) (equal to submit xddlconv with -s</td>
</tr>
<tr>
<td>@set vse_unload_ebcdic = on/off</td>
<td>Cut 1 byte from record length (VSE system)</td>
</tr>
<tr>
<td>@set omit_schema = on/off</td>
<td>write table name without schema (ex. PROD. TABLE_NAME TABLE_NAME )</td>
</tr>
<tr>
<td>@set time2char = on/off</td>
<td>time field are managed as char (default is on)</td>
</tr>
<tr>
<td>@set time_format = HH24:mi:ss</td>
<td>set time format to load data ( is set to on if time2char is off )</td>
</tr>
<tr>
<td>@set data_format = DD.MM.YYYY</td>
<td>set date format to load data</td>
</tr>
<tr>
<td>@set timestamp_format = YYYY-MM-DD-HH24.MI.SS</td>
<td>set time format to load data</td>
</tr>
</tbody>
</table>

Example

conversion from DB2 (MVS) vs ORACLE:

```bash
# xmsxconv.conf
#
# file parametrizzazione della conversione dei JCLs per BCCAP
#
# Omissione del OWNER sul nome tabella
@set omit_schema=on
@set data_format=YYYY-MM-DD
@set timestamp_format=YYYY-MM-DD-HH24.MI.SS
@set time2char=off
```

Generate table without schema name (OWNER). Sqlldr utility require for date data types the field format. If format no match with data error occurs. You can specify date format for date, time, and timestamp fields, forcing corresponding set statement. In this case only DATE and TIMESTAMP are force to specify value. In this case time data types as treat as date fields. If you force time2char setting on you create transform time field in CHAR(S) field.
conversion from DB2 (VSE) to ORACLE:

```bash
# xmvsconv.conf
#
# file parametrizzazione della conversione dei JCLs per BCCAP
#
@set omit_schema=on
@set vse=on
@set vse_unload_ebcdic=on
@set time_format=HH24:MI:SS
@set date_format=YYYY-MM-DD
@set timestamp_format=YYYY-MM-DD-HH24.MI.SS;
```

VSE system provide ARIDBS utility to download data. This utility force packed and binary fields to zoned data. So you have to set on vse options to convert and load data without problem. If you have data in ebcdic character set set vse_unload_ebcdic to on.

conversion from DB2(MVS) to UDB:

```bash
# xmvsconv.conf
#
# file parametrizzazione della conversione dei JCLs per BCCAP
#
@set omit_schema=off
@set db2=on
```

In UDB we have a different utility to load data, and also DDLs have a little bit change from original version.
9.11 Mainframe DB2 to other RDBMS

Data coming from IBM DB2 for mainframe systems, must be converted into ASCII and loaded into the target database. XFRAME contains utilities to speed-up data movement from mainframe DB2 to Oracle and IBM UDB for Unix and Linux.
9.11.1 Downloading data from DB2

The first step in the data migration is the data unloading on the source platform. Depending on the source Operating System, different utilities may be achieve this goal.

**DSNTIAUL**

DSNTIAUL is sample unload program provided with IBM DB2 on MVS, OS/390 and z/OS systems. This program, which is written in assembler language, unloads some or all rows from up to 100 DB2 tables. With DSNTIAUL, you can unload data of any DB2 built-in data type or distinct type. You can unload up to 32KB of data from a LOB column. DSNTIAUL unloads the rows in a form that is compatible with the LOAD utility and generates utility control statements for LOAD. DSNTIAUL also lets you execute any SQL non-SELECT statement that can be executed dynamically.

```
//UNLOAD2 JOB CLASS=A MSGCLASS=~, REGION=7M COND=(12,II)
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*                     /*
//SYSPUNCH DD SYSOUT=*                    /*
//SYSDUMP DD SYSOUT=*                     /*
/*INS*/
//SYSSIN DD *                            /*
SELECT * FROM OWNERL.TABELLA1;
SELECT * FROM OWNER2.TABELLA2;
/*OOS*/
//SYSSREC01 DD DSN=SCARICO.REC001,
  // UNIT=3390, VOL=SER=HTGO02,
  // SPACE=(CYL (10,10), RLSB),
  // DISP=(CATLG,DELETE)
//SYSSREC02 DD DSN=SCARICO.REC002,
  // UNIT=3390, VOL=SER=HTGO02,
  // SPACE=(CYL (10,10), RLSB),
  // DISP=(CATLG,DELETE)
//SYSSPUNCH DD DSN=SCARICO.PUNCH,
  // UNIT=3390, VOL=SER=HTGO02,
  // SPACE=(CYL (5,5), RLSB),
  // DISP=(CATLG,DELETE)
//SYSSIN DD *
  DSN SYSTEM DSN1
RUN PROGRAM (DSNTIAUL) PLAN (DSNTIB31) PARM ("SQL") -
  LIB ('DSN510.RUNLIB.MVO')
/*
```

Output file is a flat fixed-length file.

**ARIDBS**

On VSE system, ARIDBS is the utility to download data from DB2 (SQL/DS). Remember to set the --vse option when using xddiconv to generate objects.
* * $S$ JOB JNM=QFCHD00, CLASS=8, DISP=O, PRT=0, USER=EDP
* * $S$ LST CLASS=V, DISP=U, JSEP=0, LST=OOE
* * $S$ LST CLASS=V, DISP=U, JSEP=0, LST=02E, DEST=(*, TRANSFER)
// JOB QFCHD00
// ASSGN SYSLST, OOE
// SETPFIX LIMIT=40K
// OPTION NODUMP
// EXEC IDCAMS, SIZE=AUTO
//   DELETE (SQLPRVA.QFCHD00) PURGE CAT(MCALT)
//   /*
//   D8L DASD1, "SQLPRVA.QFCHD00", D, VSAM, DISP=NEW, KEYP, CAT=M2CAT7, RECORDS=20000, RECSIZE=542
//   EXECUTE, M2CAT7
//   EXEC ARIDS, SIZE=AUTO
//   CONNECT SQLDBA IDENTIFIED BY SYS2;
//   CONNECT TO SQLPRVA;
//   DATAUNLOAD
//   SELECT * FROM SQLDBA.QFCHD00;
//   OUTFILE(DASD1, PDEV(DASD))
//   */
// *
* $S$ EOJ
9.11.2 DB2 to ORACLE

Encoding data conversion is defined as the changing of character-set in data from EBCDIC to ASCII. The easiest way to perform this task for DB2 EBCDIC data is download its in flat file and converts its as common EBCDIC flat file through programs conversion. At the end of conversion reload ASCII data on Oracle database. In order to replicate DB2 objects schema is mandatory obtain from customer the up-to-date DDLs. XDDLCONV analyzing ddfs produce 3 different source.

1. DDL written in Oracle dialect
2. Copybook data structure to generate program conversion through XCONV toolkit
3. Control File to upload data through sqlldr oracle utility

Unload data

It is mandatory unload DB2 data through DSN TIAUL db2 utility because download VARCHAR data type as fixed length field. In VSE system you have to download using ARIDBS utility.

Analyze DDLs

DDLs have to be processed by XDDLCONV utility. See more detailed information in XCONV book.

Create Oracle workspace

In order to replicate DB2 DB you have to create user and tablespace. In appendix B can find an example how to create a oracle user and related tablespace. Tablespaces size depends on amount of data. After tablespace and user definition, tables have to be created. In order to upload data quickly and without hierarchy priority problem we suggest to create index at the end of uploading step. In fact XDDLCONV utility separates index and table source.

Process all *.sql file produced by XDDLCONV utility to sqlplus using user created.

```
sqlplus MYSQL/MYSQL@MYDB @tabella.sql
```

Now all table are replicated empty under Oracle DB. After data conversion you can proceed with upload step.

Data Conversion

Data file in conversion step has to be treated as flat file. Fortunately XDDLCONV provides to generate necessary copybooks. Is mandatory to generate through cpy2xml and xmlconverter utilities programs conversion.

Note:
xmlconverter command can be lunched without additional flags. Is mandatory name output ASCII file as TABLE.ASC.
For more detailed instruction to produce programs conversion look at XCONV data conversion toolkit.

Data upload

All files have been converted from EBCDIC to ASCII on a per field basis. Now upload data with sqlldr Oracle utility. To perform this step you need control files generated by XDDLCONV step. Perform this command from directory where ASCII files were created.

```
sqlldr userid=MYSQL/MYSQL@MYDB control=$CONTROL_PATH/TABELLA.ctl
```

For more detailed information about sqlldr see Oracle Utilities Guide.
Example

Original DDL

```sql
CREATE TABLE SCHEMA.TABLE1
(
    NAME                    CHAR(3) NOT NULL,
    SURNAME                 CHAR(2) NOT NULL,
    ADDRESS                 CHAR(35) NOT NULL,
    STATE                   CHAR(34) NOT NULL,
    ZIPCODE                 DECIMAL(5, 0) NOT NULL,
    BIRTHDAY                DATE NOT NULL
)
```

DDL

```sql
DROP TABLE TABLE1;
CREATE TABLE TABLE1
(
    NAME                     CHAR(3) NOT NULL,
    SURNAME                  CHAR(2) NOT NULL,
    ADDRESS                  CHAR(35) NOT NULL,
    STATE                    CHAR(34) NOT NULL,
    ZIPCODE                  DECIMAL(5, 0) NOT NULL,
    BIRTHDAY                 DATE NOT NULL
);
COMMIT;
EXIT;
```

Copy

```sql
01 TABLE1.
  03 NAME              PIC X(3).
  03 SURNAME           PIC X(2).
  03 ADDRESS           PIC X(35).
  03 STATE             PIC X(34).
  03 ZIPCODE           PIC S9(5) COMP-3.
  03 BIRTHDAY          PIC X(10).
```

Control file

```sql
--- generated by xddlconv $Revision: 1.20 $---
on Wed Oct 19 16:25:43 MEST 2005
LOAD DATA
INFILE 'TABLE1.ASC' " fix 87 "
BINARY 'TABLE1.BIN'
REPLACE
PRESERVE BLANKS
INFO TABLE TABLE1
{
    NAME POSITION (  1 ) CHAR (  3 ),
    SURNAME POSITION (  4 ) CHAR (  2 ),
    ADDRESS POSITION (  6 ) CHAR ( 35 ),
    STATE  POSITION ( 41 ) CHAR ( 34 ),
    ZIPCODE POSITION ( 75 ) DECIMAL (5, 0 ),
    BIRTHDAY POSITION ( 78 ) DATE 'YYYY-MM-DD',
```
9.11.3 DB2 to UDB

The best way to transfer data between mainframe DB2 and UDB on Unix or Linux is to use the UDB native features for data movement. Sometimes these features may not be used (i.e. the mainframe and the Unix system are not physically connected) and therefore the XFRAME utilities for data migration may be a good solution for the data transfer.

Encoding data conversion is defined as the changing of character-set in data from EBCDIC to ASCII. The easiest way to perform this task for DB2 EBCDIC data is download its in flat file and converts its as common EBCDIC flat file through programs conversion. At the end of conversion reload ASCII data on Oracle database. In order to replicate DB2 objects schema is mandatory obtain from customer the up-to-date DDLs. XDDLCONV analyzing ddfs produce 3 different source.

1. DDLs separate primary key, index and foreign key from create table
2. Copybook data structure to generate program conversion through XCONV toolkit
3. Control File to upload data through sqldlr oracle utility

Unload data

Is mandatory unload DB2 data through DSNTIAUL db2 utility because download VARCHAR data type as fixed length field. In VSE system you have to download using ARIDBS utility. In Appendix A can find an example to run those utilities from JCLs.

Analyze DDLs

DDLs have to be processed by XDDLCONV utility. See more detailed information in XCONV book.

Create UDB workspace

In order to replicate DB2 DB you have to create user and tablespace. Tablespaces size depends on amount of data. After tablespace and user definition, tables have to be created. In order to upload data quickly and without hierarchy priority problem we suggest to create index at the end of uploading step. In fact XDDLCONV utility separates index and table source.

Process all *.sql file produced by XDDLCONV utility to DB2 using user created. DB2 isn't db2 connection command, but is a Korn shell that encapsulate db2 connection ... see example in Appendix C

```
DB2 tabella.sql
```

Now all table are replicated empty under UDB DB. After data conversion you can proceed with upload step.

Data Conversion

Data file in conversion step has to be treated as flat file. Fortunately XDDLCONV provides to generate necessary copybooks. Is mandatory to generate through cpy2xml and xmlconverter utilities programs conversion.

Note:

xmlconverter command can be lunched without additional flags. Is mandatory name output ASCII file as TABLE.ASC.

For more detailed instruction to produce programs conversion look at XCONV data conversion toolkit.

Data upload

All files have been converted from EBCDIC to ASCII on a per field basis. Now upload data with load UDB utility. To perform this step you need load files generated by XDDLCONV step. Perform this command from directory where load files were created. LOAD is as DB2 a Korn shell. See Appendix C for more details.

```
LOAD TABELLA load
```

For more detailed information about load see UDB Guide.
Example

Original DDL

CREATE TABLE SCHEMA.TABLE
    (IDESNELAV DECIMAL(6) NOT NULL,
     NUMPAG DECIMAL(4) NOT NULL,
     INIDDVPAG DECIMAL(10) NOT NULL,
     FINDDVPAG DECIMAL(10) NOT NULL,
     IN IZPNK30A TC30AD66
    AUDIT NONE
    CCSID EBCDIC;
COMMENT ON TABLE SCHEMA.TABLE IS 'This is a comment';
ALTER TABLE SCHEMA.TABLE FOREIGN KEY C30SNE18
    (IDESNELAV )
    REFERENCES SCHEMA.TABLE1
    (IDESNELAV)
    ON DELETE CASCADE;

DDL

DROP TABLE SCHEMA.TABLE;
CREATE TABLE SCHEMA.TABLE (IDESNELAV DECIMAL(6, 0) NOT NULL, NUMPAG DECIMAL(4, 0) NOT NULL, INIDDVPAG DECIMAL(10, 0) NOT NULL, FINDDVPAG DECIMAL(10, 0) NOT NULL);

Index file

ALTER TABLE SCHEMA.TABLE ADD CONSTRAINT SCHEMA.TABLE_C30SNE18 FOREIGN KEY (IDESNELAV )
REFERENCES SCHEMA.DSNELAV (IDESNELAV )
ON DELETE CASCADE;

Copy

01 TABLE.
  03 IDESNELAV PIC S9(6) COMP-3.
  03 NUMPAG PIC S9(4) COMP-3.
  03 INIDDVPAG PIC S9(10) COMP-3.
  03 FINDDVPAG PIC S9(10) COMP-3.

Load

-- generated by xddlconv $Revision: 1.20 $
-- on Wed Oct 12 12:46:52 MDT 2005
--
CONNECT TO @DBNAME USER @USER USING @PASSWORD;
LOAD (CLIENT) FROM "@PATH_TABLESCHEMA_TABLE.ASC" OF ASCII
MODIFIED BY BINARYDECIMAL NULLINDICATOR PackedDECIMAL RECLEN=19 STRIPTBLANKS
METHOD L (1 4, 5 7, 8 13, 14 19)
NULL INDICATORS (0, 0, 0, 0)
MESSAGES "BEGIN LOG SCHEMA TABLE log"
REPLACE INTO SCHEMA TABLE
( IDESNELAV, NUMPAQ, INIDDVPAG, FINDDVPAG)
COPY NO INDEXING MODE AUTOSELECT;
CONNECT RESET;
;
Part

XSPOOL
10 XSPOOL

XSPOOL is a software for print files management. The organization of these files is based on a repository, one for each instance of XSPOOL, that contains all the print files, catalogued for groups and users and characterized by some accessory information as, for example, creation date, sysout id, job name, etc.

Print files are added to repository through the XSPOOL batch interface by external applications (i.e. scripts that launch print programs).

The graphic interface of XSPOOL allows to display and manage all print files using any web browser.

Every user who has access to an XSPOOL repository can:

- display the list of printouts
- display and search the contents of the single printout
- change the printing parameters of the printout
- duplicate the printouts
- delete printouts in temporary or definitive mode
- recover printouts previously deleted in temporary mode
- print objects (entire or single pages)

Printing sub-system

XSPOOL prints the files via the lpd sub-system. This means that XSPOOL does not interface directly the printers, but it issues an "lp" commands to deliver the file to print to the lpd sub-system.

The printing can be executed:

- from the web application
- directly from the scripts, with xspooladd

When the printing file is put in the repository (with xspooladd), XSPOOL reads xspool.conf configuration file and looks for the settings defined for the user and gets all the printing information to store file and its printing characteristics in the repository. Of course, as described in the following chapters, the information retrieved from the configuration file, may be overridden by the xspooladd options.

When the printout of the file is requested, XSPOOL maps accordingly to its configuration, the printing characteristics of the file into the corresponding "lp" options, executes the "lp" command and analyzes its response.

A practical example.

User "foo" has priority "A" and print class "CL01". He adds to the repository the file "JOB00.130.IGN.lst" using xspooladd: the file is stored with class CL01 and priority A.

```
[User]
id = foo
isAdmin = yes
class = CL01
priority = A
form = FA
cmds = cmd01 cmd02 cmd03
```

When foo requests the printout of the file, XSPOOL searches the file's class (CL01) and priority (A) in the XSPOOL configuration, and determines the corresponding "lp" parameters:

```
[Class]
id = CL01
```
value = lpDev01
description = printer 01

[Priority]
id = A
value = 0
description = low

In the example above the XSPOOL class "CL01" is bind to the lp printer class "lpDev01" and the priority "A" is bind to the lp priority "0". Therefore XSPOOL composes and executes the following command (i.e. on IBM AIX 5.3):

```
lp -q 0 -d lpDev01 $HOME/spool/groupa/foo/queue/JOB00.130.20060505.013322.CL01.A.2.B.df.lst
```

**Requirements**

XSPOOL requires on the server side the following software

- Jakarta Tomcat 4.1.12
- Java 1.4.0 or higher
- Perl 5.6

On the client side only a standard web browser is required (Microsoft Internet Explorer 6.0 or higher, Netscape Navigator 6.0 or higher)
10.1 Components list

XSPOOL distribution contains the following items:

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xspool.war</td>
<td>visual application Xspool</td>
</tr>
<tr>
<td>xspooladd</td>
<td>perl script used to add into the spool the files produced into the JCLs.</td>
</tr>
<tr>
<td>xspoolpr</td>
<td>perl script used to execute the lp system command.</td>
</tr>
<tr>
<td>xspoollib.pl</td>
<td>perl library of functions used by xspooladd and xspoolpr.</td>
</tr>
<tr>
<td>xspool.conf</td>
<td>example of configuration file used by visual application and perl scripts.</td>
</tr>
<tr>
<td>xspool.xml</td>
<td>example of spool context file used by Tomcat.</td>
</tr>
</tbody>
</table>

The installation procedure generate the following structure:

```
<XPOOL_HOME> --- xspool ----+- bin ----+- xspooladd 
 |              |          +- xspoollib.pl 
 |              |          +- xspoolpr 
 |              +- etc ----+- xspool.conf.Example 
 |                  |          +- xspool.xml.Example 
 |                  +- webapp ----+- xspool.war 
```

where <XPOOL_HOME> is the directory given in the installation phase.
10.2 Filter

XSPOOL allows the possibility to process the file with a user-written program, either before to store it in the repository or before to print it.

**Pre-feeding filters**

Print files may be processed by a filter program before they are stored in the repository. This kind of filtering allows to modify in permanent mode the print files: the result of the filtering process will be store in the repository and made available to the users.

This kind of filtering may be used to perform some changes on print file like, i.e., transforming characters from lower to upper case, to suppress blank line, to transform ASA style printout into human-readable formats, etc.

This mode of filtering is activated by the "-f" option of xspooladd command.

**Pre-printing filters**

XSPOOL may invoke a filtering program before to process a file before to print it. This kind of filter does not modify the file stored in the repository, but simple changes what is delivered to the printing sub-systems.

Normally this kind of filter is used to adapt the file to the printer, adding specific print control characters.

The pre-printing filter is defined in the print class definition, that means that every single printer class may have its own private filtering program, like in the following sample:

```plaintext
# Destinations (printers or classes of printers)

[Class]
id = CL01
value = ijet
description = InkJet Printer
filter = /home/user/bin/ctrlchar_InkJet.sh

[Class]
id = CLAB
value = lasert
description = Laser printer
filter = /home/user/bin/ctrlchar_Laser.sh
```

Filter variable, if defined, contains the name (if it may be found in PATH) or the full path of an user-defined executable command.

**Writing a filter**

A Filter is an executable file, and they may be written in every programming or scripting language (C, Shell, Perl, etc). Their structure is quite easy:

- read the input file
- process the input
- write the output

XSPOOL filters handle their I/O over the standard streams (STDIN/STDOUT). This means that XSPOOL will provide the input to the filter through STDIN and it will retrieve the filter output on its STDOUT.

This is a sample of a filter that only translates from lower case to upper case:
If the environment variable XSPOOLADD_FILTER_BY_ARGUMENTS is set to YES, XSPOOL invokes the pre-feeding filters providing the input file name as first command line argument and the output file name as the second one. In this case the sample above would be coded as follows:

```
#!/bin/sh
#
# script: filter.sh
# usage: filter.sh INPUT_FILE OUTPUT_FILE
#
tr '[a-z]' '[A-Z]' < $1 > $2
```
10.3 Configuration

The XSPOOL web application is based on JSP technology. Tomcat must be configured according to the following directives in order to execute XSPOOL.

As general guideline, the XSPOOL application must have the right to manage the files contained in the XSPOOL repository, therefore it must be deployed on a webserver run as a user with the right to read and write the repository. Normally it means that tomcat should be started by one of the following users:

- root
- the same user that owns the repository
- an user that has read/write permissions on the repository

There are three aspects to configure, to have the web application working properly:

- the web context
- the users
- XSPOOL behaviour

When tomcat is started, it load the the web context configuration from xspool.xml and the users definitions from tomcat-user.xml. XSPOOL web application reads the configuration information directly from xspool.conf.

**xspool.xml**

This file defines the web application context for XSPOOL.

The file xspool.xml must be located into $CATALINA_HOME/webapps directory and it must contain the context configuration of the spool for Tomcat.

i.e.

```xml
<Context path="/spoolacme"
    docBase="/opt/xspool/webapp/xspool.war" debug="0"
    reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
        prefix="spool_acme_log."
        suffix=".txt"
        timestamp="true"/>
    <Parameter name="xspoolConf"
        value="/home/acme/spool/conf/xspool.conf"
        override="false"/>
    <Parameter name="languageId"
        value="EN"
        override="false"/>
    <Parameter name="spoolRootName"
        value="Spool_Acme"
        override="false"/>
    <Parameter name="spoolRootPath"
        value="/home/acme/spool"
        override="false"/>
    <Parameter name="xspoolprCmd"
        value="/opt/xspool/bin/xspoolpr"
        override="false"/>
</Context>
```

Description:

<p>| path | identifier for the access to visual application by Internet browser (eg. <a href="http://localhost:8080/spoolacme">http://localhost:8080/spoolacme</a>). |</p>
<table>
<thead>
<tr>
<th><strong>xspoolConf</strong></th>
<th>absolute pathname of the configuration file (xspool.conf).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>languageId</strong></td>
<td>setting for language application. The possible values are:</td>
</tr>
<tr>
<td></td>
<td>- EN (English)</td>
</tr>
<tr>
<td></td>
<td>- IT (Italian)</td>
</tr>
<tr>
<td><strong>spoolRootName</strong></td>
<td>Title of the web application</td>
</tr>
<tr>
<td><strong>spoolRootPath</strong></td>
<td>absolute path of the spool directory.</td>
</tr>
<tr>
<td><strong>xspoolprCmd</strong></td>
<td>absolute pathname of the script that executes the print command (xspoolpr).</td>
</tr>
</tbody>
</table>

For further information on web context definition with Tomcat, please refer to [http://tomcat.apache.org](http://tomcat.apache.org).

**tomcat-users.xml**

The file `tomcat-users.xml` is located into `$CATALINA_HOME/conf` directory and contains the list of users authorized to login to the web application.

I.e.

```xml
<user name="user1" password="abc" roles="xspool"/>
<user name="user2" password="defg" roles="xspool,otherrole"/>
<user name="user3" password="" roles="xspool"/>
```

All the users enabled to use XSPOOL, must be defined with the role "xspool". The administrator user must be named "root". This name in not modifiable.

Storing users on the `tomcat` XML users file is a suitable solution only when a limited amount of users must be defined. If you want to define a large number of user, it would be better to store them on a database. For further information on own to administer tomcat users, please refer to [http://tomcat.apache.org](http://tomcat.apache.org).

**xspool.conf**

The file `xspool.conf` must be located in the directory defined in `xspoolConf` parameter of `xspool.xml` and it contains the configuration of spool environment.

The information are stored in tags and each single tag contains its properties.

The syntax of this file must respect the follow rules:

- each row must contain a remark or a tag or a property only
- the remarks must begin with the "#" character
- the formalism of the tags is `[tag]`
- the formalism of the properties is `property = value`
- the value of id properties cannot contain the "." character

There are three fundamental groups of tags:

- tags that define printing information (`[Priority]`, `[Form]`, `[Class]`)
- tags that define user commands (`[Cmd]`)

For further information on web context definition with Tomcat, please refer to [http://tomcat.apache.org](http://tomcat.apache.org).
- tags setting the defaults ([Default], [User])

### Class tag
Each class tag defines a printing class.

<table>
<thead>
<tr>
<th>id</th>
<th>Class destination identifier (used in the file names and given to xspooladd) Each id can't be repeated in tags of the same kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Corresponding value for lp command</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
<tr>
<td>filter</td>
<td>(optional) Pathname of the filter program to process the file before printing</td>
</tr>
<tr>
<td>lpoptions</td>
<td>Extra options to add in the lp command</td>
</tr>
</tbody>
</table>

### Priority tag
This tag defines a printing priority.

<table>
<thead>
<tr>
<th>id</th>
<th>priority identifier (used in the file names and given to xspooladd) Each id can't be repeated in tags of the same kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Corresponding value for lp command</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
</tbody>
</table>

### Form tag
(deprecated) This tag defines a printing form.

<table>
<thead>
<tr>
<th>id</th>
<th>Form identifier (used in the file names and given to xspooladd) Each id can't be repeated in tags of the same kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Corresponding value for lp command</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
</tr>
</tbody>
</table>

### User command
The user commands group is optional and you can use it to define the programs should be made available in the "User Commands" page of the web application.

### Cmd tag
This tag defines an user command that will be available in the "User Commands" page of the web application.

| id          | User command identifier. Each id can't be repeated in tags of the same kind                                                   |
name | Indicate the fullname of the executable script or program to run.
description | Description

### Default tag

This tag contains the default settings for all the users.

| priority | General default priority identifier |
| form | General default form identifier |
| class | General class identifier |
| cmds | Default command identifiers |

### User tag

This tag defines user specific defaults

| id | user name |
| priority | user's default priority |
| form | user's default form |
| class | user's default class |
| isAdmin | if set to "yes", the user is considered as administrator. |
| priorities_enabled | This field indicate all the priority identifiers available for user. Each value must be separated by space |
| forms_enabled | This field indicate all the form identifiers available for user. Each value must be separated by space |
| classes_enabled | This field indicate all the print class available for user. Each value must be separated by a blank |
| cmds | This field indicate all the command identifiers available for user. Each value must be separated by a blank |

The priorities_enabled, forms_enabled and classes_enabled properties may be used to limit the user's choice in the menus of the web application. These properties must contain the allowed identifiers separated by blanks, if they are empty or not defined, the user can see all values available. The use of the reserved keyword "none" permits to deny the access to all values and consequently the printing.

The keyword "df" may be specified in any tag, indicate the system's default value.

I.E.

```
#*******************************************************************************
# # xspool.conf.Example
# #
```
## Priority

- **id** = L
  - value = 0
  - description = low

- **id** = M
  - value = 4
  - description = medium

- **id** = H
  - value = 7
  - description = high

## Forms

- **id** = FA
  - value = formA
  - description = description of form A

- **id** = FB
  - value = formB
  - description = description of form B

- **id** = FC
  - value = formC
  - description = description of form C

## Destinations (printers or classes of printers)

- **id** = CL01
  - value = ipDev01
  - description = printer 01
  - filter = cat -n
  - lqoptions = extra options

- **id** = CL02
  - value = ipDev02
  - description = printer 02

## Global default

- **Default**
```
class = CL01
form = df
priority = L
sysout = 0
cmds = cmd01 cmd02

# tree.arranged by: start value for tree arrangement
# possible values: OWNER (default), JOB, CLASS, FORM, DATE
#
# tree.arranged by = JOB
#
# set dialog of xspooladd and xspoolpr to verbose
#
xspooladdVerbose = yes
xspoolprVerbose = yes

# defines the priority option of "1p" command of "xspoolpr"
# es. HP = HP/UX --> -p
# IBM = AIX (System V) --> -q
# SUN = Solaris --> -q
#
lpPriorityOption = -p

# Users default
#
[User]
id = user1
isAdmin = yes
class = CL01
priority = A
form = FA
cmds = cmd01 cmd02 cmd03

[User]
id = user2
class = CL02
classes_enabled = none
priority = L
priorities_enabled = none
form = FB
forms_enabled = FB df

[User]
id = user3
class = CL02
classes_enabled = df CL01 CL2
priority = M
priorities_enabled = M B
form = FC
forms_enabled = FB FC df
cmds = cmd02

# User commands
```

© 2007 H.T.W.C. Sri
How to start Tomcat

Before using XSPOOL, make sure tomcat has started.

If it's down, it can be launched with startup.sh script, located into $CATALINA_HOME/bin

```
sh startup.sh
```

At this point, you can type on your browser the address of xspool and start work.
10.4 User Interface

To connect to the XSPOOL web application, enter the URL in your browser:

i.e.

http://aix01.ht.net:8081/xspool

The browser will ask for username and password.

It necessary remind that every user must be configured on xspool.conf and tomcat-users.xml files (for more information refer to [Configuration](#)).

**Application objects**

The main screen of the application is composed by two frames:

**Navigation frame**

This area is positioned at the left hand of page and contains different items depending on the user profile.

**Data frame**

This area is positioned at the right hand of page and shows the different working pages of the application.
**Navigation area**

This area contains the command and the tree representation of the spool repository which is accessible by the user. Administrator users may see all the repository.

A simple user, can only see own directories (queue and deleted).
By default, the tree structure is suit by user. It can be change using the appropriate pop up menu.

The lowest level of the structure is composed from queue and deleted that are the real containers of the objects.

The queue directory contains files that user added on XSPOOL repository while the deleted directory contains files that user deleted in temporary mode. When the user click on one of these directories, the data area shows the corresponding list of items with their characteristic.
In order to organize tree structure there are three functions.

The command **Expand All** visualizes the entire tree structure. All directories and sub-directories are shown.

The command **Collapse All** visualizes only the directories at the highest level.

The command **Refresh tree** reloads the tree structure.

The **User commands** function loads the appropriate page to launch the executable programs defined in xspool.conf (for more details refer to User Commands).

Moreover the administrator (root user) can dynamically reload the XSPOOL configuration (xspool.conf) using the button at the bottom of the tree.

### Data frame

This area, situated on the right hand of the page, contains the list of print files contained in the directory selected on tree area.

For each file the following attributes are shown:

- name of the job
- description
- unique number used to identify the generator.
- creation date
- file size
- print class
- priority level
- number of copies to print
- form is a deprecated characteristic. This fields usually contain default value.
- sysout identifier

The number and the content of these columns may change in function of the organization type applied to the tree.
At the top of data area there are the following two windows:

The object box supplies information about the number of objects in the selection and allows the navigation among the pages.

At the top of the page, there is menu of available functions. These functions change depending on the selected container or area of work. If "queue" directory is selected the following functions are available:

- **Refresh**: reloads the list.
- **Print**: prints (if allows) the selected objects.
- **Show**: show the content of the selected objects.
- **Edit**: starts the editing of the printing attributes of the selected object.
If "deleted" directory is selected the following functions are available:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate</td>
<td>duplicates of the selected objects.</td>
</tr>
<tr>
<td>Delete</td>
<td>deletes the selected object.</td>
</tr>
<tr>
<td>Filter</td>
<td>activates the filter function.</td>
</tr>
</tbody>
</table>

Except Refresh, all the functions demand the selection at least an object: such selection is obtained marking the case to the beginning of every line of the list.

**Show Function**

The show function displays the content of the objects.

This page shows the name and location of the selected object and the its content.

If more than one object are selected, over the name area, the navigation line appears to allows the access to the following file.
The number of rows displayed depends on the values set through the SetPage function.

If there are more pages, the navigation line appears and allows the access to the following page.

The content of file is formatted depending on the "Row per Page" defined and the page interruptions present in the text (new line and page break characters).

The following functions are available on the top menu:
The Find function shows a box to enter the string to be searched.

If the string is found, the page will show the search results.

The results may be navigated with "previous" and "next" or listed clicking on "list". The result occurrence are highlighted in the text.
Edit attributes

The Edit function allows the modification of the file’s attribute.

The Update button applies the request changes.

When more than one object is selected, the navigation bar appears to move among the selected items.

The menu contains:

<table>
<thead>
<tr>
<th>Back</th>
<th>Refresh</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>It comes back to the list page.</td>
</tr>
<tr>
<td></td>
<td>It reloads the object in order to include possible changes.</td>
</tr>
</tbody>
</table>

Duplicate objects

The duplicate function loads allows to create a copy of the object in another container.
The Duplicate button executes the duplication.

**User Commands**

This functions shows the User Commands available for the current user, accordingly with the definitions provided in xspool.conf. For each command a button is shown.

When the users pushes a button the corresponding command is activated on the server, and its execution output is displayed in lower side of the page.
10.5 Batch Interface

XSPOOL repository is feed by means of its scripting interface herewith described.

**xspooladd**

xspooladd is the command used to insert a file in the repository of XSPOOL. Usually this command is invoked by a script (JCL).

When the command is invoked, some information must be provided:

- unique identify for the source process (usually the PID number or the value of XBM_REQID, if running under XBM)
- name of the source script
- the file description
- the print class (optional)
- priority (optional)
- number of copies (optional)
- form name (optional)

XSPOOL stores the print file in its repository with name composed as follow:

```
job_name . pid . date . time . class . priority . copies . form . description
```

Once added to the repository, the file is removed from the working directory (unless -k specified).

Unless -N option is specified, xspooladd prints the file.

Some information (identifier, jobname and description) are directly extracted from the source file name, therefore it must conform to this format:

```
job_name . pid . description
```

The remaining file attributes may be provided:

- by command line options
- by environment variables
- using the default values contained in the configuration of XSPOOL (xspool.conf)

The command line options override the environment variables settings. The environment variable settings override the default configuration.

**Usage**

```
xspooladd [options] <filename>
```

**Options**

- `-N` just add (no lp spooling)
- `-a<filename>` appends the print command (xspoolpr) to the specified file, instead of printing directly (the file will contain the print commands and it could be invoked separately)
- `-F<filter>` runs the specified filter over the file before to add it to the repository
- `-k` keeps the file (does not remove the input file)
-u <username> feeds the <username> repository. Multiple users may be specified using "," (comma)
  i.e.  -u user01,user02,user99

-d <YYYYMMDD> forces the specified creation date
-t <HHMMSS> forces the specified creation time
-c <class> sets the spool class id (as defined in xspool.conf). Multiple classes may be specified using "," (comma)
  i.e.  -c C01,C02,C99

-p <priority> sets the spool priority
-n <copies> sets the desired number of copies
-f <form> sets the form id. Multiple forms may be specified using "," (comma)
  i.e.  -f F01,F02,F99 (deprecated)

-s <sysout> sets the SYSOUT parameter
-v  verbose mode
-h  shows help

Environments variables

<table>
<thead>
<tr>
<th>XSPOOLDIR</th>
<th>path of the spool directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSPOOLCONF</td>
<td>pathname of the configuration file (xspool.conf)</td>
</tr>
<tr>
<td>XSPOOL_USER</td>
<td>default user assigned to files. If not defined username is get from XBM_USER or LOGNAME.</td>
</tr>
<tr>
<td>XSPOOL_CLASS</td>
<td>default spool class id</td>
</tr>
<tr>
<td>XSPOOL_PRI</td>
<td>default priority id</td>
</tr>
<tr>
<td>XSPOOL_COPIES</td>
<td>default number of copies</td>
</tr>
<tr>
<td>XSPOOL_FORM</td>
<td>default form id</td>
</tr>
<tr>
<td>XSPOOL_SYSOUT</td>
<td>default SYSOUT parameter</td>
</tr>
<tr>
<td>XSPOOLADD_FILTER_BY_ARGUMENTS</td>
<td>if set to YES, filters are invoked by argument</td>
</tr>
<tr>
<td></td>
<td>if set to NO (default), filter are invoked by pipe</td>
</tr>
<tr>
<td>XSPOOLADD_DELETE_EMPTY_FILES_AFTER_FILTER</td>
<td>if set to YES, empty output files produced by filters are deleted</td>
</tr>
<tr>
<td>XSPOOLADDVERBOSE</td>
<td>is set to &quot;yes&quot;, verbose mode is enabled</td>
</tr>
</tbody>
</table>

xspoolpr

It is the script used to send files to printer: to do that it uses the lp system command.
xspoolpr is executed directly from xspooladd or from the web application, whenever the printing is requested

Accordingly to the settings provided in the XSPOOL configuration, xspoolpr composes the lp command to spool the file and invokes it.

Setting the environment variable XSPOOLPRVERBOSE to "yes", it displays the lp command issued.

i.e.

```
xspoolpr: lp command is 'lp -c -dlaser -n1 /home/spool/GA/U0/queque/aaa.10805.20021006.171302.laser.df.1.A.df.txt
```

Usage

```
xspoolpr [options] <filename>
```

Options

- **-N** do not print (just echo information)
- **-c<class>** sets the spool class id (defined in xspool.conf)
- **-p<priority>** sets the spool priority
- **-n<copies>** sets the number of copies
- **-f<form>** sets the form id
- **-v** verbose mode
- **-h** shows help

Environment variables

<table>
<thead>
<tr>
<th>Environment variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSPOOLDIR</td>
<td>path of the spool directory</td>
</tr>
<tr>
<td>XSPOOLCONF</td>
<td>pathname of the configuration file (xspool.conf)</td>
</tr>
<tr>
<td>XSPOOLPRVERBOSE</td>
<td>is set to &quot;yes&quot;, verbose mode is enabled</td>
</tr>
</tbody>
</table>
10.6 Spool structure

The XSPOOL repository is composed by a set of directories in which the print files are stored. At the top level of the repository, there is the root directory which path is addressed by $XSPOOLDIR environment variable. The root contains the following sub-directories:

**tmp**

This directory is mandatory and its name must be written in lower case. Usually it contains temporary files.

**conf**

This directory is optional and it can be used to contain the configuration file, unless $XSPOOLCONF environment variable is set to path the configuration file somewhere else.

**groups directories**

This type of directory is used to store the repositories of groups of users. Each group directory contains one or more user directory.

**user directories**

For each user defined in the XSPOOL configuration file (xspool.conf) there must be a directory to store its files. This directory must contain "queue" and "deleted" subdirectories.

I.e.

```
<SPPOOL_HOME>
  |
  +- spool +- conf
  |       |
  |       +- tmp
  |       |
  |       +- groupA +- user1 +- deleted
  |           |       |
  |           |       +- queue
  |           |
  |           +- user2 +- deleted
  |           |       |
  |           |       +- queue
  |           |
  |           +- user3 +- deleted
  |           |       |
  |           |       +- queue
  |           |
  |           |
  |           +- groupB +- user4 +- deleted
  |           |       |
  |           |       +- queue
  |           |
  |           +- user5 +- deleted
  |           |       |
  |           |       +- queue
  |           |
```
11 XFRAME Unified Console

This book is intended to help you to operate with XFRAME Unified Console 1.0.

XFRAME Unified Console (XUC) is included in XFRAME, and its aim is to operate as single administration desktop for all the XFRAME products.

XUC is a modular environment which permits you may manage and keep under control:
- XCICS/TS regions
- XTND standalone services
- XVSAM repositories

Directly from your PC, through its graphical interface, you may remotely monitor and administer all the instances of the product above products running in your enterprise.

XUC uses the SSH protocol to connect and exchange data with the remote system being administered, avoiding the need to install any special agent on the server side, reducing the configuration effort and the connectivity problems and insuring the adequate security required by such administrative tasks.

XUC has an integrated SFTP browser to navigate on remote file system from your PC. You can browse and edit your source files with internal XUC editor, create and manage your actions task. Actions task are and actions that you can configured to execute command on remote machine. For example you can create an action to compile your TP cobol source. In this way all activities will be managed by XUC interface.

Requirements

XUC requires side the following software

- Java web start
- Java 1.5.0 or higher
11.1 Installation

In order to setup XUC, the following components must be required:

- Java 2 Standard edition 1.5.0 (JRE or JDK 1.5) or higher
- Java Web Start

XUC is deployed in a format compatible with the Java WebStart Technology (tm).

With Java Web Start, user launches applications simply by clicking on a Web page link. If the application is not present on your computer, Java Web Start automatically downloads all necessary files. It then caches the files on local computer so the application is always ready to be relaunched anytime you want -- either from an icon on your desktop or from the browser link.

Moreover if application files are out-of-date, Java Web Start automatically provides application updating by downloading newer files.

Requirements

The client machine requires support for the Java Runtime Environment (JRE), version 1.5 or later.

Applications can be deployed from any standard Web server. In order to use Java Web Start (tm), the Web server must be configured with support for a new MIME type as explained below.

Installing on a J2EE Web Server

XFRAME is deployed with a J2EE Web Application that can be used to deploy the X4J via Java Webstart, without having to configure anything. The only thing to do is to create a context in the J2EE WebServer hosting the web application located in $XFRAMEHOME/xje/webapps/webstart.

Installing on Tomcat 4.1

To install the deployment application on Tomcat 4.1 simply add in the Tomcat webapps directory ($CATALINA_HOME/webapps by default) an XML file named as the context you are going to use (i.e. xframe.xml) containing:

```
<Context path="<CONTEXT_NAME>" docBase="<XFRAMEHOME>/xje/webapps/webstart" debug="0" reloadable="true">
    <Logger className="org.apache.catalina.logger.FileLogger"
        prefix="localhost.xframe.log."
        suffix=".txt"
        timestamp="true"
    />
</Context>
```

replacing the CONTEXT_NAME with the real context name, and XFRAMEHOME with the real XFRAME installation path ($XFRAMEHOME).

Once configured the XML file and restarted the web server simply connect to http://your.tomcat.web.server/CONTEXT_NAME/xuc.jnlp (where CONTEXT_NAME is the name specified above) to start XUC.

Installing on a Standard Web Server

To make use of Java Web Start (tm) technology with XUC simply follow this step:

- configure the Web server so that all files with the .jnlp file extension are set to the application/x-java-jnlp-file MIME type
- update the supplied JNLP files (xuc.jnlp and jh.jnlp) with the correct codebase information
- make the application accessible on the Web server by uploading application's JARs, JNLP and GIF files, at the URLs listed in the JNLP file.

- Create a link from the Web page to the JNLP file. For example:

  ```html
  <a href="xuc.jnlp">Launch XUC</a>
  ```

This first step is critical for making sure that a Web browser will launch Java Web Start when downloading the file.

A link to the Java Web Start installer should also be provided on the Web page, so users who do not already have Java Web Start installed can download and install the software.

**Further information**

For further information on Java Web Start (tm) please refer to: http://java.sun.com/products/javawebstart

**XFRAME embedded web server**

When XFRAME is installed, you may optionally let the setup procedure to configure a web server ready to support the deployment of XFRAME desktop tools, such X4J and XUC. Please refer to the XFRAME setup guide to know how to configure and use this web server.
11.2 Desktop Elements

XUC has a master container window where the user can browse and run various components. The basic components into the master XUC window container are:

- The Explorer tree
- Docking panels containing different objects
- A Bottom area reserved for the log console, normally hidden, but appearing whenever a command is started
On the left side there is the Explorer window component, where all configured connections and sessions are grouped by type or by host.

Each connection may be managed by its pop-up menu (right mouse button click on the desired element).

On the right side there is the container, that can organize a number of windows and where users may navigate window-by-window very easily. The buttons on the window tabs and on the right-side of the tabbed desktop, grants fast navigation between different objects.
Common items

All the table containers displayed in XUC have some common features, even if belonging to different modules. Of these common features is the "Filter".

At the bottom of each table, there is an area, the filter, where the some controls allow the user to filter the information shown, and therefore restrict its research only to those items needed.

User has to select which fields he wants to filter and then select the value desired. For some kind of fields the choose is driven by a list of possible values, for other fields user may enter the desired value. In this case wildcards (*) may be used.

XUC main menu

From the main menu user can create new connections, set XUC options product, set panel view, read help and show product info. The Edit menu is disabled by default and it is automatically enabled whenever the text editor is actriated.

File menu

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New connection</td>
<td>defines a new connection (XCICS/TS, XVSAM, XTND, SFTP) to a server component. Please refer to the specific module documentation</td>
</tr>
<tr>
<td>New file</td>
<td>creates a new file</td>
</tr>
<tr>
<td>Save</td>
<td>saves the file (when opened in the XUC text editor)</td>
</tr>
<tr>
<td>Save as</td>
<td>saves the file with a new name</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the current file</td>
</tr>
<tr>
<td>Revert to saved</td>
<td>Reverts the file to saved version</td>
</tr>
<tr>
<td>Options</td>
<td>Opens the XUC options dialog</td>
</tr>
<tr>
<td>Exit</td>
<td>Exits from XUC</td>
</tr>
</tbody>
</table>
Edit

This is the XUC text editor menu, making available all the edit commands. This menu contains the same items shown in the pop-up menu integrated into the text editor. [see, XUC text editor section of this guide](#)

View

This menu permits to open and close some particular windows (Master log and Code navigator) directly, and to navigate windows-by-windows (previous and next view).

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Refresh</td>
<td>Refreshes the current view</td>
</tr>
<tr>
<td>Close</td>
<td>Closes the current view</td>
</tr>
<tr>
<td>Previous view</td>
<td>Shows the previous view</td>
</tr>
<tr>
<td>Next view</td>
<td>Shows the next next view</td>
</tr>
<tr>
<td>Code navigator</td>
<td>Open code navigator see, Code navigator section <a href="#">see</a></td>
</tr>
<tr>
<td>Master log</td>
<td>Open master log see, XUC log console section <a href="#">see</a></td>
</tr>
</tbody>
</table>

Help

The help menu allows the access to the on-line help feature and to the product information dialog.
11.3 Connections

XUC communicates with the server side using the SSH protocol. All the dialogs for the configuration of a connection have an SSH section in all the modules.

In this section user must define:

- the server hostname or IP address
- the SSH port
- the authentication method (password or certificate)

Please note that, when choosing password, it is stored in a clear text file. Therefore it would be better to use an SSH certificate.
11.4 Options

XUC contains a dialog to set its options, that may be activated by the 'File → Options' menu. The settings available in the options dialogs are grouped into tabs, according to their type.

Network

This tab contains all parameters involved with networking.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH Compression</td>
<td>Sets the SSH compression of data transmitting between XUC and remote system through SSH protocol.</td>
</tr>
<tr>
<td>Auto refresher threads</td>
<td>Sets the number of threads performing refresh activities.</td>
</tr>
<tr>
<td>Auto refresh interval (msecs)</td>
<td>Sets the auto refresh interval (in milliseconds).</td>
</tr>
</tbody>
</table>
Edit

These are the settings involved with text editor.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font</td>
<td>Sets the font face and size for the editor: click on 'choose font' button to select font from those available.</td>
</tr>
<tr>
<td>Tab size</td>
<td>Sets the tab character size.</td>
</tr>
<tr>
<td>Tab mode</td>
<td>Defines the tab usage mode:</td>
</tr>
<tr>
<td></td>
<td>- language driven: the editor chooses the tab mode according to the programming language</td>
</tr>
<tr>
<td></td>
<td>- tabs: the tab character is used.</td>
</tr>
<tr>
<td></td>
<td>- spaces: space characters are put in place of tab character.</td>
</tr>
</tbody>
</table>
Appearance

These settings control the appearance of the GUI.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decorated windows</td>
<td>Window frames are decorated according to the L&amp;F</td>
</tr>
<tr>
<td>Title bar docking</td>
<td>Title bars are decorated according to the L&amp;F</td>
</tr>
<tr>
<td>theme</td>
<td></td>
</tr>
<tr>
<td>Look &amp; Feel</td>
<td>Sets the desired look and feel</td>
</tr>
</tbody>
</table>
File browsing

These are the settings involved with file browsing.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show files in SFTP tree</td>
<td>Files will be show into SFTP tree also. (usually files are visible into SFTP browser only)</td>
</tr>
<tr>
<td>Open directory browser on double click</td>
<td>User can open directory with simple double click into SFTP browser.</td>
</tr>
</tbody>
</table>
X-Server

These are the settings for the embedded X-Server.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display number</td>
<td>Set display number of the X-Server</td>
</tr>
<tr>
<td>Enabled</td>
<td>Enable or disable the X-Server</td>
</tr>
</tbody>
</table>
11.5 XCICS/TS administration module

With this module you may control as many XCICS/TS regions you want, performing activities like:

- start-up and shutdown of the region
- resource administration (i.e. terminal purging, TCP/IP services management, dataset control, etc.)
- configuration maintenance
- on-line region monitoring
- statistics and reporting

Each regions controlled is monitored in real-time: the icons on the XUC desktop shows the service status of each region, so you may know, immediately, if and where a problem occurs.

Furthermore, the activity gauges show important activity parameters of the region, like the number of terminals connected, average transaction response time and stand-by queue length and wait-time, while the activity monitors display “live” charts that track the history of these information, to let you understand the trend of the performances.

Last but not least, you may get, with a single mouse click, graphical reports and charts from the statistics collected by the XCICS/TS region, for the whole region of for a particular set of transaction codes or terminal names.
11.5.1 Configuration

**XCICS/TS session configuration:**

In order to configure XCICS/TS you can execute following steps:

1. Open File menu and select New Connection - XCICS/TS region
2. Insert information about the connection that you want create to XCICS/TS service (connection name, XCICS/TS region name and configuration file)
3. Insert SSH connection parameters, as described in Connections and click OK
The new connection will appear such as the following window:

To connect XUC to the region, double click on created connection.
### 11.5.2 Operations

By the popup-menu of the region connections, the following operations area available

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connects to the XCICS/TS region</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnects from the XCICS/TS region</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the connection</td>
</tr>
<tr>
<td>Properties</td>
<td>Visualizes the connection properties</td>
</tr>
<tr>
<td>Region Operation</td>
<td>Starts, shuts down and reconfigures the XCICS/TS region</td>
</tr>
<tr>
<td>Edit Region Configuration File</td>
<td>Edits the XCICS/TS configuration file</td>
</tr>
<tr>
<td>Monitors</td>
<td>Activates monitor windows</td>
</tr>
</tbody>
</table>
11.5.3 Control panel - Status

The main window is a control panel where there is a statistic data service area (total transactions, connected terminals, response time etc.)

At the top of area you can see the information about the region:

- version
- status
- configuration File
- network port

At the bottom of area there are the following section.

Statistics

This area display statistics information about XCICS/TS activity

- Total number of transaction
- Connected terminals
- Average time of transactions response
- Information about wait time of queue
Tunables

This area allows the online set-up of some region tunables: the trace levels and the shared pool size. For more information about this parameters refer to XCICS/TS documentation. To set parameters choose their values and click on "Apply tunables".

Reports

This area allows creation of region activity report.

The following reports are available:

- Transaction count
- Terminals usage
- Transaction elapsed time
- Transaction end-to-end time
- Transaction activated by remote systems

To see report and relative chart choose type of report and click on "Run Report". Following an example of Transaction count report.
11.5.4 Terminals

This table shows the content of the Terminal Control Table (TCT). By default only connected terminals are listed. De-select the check box at bottom to shown inactive terminals too.

For each terminal the following details are displayed.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EIBTRMID</td>
<td>This is the logical name of the terminal.</td>
</tr>
<tr>
<td>Device</td>
<td>It is the device identifier of the terminal</td>
</tr>
<tr>
<td>Netname</td>
<td>The netname of the terminal</td>
</tr>
<tr>
<td>IP address</td>
<td>The IP address of daemon service (if you use X4J terminal emulator this IP address is the same of Source IP)</td>
</tr>
<tr>
<td>Source IP</td>
<td>It is the physical network address of the terminal</td>
</tr>
<tr>
<td>EIBTASKN</td>
<td>Indicates the task number assigned to the task by XCICS.</td>
</tr>
<tr>
<td>PID</td>
<td>PID of running process</td>
</tr>
<tr>
<td>Transaction</td>
<td>Currently running transaction</td>
</tr>
<tr>
<td>Program</td>
<td>Currently running program</td>
</tr>
<tr>
<td>User</td>
<td>User owner of terminal</td>
</tr>
<tr>
<td>Connected</td>
<td>Indicates if terminal is connected</td>
</tr>
<tr>
<td>Log level</td>
<td>Indicates log level set on terminal</td>
</tr>
<tr>
<td>Debugger display</td>
<td>Indicates Debugger display associated on terminal</td>
</tr>
</tbody>
</table>

Clicking mouse right button, XUC allows functions and setting of selected terminal properties.

**Kill terminal(s)**

This function kills one or more selected terminals.

**Set log level [None - User - Full]**

Set log level of selected terminals activity

**Enable debug on local display**

Set the debugger display on the X-Server currently running on XUC.

**Enable debug on user display**

Set the debugger display on a custom display. An input window is shown and the user must insert the IP address and display number where he wants to display the output of the debugger.
Refresh

Refresh the terminal list.
11.5.5 Files

This table lists the datasets configured for the region. Each line is one file and the following details are shown.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset</td>
<td>Dataset name</td>
</tr>
<tr>
<td>Remote system</td>
<td>Remote system name (for remote files only)</td>
</tr>
<tr>
<td>Catalog</td>
<td>XVSAM catalog name of the file.</td>
</tr>
<tr>
<td>Cluster</td>
<td>XVSAM cluster name of the file.</td>
</tr>
<tr>
<td>Status</td>
<td>File status (Open - Close)</td>
</tr>
<tr>
<td>Record length</td>
<td>Record length of the cluster</td>
</tr>
<tr>
<td>Key position</td>
<td>Key offset position</td>
</tr>
<tr>
<td>Key length</td>
<td>Key size</td>
</tr>
</tbody>
</table>

For each file the following operation are available:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>Sets the selected files status to open.</td>
</tr>
<tr>
<td>Close</td>
<td>Sets the selected files status on close.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refreshes the files list.</td>
</tr>
</tbody>
</table>
### 11.5.6 Users

This table contains the information about the user attributes and access permission. The following details are shown.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Userid</td>
<td>User identifier (USERID)</td>
</tr>
<tr>
<td>Username</td>
<td>The name of the user (USERNAME)</td>
</tr>
<tr>
<td>Userclass</td>
<td>The name of class owning the user</td>
</tr>
<tr>
<td>OPID</td>
<td>Indicates Operator IDentifier associated to the user (3 bytes)</td>
</tr>
<tr>
<td>LDAP DN</td>
<td>The LDAP distinguish name (DN) for the user, if defined</td>
</tr>
</tbody>
</table>
11.5.7 Transactions

This table shows the content of the Transaction Control Table (TCT), and it shows the following information:

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transid</td>
<td>The transaction identifier</td>
</tr>
<tr>
<td>Enabled</td>
<td>Status of the transaction</td>
</tr>
<tr>
<td>Program</td>
<td>The name of the program assigned to the specified transaction code</td>
</tr>
<tr>
<td>Remote system</td>
<td>The remote system name, if defined</td>
</tr>
<tr>
<td>Remote name</td>
<td>The remote transaction name associated on remote system</td>
</tr>
<tr>
<td>Logging level</td>
<td>The logging level associated to the transaction</td>
</tr>
<tr>
<td>Debugger display</td>
<td>Indicates the debugger display coordinates associated to the transaction</td>
</tr>
</tbody>
</table>

The following operations may be performed on transactions:

**Set status [Enabled - Disabled]**
Enables/Disables transactions.

**Set log level [None - User - Full]**
Sets the log level of selected transactions.

**Enable debug on local display**
Sets the debugger display to the X-Server display on XUC.

**Enable debug on user display**
Sets the debugger display to a custom X-Server.

**Refresh**
Refresh of transactions list.
11.5.8 Programs

This table shows the content of the PPT (Program Processing Table), and in particular

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Program name</td>
</tr>
<tr>
<td>Language</td>
<td>Programming language</td>
</tr>
<tr>
<td>Remote system</td>
<td>Indicates remote system name</td>
</tr>
<tr>
<td>Available</td>
<td>Indicates if the program is available</td>
</tr>
<tr>
<td>In shared lib</td>
<td>Indicates if the program is in a shared library</td>
</tr>
<tr>
<td>Path</td>
<td>Indicates the physical path of the program</td>
</tr>
</tbody>
</table>
11.5.9 Transient data

The table shows the content of the Destination Control Table (DCT). Each entry describes a transient data

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The Transient Data Queue name</td>
</tr>
<tr>
<td>Remote system</td>
<td>Indicates the remote system name, if defined</td>
</tr>
<tr>
<td>Items</td>
<td>The number of items stored in the TD</td>
</tr>
<tr>
<td>Trigger</td>
<td>The queue's trigger level</td>
</tr>
<tr>
<td>Transaction</td>
<td>The name of the transaction bind to the trigger.</td>
</tr>
</tbody>
</table>

The following operations may be performed

**Set trigger level**

Sets trigger level of the transient data
### 11.5.1 Connection

This table shows the CCT, Connection Control Table, and displays the following information:

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSID</td>
<td>System identifier</td>
</tr>
<tr>
<td>Partner LU</td>
<td>LU name of the system</td>
</tr>
<tr>
<td>OS Type</td>
<td>OS type: EBCDIC or ASCII</td>
</tr>
<tr>
<td>In service</td>
<td>Indicates if connection is in service</td>
</tr>
<tr>
<td>Acquired</td>
<td>Specifies if connection is acquired</td>
</tr>
<tr>
<td>XLN</td>
<td>eXchange Log Name status</td>
</tr>
</tbody>
</table>

From within this window, user may start the acquire process of the partner.
11.5.1 Mapsets

This table shows the MPS, and it displays information about the mapsets:

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapset</td>
<td>name of the mapset</td>
</tr>
<tr>
<td>Maps number</td>
<td>Number of maps contained</td>
</tr>
<tr>
<td>New copy</td>
<td>New copy status</td>
</tr>
<tr>
<td>Path</td>
<td>Full path to mapset module</td>
</tr>
</tbody>
</table>

Clicking mouse right button you can request to reload selected mapset module to XCICS/TS.
### 11.5.1 Journals

This table shows the definitions for log files used by journaling sub-system (JCT).

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>This is the 4 bytes symbolic identifier used in the EXEC CICS WRITE JOURNALNAME commands to reference the journal.</td>
</tr>
<tr>
<td>Number</td>
<td>This is the numeric identifier for the journal (1-99) used in the EXEC CICS WRITE JOURNALNUM and EXEC CICS JOURNAL commands</td>
</tr>
<tr>
<td>Type</td>
<td>The type of the journal file.</td>
</tr>
<tr>
<td>File name</td>
<td>The full path name to the journal log file</td>
</tr>
</tbody>
</table>
11.5.1 UriMap

This tables shows information about URIMAP definitions.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the URIMAP</td>
</tr>
<tr>
<td>Enabled</td>
<td>Indicates whether URIMAP is enabled</td>
</tr>
<tr>
<td>Usage</td>
<td>Usage of the URIMAP (server or client usage)</td>
</tr>
<tr>
<td>Scheme</td>
<td>Scheme of the URIMAP</td>
</tr>
<tr>
<td>Type</td>
<td>Type of the URIMAP</td>
</tr>
<tr>
<td>TCPService</td>
<td>Indicates the tcpservice name bind</td>
</tr>
<tr>
<td>Host</td>
<td>Indicates the host filter</td>
</tr>
<tr>
<td>Path</td>
<td>Indicate the path of the URL matching the URIMAP</td>
</tr>
</tbody>
</table>

The following operations are admitted on URIMAPs

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables the selected URIMAPs.</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the selected URIMAPs.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refresh the URIMAPs list.</td>
</tr>
</tbody>
</table>
### 11.5.1 Documents template

This tables lists information concerning the document templates defined for the region.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the document template</td>
</tr>
<tr>
<td>Type</td>
<td>Template type</td>
</tr>
<tr>
<td>Resource</td>
<td>Physical path of document template resource</td>
</tr>
</tbody>
</table>
# 11.5.1 TCP/IP services

This table contain information about the TCP/IP services.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the TCP/IP service</td>
</tr>
<tr>
<td>Port</td>
<td>TCP/IP Port of the service</td>
</tr>
<tr>
<td>Type</td>
<td>Type of the service</td>
</tr>
<tr>
<td>PID</td>
<td>PID of the process handling the service</td>
</tr>
<tr>
<td>Status</td>
<td>Indicates the service status</td>
</tr>
</tbody>
</table>

The following operation may be performed:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables the selected TCP/IP services.</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the selected TCP/IP services.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refreshes the TCP/IP services list.</td>
</tr>
</tbody>
</table>
11.5.1 TN3270 services

The TN3270 service is a particular type of background process, which serve TN3270 connections over TCP/IP. Selecting 'TN3270' section, XUC shows 3270 services defined for the region.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The TN3270 service name</td>
</tr>
<tr>
<td>Port</td>
<td>The TCP/IP port where the TN3270 service listens</td>
</tr>
<tr>
<td>PID</td>
<td>The PID of the process managing the service</td>
</tr>
<tr>
<td>Status</td>
<td>Indicates the status of the service</td>
</tr>
</tbody>
</table>

The following operations are admitted:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables the selected TN3270 service.</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the selected TN3270 service.</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refreshes the TN3270 services list.</td>
</tr>
</tbody>
</table>
11.5.1 MQ trigger monitor

This table shows the status of the MQ trigger monitors.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name of the MQ trigger</td>
</tr>
<tr>
<td>Messages processed</td>
<td>Indicates how many messages have been processed by the monitor instance</td>
</tr>
<tr>
<td>Total messages</td>
<td>Indicates how many messages have been globally processed by the monitor</td>
</tr>
<tr>
<td>processed</td>
<td>instances</td>
</tr>
<tr>
<td>PID</td>
<td>The PID of the process handling the monitoring service</td>
</tr>
<tr>
<td>Status</td>
<td>Indicates the status of the MQ trigger monitor</td>
</tr>
</tbody>
</table>

The following operations are admitted.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Enables the selected MQ trigger monitor</td>
</tr>
<tr>
<td>Disable</td>
<td>Disables the selected MQ trigger monitor</td>
</tr>
<tr>
<td>Refresh</td>
<td>Refreshes the MQ trigger list.</td>
</tr>
</tbody>
</table>
11.5.1 User shared memory

This table shows a map of the allocated shared memory reserved for user calls (GETMAIN). The information are shown per memory page.

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Indicates whether the page is busy</td>
</tr>
<tr>
<td>UID</td>
<td>Slot unique identifier</td>
</tr>
<tr>
<td>PID</td>
<td>The PID holding the page(s)</td>
</tr>
<tr>
<td>Task ID</td>
<td>The task holding the page(s)</td>
</tr>
<tr>
<td>Shared</td>
<td>Indicates whether memory is allocated with the SHARED option</td>
</tr>
</tbody>
</table>
11.5.1 Processes

This table shows the processes belonging to the region:

<table>
<thead>
<tr>
<th>Column field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>Indicates if process is main or child.</td>
</tr>
<tr>
<td>PID</td>
<td>the PID of the process</td>
</tr>
<tr>
<td>Size</td>
<td>Size of the process</td>
</tr>
<tr>
<td>Execution time</td>
<td>Execution time</td>
</tr>
<tr>
<td>Role</td>
<td>Process role (core, background, etc...)</td>
</tr>
</tbody>
</table>
11.5.2 Console log

Selecting 'Console log' section, XUC shows last log lines of XCICS/TS output console trace.
11.5.2 Monitor utility

XUC allows to user two different monitors type to check in real time the region activities. To launch monitor utility you must, first of all, connect XUC to XCICS/TS active service. After this you can show the pop-up menu as in the following screenshot and click on monitors tag menu.

You can choose between 'Response time' and 'Transaction number' monitoring windows.

- 'Response time' permits to draw XCICS/TS response time average. You can read the average time in milliseconds, and analyze XCICS/TS graphic.

- 'Transactions number' permits to draw XCICS/TS transaction number running. You can read this number on the left graphic bar and analyze XCICS/TS graphic.

All graphics can be saved as image to print, store or other uses.
Response time

This monitor shows a chart tracking the average transaction response time.

Transaction number

This monitor tracks the number of parallel running transaction number.
You can set some parameters of graphic visualization and you can also save as image file (png) or print graphic on the printer. You can fire this function from pop-up menu.

In previous example user perform a ‘zoom in’ function.
11.6 XTND administration module

XTND purpose is to convert native XCICS protocol into tn3270 protocol. The XTND daemon can be run embedded in the XCICS region or as standalone service.

When XTND is run embedded in the region, it only serves the region itself. When it runs as standalone service it can be run on the same system running the XCICS application or on another system. Each instance of can handle one or more XCICS application.

The XTND module allows you to view and manage the terminals connected to a standalone XTND services, performing operations like:

- start-up and shutdown of the service
- configuration maintenance
- monitoring activities
11.6.1 Configuration

**XTND session configuration:**

In order to configure XTND session you can execute following steps:

1. Open File menu and select 'New Connection à XTND standalone service'
2. Insert information about the connection that you want create to XTND service (connection name and configuration file)

![XTND standalone service connection](image)

3. Insert SSH connection parameters as described in [Connections](#) and click OK.
Once connected, the following window is shown:

For XTND connections, the following operations are admitted:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connects to the XTND service</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnects from the XTND service</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the configured session</td>
</tr>
<tr>
<td>Properties</td>
<td>Visualizes the session properties</td>
</tr>
<tr>
<td>service Operation</td>
<td>Start and shutdown the XTND service</td>
</tr>
<tr>
<td>Edit Configuration File</td>
<td>Edits the XTND configuration file</td>
</tr>
</tbody>
</table>
11.6.2 XTND - Status

In this window, you can check the XTND status and others parameters.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Indicates the status (active or not) of XTND service</td>
</tr>
<tr>
<td>Configuration File</td>
<td>Indicates the path and file name of xnd configuration file</td>
</tr>
<tr>
<td>Host Name</td>
<td>The host name (IP or name) of the machine where XTND service runs</td>
</tr>
<tr>
<td>Service address</td>
<td>IP address and port where XTND service listens</td>
</tr>
<tr>
<td>XTND Version</td>
<td>XTND service version</td>
</tr>
<tr>
<td>Connected Terminals</td>
<td>Indicates how many connected terminals are currently connected</td>
</tr>
</tbody>
</table>
11.6.3 Connected terminals

In the "Connected terminals" view, XUC shows the information about those connections and offers the capability to disconnect them.
11.7 XVSAM administration module

XVSAM (XFRAME Virtual Storage Access Method) is a file access method to emulate the features of the IBM VSAM on UNIX/Linux systems.

Using XVSAM, you may organize and access records in a file in physical sequence (the sequential order that they were entered), logical sequence using a key, or by the This module let you to operate with existing XVSAM repositories. Through a GUI driven interface you may perform operation like:

- XVSAM files and indexes creation, deletion and erasure
- import/export of XVSAM files
- complex REPRO-like operation
11.7.1 Configuration

**XVSAM session configuration:**

In order to configure XVSAM session you can execute following steps:

1. Open File menu and select 'New Connection → New XVSAM repository'
2. Insert information about the repository that you want link (connection name and XVSAM path)

3. Insert SSH connection parameters as described in Connections and click OK.
In order to connect XUC to XVSAM repository, double click on the created connection and then double click on the XVSAM repository. XUC connects XVSAM repository and visualize all VSAM catalog. To view VSAM files on the list double click on desired catalog. The following windows appears.

The following operations are admitted on the connection:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connects to the XVSAM repository</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnects from the XVSAM repository</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the connection</td>
</tr>
<tr>
<td>Properties</td>
<td>Opens the XVSAM repository properties dialog</td>
</tr>
<tr>
<td>Create new</td>
<td>Create new Cluster or alternate index in the repository (see Creating XVSAM entries)</td>
</tr>
</tbody>
</table>
11.7.2 Creating XVSAM entries

With this module you can create XVSAM entries, such as a cluster or an alternate index.

Creating an XVSAM Cluster

To create an XVSAM cluster you must connect XUC to VSAM repository and then select 'create new → Cluster' from popup menu on XVSAM repository into XUC explorer.

The following dialog appears:

First of all you must enter the new name of your new cluster and catalog name. After this you have four areas to set new cluster parameters.

In the "General" tab you must insert logical name, path, reuse flag, and file type.
In the "Record length" tab you must set record type and minimum length and maximum length of record.

In the "Keys" section, you may define the file key(s): click on the 'Add' button to create new key.

Insert key offset and length, and click 'Ok' button to create new key.
The key may be edited or removed using the other commands on the dialog.
Finally, in the "Storage" tab you can set the storage type of new cluster.
Creating an XVSAM alternate index

Selecting ‘Create new → Alternate index’ from popup menu on XVSAM repository into XUC explorer, the following dialog appears.

First of all you must enter the name of the alternate index and the catalog name. Then you have to define the primary cluster:

and then, the keys for the index
11.7.3 Basic VSAM functions

From VSAM file list you can execute some basic functions on your file. To fire this functions select desired file from the list and show pop-up menu.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create new</td>
<td>To create new cluster or alternate index (see. Create VSAM object section)</td>
</tr>
<tr>
<td>Erase</td>
<td>Erases the selected XVSAM file(s)</td>
</tr>
<tr>
<td>Delete</td>
<td>Deletes the selected XVSAM file(s)</td>
</tr>
<tr>
<td>Import</td>
<td>Imports a flat file on the selected XVSAM file. The file can be either binary or ascii (line sequential).</td>
</tr>
</tbody>
</table>

Now you can click on desired function to execute command.
Choose your file, set its format and then click on 'Ok' button to import it.

**Export**

Exports the selected XVSAM file on a flat file, it ascii or binary format.

Choose the path of the file to be created, set its format and then click on 'Ok' button to export it.
Repro

Executes the REPRO function on the selected XVSAM file. For this operation you must define: input, output and parameters for the REPRO.

In the "Input" section, you define the characteristics of the input file.

In the "Output" section, you define the characteristics of the output file.
Finally, you may set parameters that affect the REPRO behaviour and the content of generated file.

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replace</td>
<td>□</td>
<td></td>
</tr>
<tr>
<td>Skip</td>
<td>□</td>
<td>Number of record to skip</td>
</tr>
<tr>
<td>Count</td>
<td>□</td>
<td>Number of output record</td>
</tr>
<tr>
<td>Init</td>
<td>□</td>
<td>Init value</td>
</tr>
<tr>
<td>From-key</td>
<td>□</td>
<td>to-Key</td>
</tr>
</tbody>
</table>

□ Hex
11.8 IDE module (SFTP Connection)

Among with the administrative modules, XUC contains the IDE module, which offers many features oriented to the programming task. In fact, using this module, you have to possibility to:

- navigate in the directory structure of the file system
- edit source code
- activate every kind of task on the server side (i.e. program compiling, debugger activation, etc.)

The editor is simple but powerful: it has all the features of any standard GUI editor, plus some feature oriented to the traditional code programmers, like:

- syntax highlight for COBOL, C, perl and shell scripting
- code navigator for COBOL and shell scripting
- special edit features for COBOL and shell scripting

The module contains the "Action manager", a special feature that allows to create user defined tasks to perform by the server side, to bind those tasks to a particular files types and to automatically retrieve their output.

For example you could create a task that execute the compiler and associate the task to the files in the directory $HOME/src/app1 and $HOME/src/app2 with extension .cbl, and that opens the file listing <filename>.lst when the compiler returns a non-zero return code. You will be able to activate this action will directly opening the popup menu on the file or using a key combination (i.e. CTRL+1).

With these features the programmers will be perform faster maintenance of the source code in complete IDE. In particular, those reluctant mainframe users will have the possibility to avoid to operate directly under the UNIX/Linux shell or to learn the VI editor as well as other new tools, therefore generating a reduction of the learning curve of the new environment.
11.8.1 Configuration

**SFTP session configuration:**

In order to configure SFTP session you can execute following steps:

1. Open File menu and select 'New Connection → New SFTP Connection'
2. Insert information about the connection that you want create to SFTP connection (connection name and initial path)
3. Insert SSH connection parameters as described in Connections and click OK.

The following operations are admitted on the IDE connection:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>Connects to the SFTP service</td>
</tr>
<tr>
<td>Disconnect</td>
<td>Disconnects from the SFTP service</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the configured connection</td>
</tr>
<tr>
<td>Properties</td>
<td>Visualizes the connection properties</td>
</tr>
<tr>
<td>Manage Action</td>
<td>Manage and configure actions (see Manage actions section)</td>
</tr>
</tbody>
</table>
11.8.2 The file browser

With this module you can browse the remote file systems via SFTP. You can browse it clicking on the desired directory name either on the left side tree and on the right side list. The Home and Up buttons allows to move respectively to the Home directory or up by one level.

You may select one or more elements, and then you can click with right mouse button to visualize pop-up menu, and choose desired function.

Add to favorite

Add file or directory to the user’s favorites list
**New file**

Creates a new file

![Image of New file dialog]

**New directory**

Creates a new directory

![Image of New directory dialog]

**Delete**

Deletes the selected file(s) or directory(ies).

![Image of Delete dialog]
Move/Copy file

Move or Copy the selected file(s)

Choose file

```
Permissions  Size  Owner  Group  Filename
---  ------  -----  ------  -------
-rw-rw-rw-  825    galletti  devel  ASSICURA
-rw-rw-rw-  62460  galletti  devel  MBCANDP.pre
-rw-rw-rw-  102035  galletti  devel  MOCTLGP.pre
-rw-rw-rw-  28510  galletti  devel  MOCTLUP.pre
-rw-rw-rw-  993    galletti  devel  PROVA.pre
-rw-rw-rw-  3952    galletti  devel  QUALF.pre
-rw-rw-rw-  1049    galletti  devel  QUERY.SQL
-rw-rw-rw-   4749    galletti  devel  STRINGA.pre
-rw-rw-rw-   7859    galletti  devel  TEST1.pre
-rw-rw-rw-  14504    galletti  devel  TESTELEM.pre
-rw-rw-rw-   2966    galletti  devel  cambia.pre
-rw-rw-rw-   1106    galletti  devel  cisco.pre
-rw-rw-rw-   1129    galletti  devel  cisco2.pre
-rw-rw-rw-   1790    galletti  devel  inserici0X.pre
-rw-rw-rw-   2692    galletti  devel  msha.pre
-rw-rw-rw-   3201    galletti  devel  modifica.pre
-rw-rw-rw-   1460    galletti  devel  stefano.pre
-rw-rw-rw-   256    galletti  devel  xcob.log

File name: PROVA.pre
```

**Ok**  **Cancel**
Rename file
Renames the selected file(s)

Search file (grep)
Executes a grep command to search a regular expression on one or more files (an input box will appear to insert grep parameters, press confirm to execute command)
You can read grep output listing on XUC log console (see XUC log console)
Actions

Lists and activates the actions available for the selected file(s). (see, Action manager)
11.8.3 The Editor

XUC has an internal text editor. The editor is simple but powerful: it has all the features of any standard GUI editor, plus some features oriented to the traditional code programmers, like:

- syntax highlight for COBOL, C, perl and shell scripting
- code navigator for COBOL and shell scripting
- special edit features for COBOL and shell scripting

To open a file with XUC text editor, you can click from FTP file browser on your desired file.
The text editor appears.

On the bottom text editor dialog you can read some information about opened file. Such as file type (Cobol, C, Perl, Generc text, etc.), File format (Unix or DOS) and Insertion key method (Insert or overwrite).

On the left gray bar there is a line progress number, and in the right bottom corner there is cursor position (row, column).

On the top bar you there is file name.
Many functions of the editor may be activated from the pop-up menu or from the edit menu.

The following operations are admitted:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undo Typing</td>
<td>Undo function</td>
</tr>
<tr>
<td>Cut</td>
<td>Cut selected text</td>
</tr>
<tr>
<td>Copy</td>
<td>Copy selected text</td>
</tr>
<tr>
<td>Paste</td>
<td>Paste selected text</td>
</tr>
<tr>
<td>Delete lines</td>
<td>Delete selected lines</td>
</tr>
<tr>
<td>Convert Wrap To CR/LFs</td>
<td>Convert wrap to formatted lines (CR/LF)</td>
</tr>
<tr>
<td>Convert CR/LFs To Wrap</td>
<td>Convert formatted lines to single wrap line</td>
</tr>
<tr>
<td>Show spaces/tab</td>
<td>Not yet implemented</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>Tab To Spaces</td>
<td>Convert tab to spaces</td>
</tr>
<tr>
<td>Spaces To Tab</td>
<td>Convert spaces to tab (only leading)</td>
</tr>
<tr>
<td>(leading)</td>
<td></td>
</tr>
<tr>
<td>Space To Tab (all)</td>
<td>Convert all spaces to tab</td>
</tr>
<tr>
<td>To Lower Case</td>
<td>Convert text to lower case</td>
</tr>
<tr>
<td>To Upper Case</td>
<td>Convert text to upper case</td>
</tr>
<tr>
<td>DOS to UNIX</td>
<td>Convert file from DOS to Unix format</td>
</tr>
<tr>
<td>UNIX To DOS</td>
<td>Convert file from Unix to DOS format</td>
</tr>
<tr>
<td>Duplicate lines</td>
<td>Duplicate selected lines</td>
</tr>
<tr>
<td>Remark line(s)</td>
<td>Remark selected line(s)</td>
</tr>
<tr>
<td>Find/Replace</td>
<td>Find and replace functions on text</td>
</tr>
<tr>
<td>Find next</td>
<td>Find next pattern on text</td>
</tr>
<tr>
<td>Find previous</td>
<td>Find previous pattern on text</td>
</tr>
<tr>
<td>Code navigator</td>
<td>Open code navigator (see Code Navigator section)</td>
</tr>
<tr>
<td>Actions</td>
<td>Execute action (see Action manager section)</td>
</tr>
</tbody>
</table>
11.8.4 Console log

The Console log is a structured interactive list of output listing action execution. Each execution has a single line that contains the command statement and its return code. To expand detailed information, click on it.

Each sub node of the first line contains a detailed information of execution command. You can click on these lines to open the file referred into the editor.
11.8.5 Code Navigator

Code navigator is a docking window that works with text editor. In this window you may see the main sections of the source, its includes, its functions and syntax errors detected by XUC. To open code navigator show pop-up menu from text editor and click on Code Navigator item.
The "Outline" Tab shows the outline of the programs (functions, sections, data items, etc.) according to the programming language. The "Includes" Tab shows the files included in the source (copybook, headers, etc.). The "Errors" tab shows the eventual syntax error detected by the XUC parser.

Clicking on an item in the code navigator, the editor automatically locates the cursor in the source code. In the following example user clicks on specific program section in 'Outline' code navigator tab.
11.8.6 Action manager

This module allows to create user defined tasks to be performed by the server side. These tasks, called "Actions", may be bind to a particular files types and to automatically retrieve their output.

To open Action manager dialog property open pop-up menu clicking on SFTP configured connection, and click on 'Manage actions':

The following operations are available:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Adds a new action (see, Adding an action)</td>
</tr>
<tr>
<td>Remove</td>
<td>Removes the selected action</td>
</tr>
<tr>
<td>Edit</td>
<td>Edits the selected action</td>
</tr>
<tr>
<td>Presets</td>
<td>Choose action from the presets list</td>
</tr>
</tbody>
</table>
### 11.8.6. Adding an action

To define/create a new action, you must set some properties in the action editor.

First of all you must insert the name of the new action and its shell command.

![Action editor](image)

To define shell command you can use any UNIX/Linux command line program or scripts, and you can include in the command either environment variables and the special XUC variables, listed below:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%h</td>
<td>HOME directory of the user</td>
</tr>
<tr>
<td>%x</td>
<td>XFRAMEHOME directory</td>
</tr>
<tr>
<td>%F</td>
<td>Full path of the target file (i.e. $HOME/src/template.pre)</td>
</tr>
<tr>
<td>%n</td>
<td>Name of the target file (i.e. template.pre)</td>
</tr>
<tr>
<td>%N</td>
<td>Name of the target file with no extension (i.e. template)</td>
</tr>
<tr>
<td>%d</td>
<td>Path of the target file (i.e. $HOME/src)</td>
</tr>
<tr>
<td>%P[0-9]</td>
<td>User parameters, from 0 to 9 (i.e. %P0 or %P2...)</td>
</tr>
</tbody>
</table>
Then you have to define parameters that affect the execution:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save file before execute</td>
<td>If you want to launch an action from XUC editor on an edited source and save by default before execution.</td>
</tr>
<tr>
<td>SSH</td>
<td>Defined action will be executed as an SSH exec command.</td>
</tr>
<tr>
<td>SSH : Shell</td>
<td>Defined action will be executed as a SSH shell command.</td>
</tr>
</tbody>
</table>

The command availability may be restricted to special file type and/or files located in a particular directory by defining one or more regular expression patterns for the file type (i.e., "A.*\.cbl", that means: all files beginning with A and with suffix .cbl) and for its path.
The action may be bind to a special key binding to activate it directly from the keyboard:

Finally, the output of the command may be managed, redirecting it on the XUC Console log or to a panel (a dialog), and to define a special action at command end.

For example you can define an action that compile TP cobol source (xcob -sua %n) and open file %N.errors if return code is not zero. Following the screenshot example:
12 Programmer's Reference

This book contains a reference for XFRAME programmers. It contains reference information needed to develop, compile and debug COBOL, C, PL/I, and Java application programs using XFRAME platform APIs. The book is intended primarily for use by application programmers, but will also be useful to system programmers and systems analysts.
12.1 Utilities

12.1.1 xcob

The compiling of an xframe-targeted COBOL source (TP or batch) follows normally the same steps like on the original mainframe environment, in dependence with the program's typology and language. It comprehends:

- source normalization
- SQL precompiling
- XCICS precompiling
- COBOL compiling

In order to give an user-friendly interface to all compiling processes, the "xcob" script is supplied.

SYNOPSIS

```
xcob [options] <source>
```

USE

xcob is used to compile both TP and batch COBOL programs. The filename specified as source has to be suffixed with ".pre", and all files generated have the base name of the source.

<source> Is the COBOL source to be compiled.

Options

- `-b` handles programs as batch. By default xcob handles programs as TP.
- `-x` handles programs as TP. By default xcob handles programs as TP.
- `-m` handles programs as an IMS online program.
- `-i` generates .INT code
- `-u` generates .GNT code
- `-c` generates a link-editable object (.o)
- `-z` generates a .o object to build a shared library
- `-a` generate debug information
- `-o<path>` output to specified directory
- `-U<cmd>` specifies a user exit filter
- `-k` activates BLL handling
- `-E<ext>` handles input files is with specified extention
- `-s` preprocess for E/SQL
- `-R(oracle|sybase|db2|odbc)` specifies RDBMS: oracle, sybase, db2 or odbc
- `-I<path>` alternative path for SQL include
- `-C<path>` alternative path for COBOL copybooks
- `-E` xprecob nosevere mode: precompiler doesn't stops on error.
- `-L` let the COBOL compiler generating a list file.
-x let the COBOL compiler to generate a cross-reference file.
-v produces verbose output.
-h produces command line help
-l (lowercase L) activity is logged ob 'xcob.log'
-A forces ACUCOBOL compiler invocation
-M forces Microfocus compiler invocation
-D debug: all itemediate files are left in $PWD
-O<options> specifies one or more of the following options:
  HOST2UNIX process source with 'host2unix'
  MFCOPY uses MF copy/include preprocessor
  ACUCOPY use ACU copy/include preprocessor
  NOMFCOPY uses xframe internal copy/include preprocessors
  NOACUCOPY uses xframe internal copy/include preprocessors
  MFCOBOL enables MF COBOL dialect
  OSVS enables OSVS dialect (default)
  VSC2 enables VSC2 dialect
  COBSQL uses COBSQL oracle interface
  NOCOBSQL uses SQL precompiler directly
  NOCICS compile a TP program without EXEC CICS (CALLs)
  ANIM same as -a
  NOANIM do not generate debug information
  COBOL1 source file contains COBOL 1 limitations
  SQL same as -s
  LOG same as -l
  ORACLE same as -R oracle
  DB2 same as -R db2
  ODBC same as -R odbc
  SYBASE same as -R sybase
  SQLCA includes SQLCA
  CVDA enables precompiler to handle CVDA values (DFHRESP)
  POOLDIR includes XCICS POOLDIR addressing
  NOLINK no CALL to LINK conversion performed
  TRACE enables script trace
  NOERROR manage COBOL errors (non severe) as warnings
  DLI PRE processs source with d1 precompiler 'dl1pre.pl'
  INTGNT same as -u -a
AUTOCA

enables precompiler automatic COMMAREA definition

NORETURN

inhibits RETURN command

MAKESYN

forces COBSQL convert all COMP host variables to COMP-5 host variables
(default on Linux)

NOMAKESYN

forces COBSQL to do no conversion of variables or host variables from COMP to
COMP-5

ALTSPLIT

enables alternative parsing routine in precompiler

xcob produces the following files:

<sourcename>.errors List of errors caught during (pre)compiling phases
<sourcename>.lst List file if required (-L)

Moreover it generates debugable and non debugable code in fixed directories receiving this kind of coding.

RETURN CODES

A return code of 0 (zero) is returned if compiling succeeded. Non zero if something went wrong.

CONFIGURATION

xcob may be configured both using environment variables and a configuration file.

The enviroment

Some environment variables should be previously set to let xcob work properly:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCOBCONF</td>
<td>specifies an alternate configuration file (default is $HOME/etc/xcob.conf)</td>
</tr>
<tr>
<td>XCOB_INT_DEST</td>
<td>specifies output destination for executable code with debug symbols (-i option)</td>
</tr>
<tr>
<td>XCOB_GNT_DEST</td>
<td>specifies output destination for executable code without debug symbols (-u option)</td>
</tr>
<tr>
<td>XCOB_OBJ_DEST</td>
<td>specifies output destination for .o files (-c option)</td>
</tr>
<tr>
<td>XCOB TP COBOPT[1-9]</td>
<td>XCOB_TP_COBOPT1 up to XCOB_TP_COBOPT9 are used to pass compiler directives (COBOPTs) to the Microfocus COBOL compiler, while compiling TP programs. Please refer to your MicroFocus COBOL documentation.</td>
</tr>
<tr>
<td>XCOB BATCH COBOPT[1-9]</td>
<td>XCOB_BATCH_COBOPT1 up to XCOB_BATCH_COBOPT9 are used to pass compiler directives (COBOPTs) to the MicroFocus COBOL compiler, while compiling batch programs. Please refer to your MicroFocus COBOL documentation.</td>
</tr>
<tr>
<td>XCOB TP CBLFLAGS</td>
<td>ACUCOBOL compiler options for TP compiling</td>
</tr>
<tr>
<td>XCOB BATCH CBLFLAGS</td>
<td>ACUCOBOL compiler options for BATCH compiling</td>
</tr>
<tr>
<td>XCOB CBLFLAGS</td>
<td>common ACUCOBOL compiler option for both TP and batch compiling</td>
</tr>
</tbody>
</table>
### The configuration file

The environment variable listed above may be contained in a file named `xcob.conf` which is loaded at XCOB startup. In this phase, xcob looks for the `xcob.conf` in present working directory and then in $HOME/etc. An alternative file may be specified setting the `XCOBCONF` environment with the full path to the desired configuration file.

#### Sample configuration file 1

```bash
# xcob configuration file
# compile with Microfocus COBOL and Oracle
export COBOPY=/home/cpy;$XORPHOME/cpy;$XORSHOME/cpy;$XOBHOME/cpy;$XORPHOME/$HOME/cpy
export COBOPY=/home/cpy
export XOB_LD=/home/objs/ld
export XOB_RDI=/home/objs/rdi
export XOB_TP_COBLOPT1="remove=MESSAGE remove=TEXT remove=ROLLBACK remove=CURSOR RDMFILE=EZIPFAIL"
export XOB_TP_COBLOPT2="perform=type=csv IBMCOMP warning=3 defaultbyte=0 nocheck constant ALPHA=0 FREQ=0 ALPHASTR=0"
export XOB_TP_COBLOPT3="COMP NOTRUNC SIZE_ZERO"
export XOB_BATCH_COBLOPT1="perform=type=csv IBMCOMP warning=3 ALPHASTR=0"
export XOB_BATCH_COBLOPT2="remove=COL remove=DEPAUL remove=CURSOR remove=ERROR remove=ERROR "
export XOB_BATCH_COBLOPT3="COMP NOTRUNC SIZE_ZERO COPYEXT=cpy" "
export XOB_USEREXIT=""
export XOB_RDBMS="oracle"
export XOB_SQLINCLUDE=$HOME/cpy
```
Sample configuration file 2

```bash
# # x cob configuration file # compile with ACUCOBOL and IBM DB2 UDB
# export COBPATH=/home/cpy; $XADMHOME/src $XCOBSSHOME/cobcopy
export XCOB_INIT_BEST=/home/objs/int
export XCOB_GNT_BEST=/home/objs/gnt
export XCOB_USREXIT=
export XCOB_RDSID="db2"
export XCOB_SQLINCLUDE=/home/cpy
export XCOB_DDID="user1/mypass@HTDB2"
export XCOB_CBLFLAGS="-Dz -Dv=0 -D7 -Rw ERROR -Rw OVERFLOW -Rw POS"
```

**COBOL Compilers options**

To get best results while migrating from IBM COBOL, we suggest to use the following options for the COBOL compilers:

**Microfocus COBOL**

- IBMCOMP
- NOTRUNC
- DEFAULTBYTE=0

**ACUCOBOL**

- -Dz
- -Dv=0

Please refer to the compiler specific docs for further information.

**Working with DB**

**Oracle DB**

When Oracle precompiling is required, x cob invokes procob. With Microfocus compiler procob call may be embedded in the compiling process (default for MF) or called directly. With ACUCOBOL procob is always called directly.

In the first case options are passed through the file "cobsqldir" that must be placed in the directory or the source being compiled. To get further information on cobsqldir please refer to Microfocus Server Express docs.

Sample cobsqldir

```sql
csql=ora8
sqldebug
endc
varchar=yes
declare section=no
userid=xbm/xbm@htora1
```
When procob is invoked directly options are passed through the environment XCOB_SQLOPT and Oracle userid is specified through XCOB_DBCID.

I.e.:

```
export XCOB_DBCID=pli2cob/pli2cob@dbh1
export XCOB_SQLOPT="ireclen=132 sqlcheck=syntax"
```

**IBM DB2 DB**

When DB2 precompiling is required, xcob invokes db2 preparation.

With Microfocus compiler, DB2 precompiler is invoked only through the compiler. With ACUCOBOL DB2 precompiler is invoked directly.

For Microfocus compiler, options and userid are passed through the environment XCOB_SQLOPT in the microfocus format. To get further information on DB2 precompiling through cob command please refer to Microfocus Server Express docs.

For ACUCOBOL compiler the options to DB2 precompiler are always passed through the environment XCOB_SQLOPT, but with the format specified on the IBM DB2 documentation. Userid and password must be specified using XCOB_DBCID.

I.e.:

```
# for Microfocus
export XCOB_SQLOPT="db=SAMPLE pass=db2inst2.db2inst2 bindfile "

# for ACUCOBOL
export XCOB_DBCID=user1/mypass@udbdb2
export XCOB_SQLOPT=""
```

**ODBC Data Sources**

When ODBC precompiling is required, xcob instructs the compiler to process SQL through ODBC.

To get further information on ODBC precompiling through cob please refer to Microfocus Server Express E/SQL or ACUCOBOL ACU/SQL docs.

Additional options may be passed through the environment XCOB_SQLOPT.

I.e.:

```
export XCOB_SQLOPT="dbman=odbc autocommit odbctrace=always"
```

**Example**

```
# xcob -s PGMS010
xcob PGMS010: XPRE precompiler started
xcob PGMS010: Precompiled successfully.
xcob PGMS010: Oracle E/SQL precompilation started
xcob PGMS010: Oracle precompilation successfully
```
The compiling was successful and intermediate (.int, .idy, .cbl) files have been generated and located in the correct path.
12.1.2 xpli

The compiling of an xframe-targeted PL/I source (TP or batch) follows normally the same steps like on the original mainframe environment, in dependence with the program’s typology and language. It comprehends:

- source normalization
- SQL coding conversion
- SQL precompiling
- XCICS precompiling
- PL/I compiling

In order to give an user-friendly interface to all compiling processes, the "xpli" script is supplied.

SYNOPSIS

xpli [options] <source>

USE

xpli is used to compile both TP and batch PL/I programs. The filename specified as source has to be suffixed with ".pre", and all files generated have the base name of the source.

Options

- \-b precompiles for batch use
- \-s precompiles SQL
- \-c generates object file (.o)
- \-x generates executable file
- \-l generates shared library (.so|sl) <default>
- \-o<name> specifies an output name
- \-d compiles for debug
- \-L generates listing
- \-I<path> specifies an alternate include path
- \-f removes target option before generation

xpli produces the following files:

<sourcename>.errors List of errors caught during (pre)compiling phases
<sourcename>.lst List file if required (-L)

Return codes

A return code of 0 (zero) is returned if compiling succeeded. Non zero if something went wrong.

Configuration

xpli may be configured both using environment variables and a configuration file.
**The environment**

Some environment variables should be previously set to let xpli work properly:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPLI_OBJECT_PATH</td>
<td>default directory for generated objects</td>
</tr>
<tr>
<td>XPLI_OUTPUT_PATH</td>
<td>default directory for generated executables/libraries</td>
</tr>
<tr>
<td>XPLI_STB_PATH</td>
<td>default directory for generated stb files (for debugging)</td>
</tr>
<tr>
<td>XPLI_PROPLI_CONF</td>
<td>path to pccpli.conf (Liant precompiler conf.)</td>
</tr>
<tr>
<td>XPLI_RDBMS_TYPE</td>
<td>database type: db2 or oracle</td>
</tr>
<tr>
<td>XPLI_INCLUDES_PATH</td>
<td>path for PLI includes</td>
</tr>
<tr>
<td>XPLI_PREPROCESSOR_OPTIONS</td>
<td>options to be passed to the Liant PLI preprocessor</td>
</tr>
<tr>
<td>XPLI_COMPILER_OPTIONS</td>
<td>options to be passed to the Liant PLI compiler</td>
</tr>
</tbody>
</table>

**The configuration file**

The environment variable listed above may be contained in a file named `xpli.conf` which is loaded at xpli startup. In this phase, xpli looks for the its configuration in present working directory and then in $HOME/etc. An alternative file may be specified setting the `XPLICONF` environment with the full path to the desired configuration file.

**Sample configuration file**

```bash
# configuration file for xpli
# XPLI_INCLUDES_PATH="/home/fabrizio/xcixc/plxcpp"
XPLI_OUTPUT_PATH="/home/fabrizio/xcixc/plxcpp"
XPLI_PROPLI_CONF="/home/fabrizio/xcixc/plxcpp"
XPLI_COMPILER_OPTIONS="-c -O2 -march=2,72"
XPLI_PREPROCESSOR_OPTIONS="-c -O2 -march=2,72"
XPLI_STB_PATH="/home/fabrizio/xcixc/plxcpp"
```

**Example**

```bash
# xpli -d PL02.plx
xpli: loading configuration from /home/fabrizio/xcixc/plxcpp.conf
xpli: include PL02.plx PL02.pp
xpli: precompiling CFS calls
xpli: compiling PL/I source PL02.pli
xpli: compiler output is /home/fabrizio/xcixc/plxcpp.conf
pli: linking library /home/fabrizio/xcixc/plxcpp.conf
pli: linking library /home/fabrizio/xcixc/plxcpp.conf
```

The compiling was successful and a shared library has been generated and located in the correct path.
12.1.3 xpre

XCICS supports a large part of the original IBM’s CICS Command Level services. Interfaces of user programs remain unchanged and programs using CICS, when migrated to UNIX/Linux, need only to be recompiled in the new environment.

Program compiling is performed through the following steps:

- Translation or precompiling;
- Compiling

The XCICS precompiler processes programs originally written in COBOL, PL/1 or C-language translates the EXEC CICS instructions coded in user programs into a language code that is compatible with the compiler and provides initialization for service functions.

The output of the precompilation step consists of an intermediate source program that will be compiled with appropriate compiler, producing the final object module.

Syntax

```sh
xpre [options] <source>
```

Options

- `-v`, `--verbose`: verbose messaging
- `-o`, `--output <file>`: specifies an alternative output file
- `-l`, `--language=<lang>`: specifies source code language. `lang` can be:
  - COBOL2 (default)
  - COBOL
  - C
  - PL/1
- `-i`, `--ignore <cmd>`: ignore the specified command while precompiling, option can be repeated
- `-t`, `--stdout`: output flushes on standard output
- `-n`, `--stdin`: input is read from standard input (`--stdout` is assumed)
- `-S`, `--nosevere`: `xprecob` handles error like warnings
- `-h`, `--help`: produces command line help
- `-d`, `--dump`: produces the list of supported commands
- `-D`, `--debug`: enables extra messages generation for debug
- `-P`, `--pooldir`: enables XCICS pooldir addressing
- `-C`, `--cvda`: changes CVDA values
- `-a`, `--autocommarea`: automatically adds DFHCOMMAREA
  - `--gbd`: if not declared precompiles C language for GDB usage
  - `--noautoreturn`: prevents automatic EXEC CICS RETURN statement at end of PL/1 programs
  - `--noexit`: prevents exit() to be transformed in EXEC CICS RETURN in C programs
- `-P`, `--pooldir`: provides addressing for internal XCICS areas
Variables to be set are:

```
XPREHOME=$XFRAMEHOME/xpre PATH=$PATH:$XPREHOMEPERL5LIB=$XPREHOME/bin:$PERL5LIB
```

`xpre` returns 0 upon a successful work, even if warning have been found. A negative value is returned if one or more errors have been found.

Messages are logged both on standard error and output file. No listing file is generated as the produced COBOL output contains remarked:

- warnings
- errors
- messages
- command report

**Precompiler Messages**

**PROCEDURE DIVISION USING found**

PROCEDURE DIVISION USING has been found. This is not normal for an XCICS program: a warning is issued.

**no linkage section found: forcing**

source code does not contains LINKAGE section. Xprecob forces it.

**unsupported/wrong command**

xprecob does not support/recognize the specified EXEC CICS command.

**unsupported/wrong option(s)**

xprecob does not support/recognize the specified EXEC CICS option.

**command ignored**

xprecob is ignoring the specified command: no code is generated for the specified EXEC CICS.

**parameter ignored**

specified parameter has no meaning for xprecob.

**option required**

specified option is required for the command
syntax error
  xprecob does not recognize the format of specified EXEC CICS command.

Requires
  specified command require specified options

no host defined
  A literal is supplied as parameter in EXEC CICS while a working storage field is required.

requires user data
  specified options requires user data
12.1.4 xbms2mod

xbms2mod is the BMS Macro Compiler from XFRAME. It allows to import BMS source originally written on and for mainframe. xbms2mod converts BMS code into a module file, to be used by XCICS.

The name of the map set and maps to import is obtained from the declarations made in the source file. The new generated module file is named according to the XSDF convention.

**Configuration**

In order to configure xbms2mod, following environments must be set:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSDF_MODULES_PATH</td>
<td>location of XSDF map modules</td>
</tr>
<tr>
<td>XSDF_INCLUDES_PATH</td>
<td>location of XSDF generated “C” includes</td>
</tr>
<tr>
<td>XSDF_COPIES_PATH</td>
<td>location of XSDF generated COBOL copies</td>
</tr>
<tr>
<td>XSDF_DUMMYSECT_PATH</td>
<td>location of XSDF generated ASM dummies</td>
</tr>
<tr>
<td>XSDF_BMSSRC_PATH</td>
<td>location of the BMS sources generated by XSDF</td>
</tr>
<tr>
<td>XMAPDEFLANG</td>
<td>(L)A(C) “C” language, Assembler, Cobol : defines the default map language</td>
</tr>
</tbody>
</table>
<pre><code>                                       | when not specified in the XSDF module                                       |
</code></pre>

The values passed to these environment variables should be customized to adapt it to the specific user's environment.

**XSDF_MODULES_PATH** is the so-called modules path and it indicates the directory containing the module files. A module file is the file resulting from the import and compilation of the original BMS code. This path is very important and should be set in accordance with the used directory s' structure.

**XSDF_INCLUDES_PATH** is the includes path. It indicates the directory which contains the include files used as header files by programs written in C language.

**XSDF_COPIES_PATH** is the COBOL copy path and it indicates the directory containing the copybooks used by the COBOL programs.

**XSDF_DUMMYSECT_PATH** is the Assembler /370 dsect path. It indicates the directory containing the structures (DSECTs) for Assembler programs.

**XSDF_BMSSRC_PATH** is the BMS sources path and it specifies where the original BMS macros, that are to be imported and compiled by XSDF, are located. This directory is also used by XSDF during the generation of the BMS macro source code, every time one or more maps are to be exported back to the mainframe environment.

Normally it is not necessary to set up all these variables for generating include, copy or dummy section elements: if, for example, only COBOL programs are used then the path variables for C or Assembler structure files need not to be defined.

If xbms2mod cannot locate the configuration environments, or some environment variables are not set, it will assume the following default values:

```plaintext
modules path = $HOME/sdf
includes path = $HOME/include
copies path = $HOME/cobcopy
dummysect path = $HOME/asm
BMS sources path = $HOME/bms
```

**SYNTAX**
The syntax of XBMS2MOD is:

```
xbms2mod [options] file
```

**Options**

When no argument is specified, XBMS2MOD displays its syntax and also its options. This possibility to call XBMS2MOD without options can be used as a help display. `xbms2mod` offers the following options:

- `-v` turns verbose mode on. With verbose mode on, XBMS2MOD prints out its settings.
- `-d` do nothing (for compatibility with older version)
- `-s` shifts map one byte right (default behavior)
- `-l` shifts map one byte left
- `-r` shifts map one byte right
- `-c` enables copybook generation. By default no copybook is generated. The specification of the `-c` option produces the creation of copybooks. The programming language in which the copybook is generated depends on the `LANG` statement coded in the definition of the map using BMS macros. If the `LANG` statement is omitted, it will be assumed that an Assembler copy must be generated.
- `-j` enables XJWEB customizable JSPs generation
- `-M<path>` specifies an alternate path for modules
- `-I<path>` specifies an alternate path for C and PL/I includes
- `-C<path>` specifies an alternate path for copybooks
- `-A<path>` specifies an alternate path for Assembler DSECT
- `-J<path>` specifies a path for JSPs
- `-e` assumes ASSEMBLER as default language (default is COBOL)
- `-a` forces STORAGE=AUTO mode
- `-k` keeps slack field (LENGTH=0) in module
- `-i` ignores syntax errors
- `-g` generates alternate copybook structure
- `-L<lang>` overrides language specification for user structures generation. `<lang>` can assume the following values:
  - cobol generates COBOL copybook
  - pli generates PL/1 include file
  - clang generates C include file
  - asm generates 370 assembler DSECT
- `-t<type>` overrides system specific settings for COBOL copybooks generation. `<type>` can assume the following values:
  - legacy generates COMP length fields
  - native generates COMP-5 length fields (default)
Override settings for structure packing in C includes generation.

`-p<mode>` can assume the following values:

- **default** generates OS specific code
- **gcc** generates with GCC compliant code
- **attribute** generates with `__attribute__((__packed__))`
- **pragma** generates with `#pragma pack`
- **hpppragma** generates with old HP-UX 11.11 style pragmas
- **none** generates with no pragma/attribute

**Example**

```
# xbms2mod -vc -I/home/cfiles/include mapsetA
```

The instruction above will compile mapsetA with turning verbose mode on, generating Copybooks in the following directory:

```
/home/cfiles/include/ཡིག湟较量
```

Once `xbms2mod` has finished compiling a map set, it displays the following messages:

```
xbms2mod - BMS compiler version 2.0
Current configuration:
Copybook generation : enabled
First char cutting : disabled
C language header files path : /home/cfiles/include
COBOL copy files path : /users/xsdf/v2/copybook/cobol
Assembler dummy section files path : /users/xsdf/v2/copybook/assembler
```
12.2 Debug

12.2.1 Batch

Batch programs can be traced and debugged using the Animator feature offered by MicroFocus COBOL or by the ACUCOBOL interactive debugger.

As standard descriptors are used to pass and retrieve data for the program (i.e. parameters cards), cross session debug is required.
12.2.1.1 Debugging Batch programs

Debugging with Microfocus

The following steps must be executed to start the animation of a Batch program, with Microfocus animator:

1. prepare a terminal session to host the animator session
2. prepare a terminal session to start batch program
3. export the environment COBANIMSRV=<username> in both session
4. start cobanimsrv in a session
5. start batch job in the other session

Please note that such a sequence must be respected.

A terminal session (normally connected via Telnet) must be disposed for Animator. To do this, follow the following procedure:

First connect you to the UNIX system with the same user where the batch program to debug will be executed. Remember that the Animator session requires a terminal that belongs to this user, therefore logon should be made while using 'login' and not 'su'.

Once connected, set the COBANIMSRV and start animator

```
# export COBANIMSRV=MYNAME
# cobanimsrv
```

At this point your terminal is ready to interact with the Animator session.

On the other session, export the samp value and start job:

```
# export COBANIMSRV=MYNAME
# csh MYJOB.csh
```

Normally a batch program is contained in a JCL, and it is often not the first or only called program. In order to Animate the right program at the right place, it is therefore required to edit the script and to place the setting of COBANIMSRV just before the XRUN of the program you intend to debug. After XRUN, it would be opportune to turn the animator SWITCH

Example:

```
setenv COBANIMSRV "MYNAME"
XRUN MPP550
set RC=$status
if ( $RC >= $X_RCC ) then
goto EDJ
endif
if ( $?X_ONLABEL && $RC > 99 ) then
goto $X_ONLABEL
endif
source $X/COMMAND/bin/XUNSETENV
xspooladd $X/POOLOPT -CA -nl
$X/COMMON/SYSLIB/SY_CNAME $S_O2E $X_CNF_02E_JMP109.list
@ X_CNF_02E $X_CNF_02E +10
unsetenv COBANIMSRV
```

Once done it, start normally your JCL containing the program to be animated.

Parameters cards to be passed to your program cannot in this case be read directly from the JCL script, so that you have to enter them manually from the Animator terminal.

For detailed information about Animator commands and features, see your COBOL Debugging Handbook.
Debugging with ACUCOBOL

The following procedures must be executed to start the debugging of a Batch program, with ACUCOBOL debugger

**XTerm method**

This method requires an X/Server.

- activate the X/Server
- export the environment variables:
  
  ```
  export ACUDBGMODE=display
  export ACUDBGSRV=<display address>
  ```
- start the program or the job to debug

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

**Terminal method**

If an X/Server is not available, ACUCOBOL may connect to a telnet session too. The procedures are the same as described above, except for the presence of the parameter TTY in place of DISPLAY:

This method requires a two UNIX terminal connections (telnet, ssh or serial).

- connect the UNIX/Linux system, with character terminal
- run the command
  ```
  runcbl --wait
  ```
- connect another session to the UNIX/Linux system
- export the environment variables:
  ```
  export ACUDBGMODE=term
  export ACUDBGSRV=<terminal id>
  ```
- start the program or the job to debug

The device string is the one shown in field "Terminal" by the runcbl command.

**Example**

On the UNIX terminal 1:

```
# runcbl --wait
Named pipes created
Terminal: pts/14
Waiting for application runtime to open pipe
```

On the UNIX terminal 2:
Thin client method

To use this method, the ACU thin client must be installed on the programmer workstation.

- open a DOS shell on the local PC, and run
  
  ```
  acuthin --wait --port <port number> --restart
  ```

- connect another session to the UNIX/Linux system

- export the environment variables:
  
  ```
  export ACUDBGMODE=thin
  export ACUDBGSRV=<thin client id>
  ```

- start the program or the job to debug

The thin client address is the address of the workstation, followed by ":" and the port number.

Example

On the workstation at IP 192.168.1.32, open the DOS shell and type:

```
C:\Acucorp\Acudbl700\AcuGT\bin> acuthin --wait --port 8000 --restart
```

On the UNIX terminal 2:

```
export ACUDBGMODE=thin
export ACUDBGSRV=192.168.1.32:8000
csh myjob.csh
```
Online

XCICS/TS offers some facilities to easily debug user programs.

The easiest facility is the log tracking: when transaction logging is active, all XCICS instructions and parameters are logged in the terminal log file and the program output on stdout and stderr (that means DISPLAYS, PUT and printf commands) is redirected on the same log file too.

This means that user can track its programs, by reading the log of performed instructions and, eventually, of the messages he sent from the program.

Please refer to XCICS configuration books to know more about XCICS logs.

Of course this method is very useful to track unexpected situations that may occur, but to do application development the best solution is the on-line animation system.

This facility enables the debug of user application through interactive debuggers. Every language has its own debugger.
12.2.2.1 Debugging COBOL

COBOL code may be easily debugged using the native compiler features. These feature vary depending on the COBOL compiler used.

XCICS/TS supports the following COBOL compilers:

- Microfocus ServerExpress
- ACUCOBOL

Debugging with MicroFocus Server Express

Cross-session method

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

Debug of programs running on current terminal

The following steps must be followed to start the debug of a the current terminal:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON,DISP=<display address>
```

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the Microfocus animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

Debug of a specific transaction code

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON,DISP=<display address>,TRAN=<transid>
```

Whenever the specified transid is started in the XCICS region, the Microfocus animator will start an XTERM session on the target display.

Debug of programs running on a specific terminal

Whenever the programmer wants to debug programs running on background terminal (ie. on a printer terminal), the TERM parameter must be provided, according to this procedure:
- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

CEDB ON, DISP=<display address>, TERM=<termid>

Whenever a transaction is started on the requested terminal identifier, the Microfocus animator will start an XTERM session on the target display.

**Unsolicited Dynamic Attachment Method**

This method allows the normal execution of an XCICS program, and only when the user wants to start to debug, attaches the debugger to the running process.

The following steps must be followed to start the animation of a TP program:

- connect XCICS using the terminal emulator
- enter the transaction:

  ```
  CEDB ON
  ```

- wait for terminal to switch in debugging status
- take note of the PID shown on the screen (the PID is always shown in the X4J status bar)
- open a telnet session and enter:

  ```
  anim <PID>
  ```

Note: when connecting XCICS with a TN3270 emulator (i.e. IBM Personal Communication), obviously no PID number is shown anywhere. In order to easily get the debug status of the terminal and the PID of the working process, simply enter on the screen

```
CEDB
```
Debugging with ACUCOBOL Extend

XTerm method

This method requires an X/Server to be available on the programmer's desktop and it allows the debug of the COBOL programs running on the programmer's terminals as well as background tasks or terminals.

Debug of programs running on current terminal

The following steps must be followed to start the debug of a the current terminal:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON, DISPLAY=<display address>
```

- wait for the terminal to switch in debug mode
- enter the transaction code to debug
- the ACUCOBOL debugger animator will start in an XTERM session on the target display

The display address must be in the UNIX standard display format: address:display (i.e. 192.168.1.32:0). If no DISPLAY is set to * (asterisk) the display 0 (zero) of the connected terminal is assumed.

Debug of a specific transaction code

Whenever the programmer wants to debug a specific transaction code, even if running on another terminal or as a background task, the TRAN parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON, DISPLAY=<display address>, TRAN=<transid>
```

Whenever the specified transid is started in the XCICS region, the ACUCOBOL debugger will start an XTERM session on the target display.

Debug of programs running on a specific terminal

Whenever the programmer wants to debug programs running on background terminal (ie. on a printer terminal), the TERM parameter must be provided, according to this procedure:

- connect XCICS using the terminal emulator
- activate the X/Server
- start the transaction CEDB with these parameters

```
CEDB ON, DISPLAY=<display address>, TERM=<termid>
```

Whenever a transaction is started on the requested terminal identifier, the ACUCOBOL debugger will start an XTERM
session on the target display.

**Terminal method**

If an X/Server is not available, ACUCOBOL may connect to a telnet session too. The procedures are the same as described above, except for the presence of the parameter TTY in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a unix session (telnet, ssh or serial) and enter:

  ```
  runcbl --wait
  ```

- start the transaction CEDB with these parameters

  ```
  CEDB ON,TTY=<device>
  ```

The device string is the one shown in field "Terminal" by the runcbl command.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.

**Example**

On the UNIX terminal:

```
# runcbl --wait
Named pipes created
Terminal: pts/14
Waiting for application runtime to open pipe
```

On the XCICS terminal:

```
CEDB ON,TTY=pts/14,TRAN=ACCT
```

**Thin client method**

If XCICS has been configured with the option "allow_acu_thin=yes", programmers have to debug on the ACUCOBOL thin client. To do that, the ACU thin client must be installed on the programmer workstation.

The procedures are the same as described above, except for the presence of the parameter THIN in place of DISPLAY:

- connect XCICS using the terminal emulator
- open a DOS shell on the local PC, and run

  ```
  acuthin --wait --port <port number> --restart
  ```

- start the transaction CEDB with these parameters

  ```
  CEDB ON,THIN=<thin client address>
  ```

The thin client address is the address of the workstation, followed by ":" and the port number.

To debug a specific XCICS terminal or a specific transaction code, add the parameters TERM or TRAN, as described before.
Example

On the workstation at IP 192.168.1.32, open the DOS shell and type:

```
C:\Acucorp\Acudbl700\AcuG7\bin> acuthin --wait --port 8000 --restart
```

On the XCICS terminal:

```
CEDB ON,THIN=192.168.1.32:8000,TRAN=ACCT
```
12.2.2. Debugging C

C language programs may be debugged using every symbolic debugger, which is able to remotely attach a process. In any case XCICS process to attach must be created with this procedure:

- connect XCICS using X4J
- enter CEDB ON
- wait for debug to be activated
- use the PID displayed to connect with your debugger

Using GDB

To debug interactively C programs with GDB (GNU Debugger), this procedure should be followed:

1. compile programs with debug symbols (-g option)
2. define an empty entry point (function) to be compiled & linked into a shared library to preload with "load library" command in xcics.conf. I.e.

```c
int gdb_entry_point() { return 0; }
```
3. put a call to this function in the module to debug
4. startup XCICS
5. connect XCICS and issue CEDB ON and wait for its messages
6. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
7. from command-line, change PWD to source directory and issue:

```
gdb $XFRAMEHOME/bin/xcicsd <PID>
```
8. set a breakpoint on the dummy entry point and continue execution. I.e

```
gdb> break gdb_entry_point
gdb> continue
```
9. continue using the transaction
10. when the program execution reaches the dummy entry point, debugger will stop
11. at this point GDB have to reload symbols from user library using the sharedlibrary command. I.e.

```
gdb> sharedlibrary
```
12. GDB is now ready to debug the application

Remember that user programs is always reloaded from the shared library every time the transaction restart.

Please refer to GDB documentation for further information about it.

Using DBX on AIX

To debug interactively C programs with DBX, this procedure should be followed
1. compile programs with debug symbols (-g option)
2. startup XCICS
3. connect XCICS and issue CEDB ON
4. for the CEDB messages on terminal.
5. At this point we have a dedicated process for debug: PID is shown in X4J status bar (or using XADM PID)
6. from command-line, change PWD to source directory and issue:

```
    dbx -a <PID>
```

7. set a breakpoint on the module or function to debug and continue execution. I.e

```
    (dbx) stop in MYPROG
    (dbx) c
```

8. continue using the transaction
9. when the program execution reaches the dummy entry point, debugger will stop.
10. now it is possible to use dbx commands to debug the program

Remember that user programs is always reloaded from the shared library every time the transaction restart.
Please refer to DBX documentation for further information about it.
12.2.2. Debugging PL/I

PL/I on-line programs can be debugged using Liant Codewatch™.

The following steps must be followed to start the interactive debug of an XCICS program:

- connect XCICS using X4J
- issue "CEDB ON"
- wait for CEDB to complete
- issue "XANM PLISTOP=<MODULE>" to define to PL/I entry to break on
- read the PID number on the status line of X4J

Once PID is obtained, from a command line session, issue the command:

```
xcw <PID> <MODULE>
```

This script is facility provided with XFRAME, which starts codewatch and instructs it to attach the specified process, to load defined module and start debugging.

Please refer to your Liant manual for further information about Codewatch™.
12.3 Code changes

In order to run on a UNIX system programs coming from the mainframe environment must be modified. Even if there are only few changement in the source code, it is very important to understand them.

Changements may concern:

- ASCII/EBCDIC collating sequence
- file declaration syntax
- RDBMS codes and syntax

File declaration

While using xvsamRts, only little changes to COBOL coding are required because, by utilizing the EXTFH (External File Handler) feature offered by MicroFocus COBOL, xvsamRts offers the possibility to access flat UNIX as well as XVSAM files at the same time, through the standard COBOL READ, WRITE, START etc. instructions.

The required changes in user programs regards:

- SELECT clause
- ACCEPT FROM DATE / CURRENT-DATE usage

But you are not involved in the modification process, as all required programs' changes are dynamically performed by the xcbob procedure. It means that original code coming from mainframe remains mainly the same.

SELECT clause modification

In the original mainframe environment, the internal file name used in the program code is linked to the cluster name using the features of the JCL language, DD for MVS or OS/390 and DLBL for VSE. More exactly the name specified in the ASSIGN TO of a SELECT clause is used to link logical name and cluster name.

In the XFRAME environment, the assignment is made using the EXTERNAL clause: the assignment is done by an environment variable defined and exported in the environment where programs run.

Therefore the following ASSIGN TO syntax will be required:

```
SELECT <internal-filename> ASSIGN TO EXTERNAL <external-filename>
```

Where `<internal-filename>` is the logical name of the file used in the COBOL code, and `<external-filename>` is the cluster name of the data set: both them will passed to the environment variable used in the corresponding JCL to link the logical with the cluster name.

The following sample shows modifications applied on a COBOL source directly imported from mainframe:

Mainframe version

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT ID2MOVM ASSIGN TO SYS010-FBA1-ID2MOVM
    ORGANIZATION IS SEQUENTIAL
    ACCESS MODE IS SEQUENTIAL
    FILE STATUS IS ID2MOVM-STATUS.
  SELECT ID1ANA ASSIGN TO SYS010-FBA1-ID1ANA
    ORGANIZATION IS INDEXED
    ACCESS MODE IS DYNAMIC
    RECORD KEY IS IDIGES-KPREGES
    FILE STATUS IS IDIANA-STATUS.
```
XFRAME Version

The association between the two entities logical-name and cluster-name takes place in the setenv or export statement, which substitutes itself to the DD or DLBL card:

```
setenv dlb_ID2MOVM "MY.CLUSTER.MOVM,cat=CAT1,type=flat"
setenv dlb_ID1ANA  "MY.CLUSTER.ANA,cat=CAT2,type=vsam"

export dlb_ID2MOVM="MY.CLUSTER.MOVM,cat=CAT1,type=flat"
export dlb_ID1ANA="MY.CLUSTER.ANA,cat=CAT2,type=vsam"
```

**ACCEPT FROM DATE /CURRENT-DATE**

Depending on the system and COBOL versions, programs coming from mainframe normally get the system date using ACCEPT <var> FROM DATE or MOVE CURRENT-DATE TO statements.

MicroFocus COBOL supports these statements, but, in order to grant the possibility to run programs with dates different from the system one, the following

```
ACCEPT <varname> FROM DATE and MOVE CURRENT-DATE TO <varname>
```

must be substituted by:

```
DISPLAY "UNIX_CURRENT_DATE" UPON ENVIRONMENT-NAME
ACCEPT <varname> FROM ENVIRONMENT-VALUE
```

Depending on the originating COBOL version, batch programs coming from VSE or MVS normally get the system date using the ACCEPT <varname> FROM DATE - in case of COBOL II - or MOVE CURRENT-DATE - in case of ANS74 or COBOL I - instructions.

The format of the <varname> field, where the date is returned by the system differs in the two cases: if the MOVE CURRENT-DATE statement is used, the date is 8 bytes long, and it has format DD/MM/YY (in case the European notation for the date has been chosen on your system); if the ACCEPT FROM DATE has been used the returned date has format YYMMDD in an unsigned element numeric data item six digits in length.

MicroFocus COBOL is able to support both these two statements, but, in order to give the user the possibility to run his programs with a date that can differ from the current date, the ACCEPT <varname> FROM DATE and MOVE CURRENT-DATE TO <varname> are substituted with:

```
DISPLAY "UNIX_CURRENT_DATE" UPON ENVIRONMENT-NAME
ACCEPT <varname> FROM ENVIRONMENT-VALUE
```

being <varname> a COBOL variable initialized by default with the system date, but that you can modify within one or more procedure.
Therefore, in dependence by which statement have you used in your programs to catch the date, the setting of the UNIX-CURRENT-DATE environment variable will be carried out differently in the XRUN batch manager.

The following sample shows modifications applied on a COBOL II program coming from mainframe:

**Mainframe version**

```cobol
PROCEDURE DIVISION.
    ACCEPT TODAY-DATE FROM DATE.
```

**XFRAME version**

```cobol
PROCEDURE DIVISION.
    DISPLAY "UNIX_CURRENT_DATE" UPON ENVIRONMENT-NAME.
    ACCEPT TODAY-DATE FROM ENVIRONMENT-VALUE.
```
12.3.1 EBCDIC Collating Sequence

As UNIX works with the ASCII character set, numbers from zero to nine come before letters from A to Z, and that is exactly the contrary of what happens on a mainframe environment, where EBCDIC is used.

Therefore some IF or PERFORM UNTIL clauses with break for condition GREATER or LOWER THAN may lead to different results depending on the chosen collating sequence. If you have written your programs taking into consideration that numeric values always follow letters, it may be necessary in some cases to change the logic of comparisons.

The same may happen when supplying the lower and upper limit in the parameters cards for the program. It must be necessary in some cases to change these values to produce the same results.

COBOL internal SORT too are influenced by the collating sequence and they may be controlled by using the appropriate charset.

The standard way of working in UNIX is ASCII, and therefore programs automatically work in ASCII modality, except when a different COLLATING SEQUENCE is given in the SPECIAL NAMES section, or in the charset COBOL compiler directive . This allows to maintain a sequence where letters precede numbers.

The following example shows how a COBOL program can be modified:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
    SOURCE-COMPUTER IBM-370.
    OBJECT-COMPUTER IBM-370
    PROGRAM COLLATING SEQUENCE IS EBCDIC.
    SPECIAL-NAMES.
        DECIMAL-POINT IS COMMA
        UPSI-0 ON IS SW0
        C01 IS NPA0
```

These lines may be automatically inserted by xcob during normalization phase that precedes compilation through MicroFocus.

The ordering sequence that better fit to the logic of your programs is always an important decision which is to be taken after having analyzed which effect the changing of collating sequence may have on your programs.

In any case, independently on which set of characters - EBCDIC or ASCII - you have decided to use, data are always stored by C-ISAM accordingly to the ASCII sequence, and therefore you should be careful when you want initiate positioning - by means of the START or START BROWSE instructions - on a certain record of a data set.

Searching a table for a table element that satisfies a specific condition - SEARCH ALL instruction - may produce invalid results in case the table is not sorted accordingly to the used collating sequence, as in case it has been loaded with a START followed by READ sequential next statements, and you use the EBCDIC collating sequence as the records have been read using the ASCII native sequence.

In this case you should change the binary search in a sequential search, done by means of the SEARCH instruction without ALL, or the table should be sorted after loading and before starting a SEARCH ALL, (i.e. using the COBOL table SORT statement)

The SORT instruction orders elements in the table accordingly to the ASCENDING or DESCENDING KEY sequence you have specified when defining the occurrences of the table.

Under IBM /370 mainframe COBOL, if you perform arithmetic operations on non-numeric data, it causes a program check (data exception or 0C7); if you want to have also in UNIX the same behavior, you must be sure that the F run-time switch is set to the default “+F”.

If your source program contains PIC 9(n) DISPLAY usage data items which contain spaces, it may fail at run-time.

MicroFocus COBOL is more restrictive than the IBM /370 COBOL compiler you are used with, so that the default actions or assumptions made by the MicroFocus COBOL after it has issued W or E level error messages may differ from those made by the IBM /370 mainframe compiler.

It is i.e. the case of 88 levels following the item they refer to. The 88 level must be defined with a numeric PICTURE if the field they refer to is numeric, or alphabetic if the referring field has a PICTURE X definition. Not respecting this rule may cause conditions not to be correctly recognized.

Also multiple VALUE in a 88 level should respect the correct syntax - comma followed by blank, as separator among them, otherwise only the first value is considered.
Example:

```
X2CA 03  W-HH PIC XX.
  007300  88  OKGX1 VALUE '01' '03' '05' '07' '09' '11'.
  007400  88  OKGX0 VALUE '04' '06' '08' '10' '12'.
X2CA 03  W-GG PIC XX.
  007600  88  OKG28 VALUE '01' THRU '28'.
  007700  88  OKG30 VALUE '01' THRU '30'.
  007800  88  OKG31 VALUE '01' THRU '31'.
```

MicroFocus COBOL expects at least one space after comma to separate file names in OPEN and CLOSE statements, whereas IBM /370 not.

Results of a COMPUTE statement after an OVERFLOW condition are incompatible between the mainframe environment and MicroFocus COBOL. The solution consists in using the ON SIZE clause if there is the possibility of numeric OVERFLOW; you should never depend on the result if this condition occurs.

MicroFocus COBOL system uses the concept of two numeric registers designed for worst-case calculation. This means you need 18 decimal digits plus one for overflow) before the decimal point for rounding capability during MULTIPLY and DIVIDE operations. You also need 18 decimal digits (plus one for overflow) for temporary overflow during ADD and SUBTRACT operations. For this reason, intermediate result calculations by MicroFocus COBOL are more accurate than in OS/VS COBOL and VS COBOL II.

An IBM /370 mainframe COBOL ACCEPT statement from SYSIN is not supported by MicroFocus COBOL; for larger amounts of data (that is, a set of parameter cards) a new sequential file is automatically created by the procedural language converter and the program is modified in order to accept them from the COMMAND-LINE.

The IBM /370 COBOL DISPLAY statement to SYSDUM (or any other DDNAME) is not supported by the MicroFocus COBOL; all DISPLAYs are directed to the terminal; if a station acting as central console is provided, the DISPLAY instruction may be modified in CALL ‘CBLDISP’ which will send the message to the appropriate console (i.e. CA-UNICENTER).

Under MicroFocus COBOL, when you redefine a VS COBOL II POINTER variable as a COMP field, and make an arithmetical calculation on this field to change the value of the POINTER, its behavior will not be the same as under IBM /370 COBOL.

The problem is because, in MicroFocus COBOL, the POINTER has a format defined by the machine architecture, whereas the VS COBOL II POINTER uses a /370 machine architecture. MicroFocus COBOL system provides syntax in the SET statement which enables you to change the value of a POINTER. See your Language Reference for details on how to do this.

The mainframe COBOL NOMINAL KEY IS clause for relative and indexed files is not supported by MicroFocus COBOL. For both types of file, it is necessary to replace the NOMINAL KEY IS clause with the ANSI standard RECORD KEY IS clause. Also, for indexed files, it will be necessary to change any Procedure Division reference to nominal key in your source program so that they reference records key.

xsvamRts and MicroFocus COBOL check both the minimum and maximum record length of a variable length file when it is opened. The file status is automatically set to “39” if the attributes do not match. However the mainframe environment only checks the maximum length value.

Calls to subprograms using MicroFocus COBOL system default to dynamic linkage, whereas call to subprograms on an IBM /370 mainframe default to static linkage. It is suggested to specify the NODYNAM system directive when you submit your source program to the COBOL compiler; this will cause any CALL operations to be static, see your System Reference for further details on this directive.

The COBOL EXAMINE instruction is to be replaced by INSPECT.
12.3.2 DB2 to Oracle

**Datatypes**

### DATE

<table>
<thead>
<tr>
<th>DB2 datatype</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 default format</td>
<td>DD/MM/YYYY (may change)</td>
</tr>
<tr>
<td>Oracle datatype</td>
<td>DATE</td>
</tr>
<tr>
<td>Oracle default format</td>
<td>DD-MON-YY</td>
</tr>
</tbody>
</table>

To normalize it, the `NLS_DATE_FORMAT` parameter must be changed in one of the following ways:

- globally using `init.ora`
- with the command:

  ```sql
  ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY/MM/DD';
  ```

Both XCICS and SQLLOADER (XRUN) support the setting of the parameter.

XCICS uses the configuration parameter `rdbms_date_format` in `xcics.conf`. I.e.

```sql
set rdbms_date_format = 'YYYY/MM/DD';
```

SQLLOADER (XRUN) uses the environment `XRUN_SQL_DATE_FORMAT`.

I.e.

```sql
export XRUN_SQL_DATE_FORMAT = "YYYY/MM/DD"
```

No change ment to user program is required

### TIMESTAMP (on Oracle 9i)

<table>
<thead>
<tr>
<th>DB2 datatype</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB2 default format</td>
<td>DD/MM/YYYY (may change)</td>
</tr>
<tr>
<td>Oracle datatype</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Oracle default format</td>
<td>DD-MON-YY HH.MI.SSXFF AM</td>
</tr>
</tbody>
</table>

To normalize it, the `NLS_TIMESTAMP_FORMAT` parameter must be changed in one of the following ways:

- globally using `init.ora`
with the command:

```
ALTER SESSION SET NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD HH24:MI.SSXFF'
```

Both XCICS and SQLLOADER (XRUN) support the setting of the parameter.

XCICS uses the configuration parameter `rdbms_timestamp_format` in xcics.conf.

```
set rdbms_timestamp_format='YYYY-MM-DD HH24:MI.SSXFF' ;
```

SQLLOADER (XRUN) uses the environment `XRUN_SQL_TIMESTAMP_FORMAT`:

```
export XRUN_SQL_TIMESTAMP_FORMAT="YYYY-MM-DD HH24:MI.SSXFF"
```

User programs changesements

The DB2 `CURRENT_TIMESTAMP` must become `LOCALTIMESTAMP`:

```
EXEC SQL SELECT SURNAME, NAME, CURRENT_TIMESTAMP INTO :CX, :CY, :CTS FROM TABLE1 END-EXEC.
EXEC SQL SELECT SURNAME, NAME, LOCALTIMESTAMP INTO :CX, :CY, :CTS FROM TABLE1 END-EXEC.
```

Please note the `LOCALTIMESTAMP` must be used instead of using `CURRENT_TIMESTAMP` as it is a TIMESTAMP WITH TIME ZONE, while `LOCALTIMESTAMP` is a simple TIMESTAMP.

Please note also that `SYSTIMESTAMP` is not sensible at TIME ZONE changes.

TIME ZONE management

It is possible to establish the time zone for a session with the command:

```
ALTER SESSION SET TIME_ZONE = '-5:0'
```

**TIMESTAMP (on Oracle 8i)**

<table>
<thead>
<tr>
<th>DB2 datatype</th>
<th>Oracle datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>datetime</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>DATE9</td>
<td>DATE9</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DB2 default format</th>
<th>Oracle default format</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD/MM/YYYY</td>
<td>DD/MM/YYYY</td>
</tr>
<tr>
<td>YYYY-MM-DD</td>
<td>YYYY-MM-DD</td>
</tr>
<tr>
<td>YYYY-MM-DD HH24:MI.SSXFF</td>
<td>YYYY-MM-DD HH24:MI.SSXFF</td>
</tr>
</tbody>
</table>

© 2007 H.T.W.C. Srl
The TIMESTAMP field may be simulated on Oracle8 with a user function.

**User programs changesments**

The DB2 CURRENT TIMESTAMP must become LOCALTIMESTAMP—i.e.

<table>
<thead>
<tr>
<th>DB2</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXEC SQL SELECT SURNAME,</td>
<td>EXEC SQL SELECT SURNAME,</td>
</tr>
<tr>
<td>NAME CURRENT TIMESTAMP</td>
<td>NAME LOCALTIMESTAMP</td>
</tr>
<tr>
<td>FROM TABLE1 END-EXEC.</td>
<td>FROM TABLE1 END-EXEC.</td>
</tr>
</tbody>
</table>

Generating user function

The user function LOCALTIMESTAMP may be written using:

- PL/SQL
- Java
- C language

**Generating with PL/SQL**

This simple script generates the function:

```plsql
CREATE OR REPLACE FUNCTION LOCALTIMESTAMP  
RETURN CHAR IS RC CHAR(26);
BEGIN
  SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24-MI-SS')  
  INTO RC
  FROM DUAL;
RETURN(RC);END;
```

Generating the function in PL/SQL has a problem: no milliseconds field is returned (i.e. `[2002/08/08 19:44:21:`).

**Generating with Java**

To generate in Java proceed as follow:

1. write a file named LocalTimeStamp.java containing

```java
/**
 * LocalTimeStamp.java
 */
import java.util.*;
import java.text.*;
public class LocalTimeStamp 
public static String getTimeStamp() { 
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy/MM/dd HH:mm:ss.SSS");
    return sdf.format(new Date());
}
```
2. execute the following commands

```bash
# javac LocalTimeStamp.java
# loadjava -user userid/password LocalTimeStamp.class
```

3. run a SQL script containing:

```sql
CREATE OR REPLACE FUNCTION LOCALTIMESTAMP
    RETURN VARCHAR2
AS LANGUAGE JAVA
    NAME ?LocalTimeStamp.getTimeStamp() return java.lang.String?
/
```

Using this method the first 3 digit of milliseconds may be obtained (i.e. [2002/08/08 19:44:21:206 ])

Moreover Java code does not manages the oracle Time Zone therefore Java Locales should be used.

**EBCDIC sequence**

Using Oracle ordering and matches may be done according to EBCDIC sequence on different levels:

- for a single query
- for a session
- for the entire system

**Ordering the single query**

A single query may be ordered according to the EBCDIC collating sequence:

```sql
EXEC SQL SELECT CODICE, COGNOME, NOME, INTO :CD, :CO, :NO ORDER BY NLSSORT(CODICE, 'NLS_SORT=EBCDIC') END-EXEC
```

Be careful using the WHERE clause as it uses the standard ASCII sequence.

**Ordering for the session**

To set up the session seems to be the best solution, if necessary. To set up the session two runtime parameter must be set:

```sql
ALTER SESSION SET NLS_SORT = 'EBCDIC';

ALTER SESSION SET NLS_COMP = 'ANSI';
```

The first statement sets EBCDIC for ordering, and the second one forces matches to work against the EBCDIC sequence.

**Column indexing using EBCDIC**

It is possible to generate EBCDIC ordered indexes using function based indexes. I.E.

```sql
CREATE INDEX EBCIDX ON TABLE1 (NLSSORT(FIELD1, 'NLS_SORT=EBCDIC'));
```

In order to create this kind of indexes, user must have granted for query rewrite. I.e.
SQLCODE

Following tables describe the code changes between DB2 and Oracle.

<table>
<thead>
<tr>
<th>DB2</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>+1403</td>
</tr>
<tr>
<td>-1</td>
<td>-2291</td>
</tr>
<tr>
<td>-204</td>
<td>-942</td>
</tr>
<tr>
<td>-302</td>
<td>-1722</td>
</tr>
<tr>
<td>-305</td>
<td>-1405</td>
</tr>
<tr>
<td>-407</td>
<td>-1407</td>
</tr>
<tr>
<td>-501</td>
<td>-1001</td>
</tr>
<tr>
<td>-502</td>
<td>-59</td>
</tr>
<tr>
<td>-530</td>
<td>-2291</td>
</tr>
<tr>
<td>-531</td>
<td>-2292</td>
</tr>
<tr>
<td>-532</td>
<td>-2292</td>
</tr>
<tr>
<td>-534</td>
<td>-2292</td>
</tr>
<tr>
<td>-803</td>
<td>-1</td>
</tr>
<tr>
<td>-811</td>
<td>-2112</td>
</tr>
<tr>
<td>-818</td>
<td>-9999</td>
</tr>
<tr>
<td>-904</td>
<td>-62</td>
</tr>
<tr>
<td>-911</td>
<td>-60</td>
</tr>
<tr>
<td>-913</td>
<td>-60</td>
</tr>
<tr>
<td>-922</td>
<td>-1031</td>
</tr>
<tr>
<td>-923</td>
<td>-1012</td>
</tr>
</tbody>
</table>
Part XIII

Notices
13 Notices

HIGH TECHNOLOGY WORLD COMPANY S.r.l. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. HTWC may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The licensed program described in this document and all licensed material available for it are provided by HTWC under terms of the HTWC Customer Agreement, or any equivalent agreement between us.
Endnotes 2... (after index)