# zJOS-XDI© Agent Design & Protocol

## Introduction

To provide a true enterprise automation solution, zJOS-XDI since version 2.1.3 has been enhanced with a new component, called socket interface. This is a socket server program which runs on zJOS-XDI on the mainframe to interface information exchange between system events listener on zJOS-XDI on the mainframe and system events listener on other hosts, which is called *zJOS-XDI agent* or simply *zJOS agent*.

The objective of zJOS agent is to automate computer host on which the agent is running integrally with zJOS-XDI on mainframe site. All parameters and definitions of each host are kept by zJOS-XDI on the mainframe and controlled integrally from a single dashboard. Once all parameters are ready, the whole enterprise will now be able to run unattendedly on the dark room.

zJOS agent is a simple program, which works on any host outside host on which zJOS-XDI is running. Agent listens, collects and transfer system events information to zJOS-XDI on the mainframe via socket interface, and listens, receives and executes zJOS-XDI instructions.

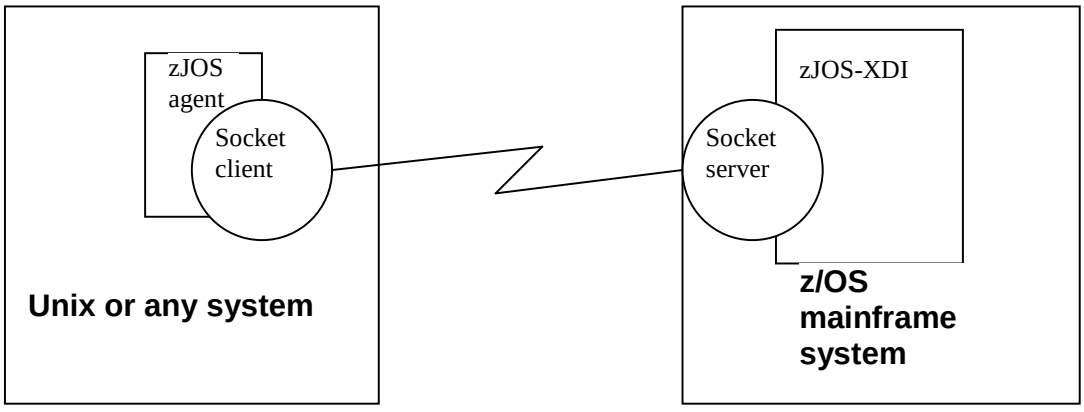


Figure 1. zJOS-XDI multiplatform design

# Agent Design

To have its functions work properly, agent must consist of 2 major logic components which work simultaneously:

1) System events listener or simply events listener
2) Socket interface

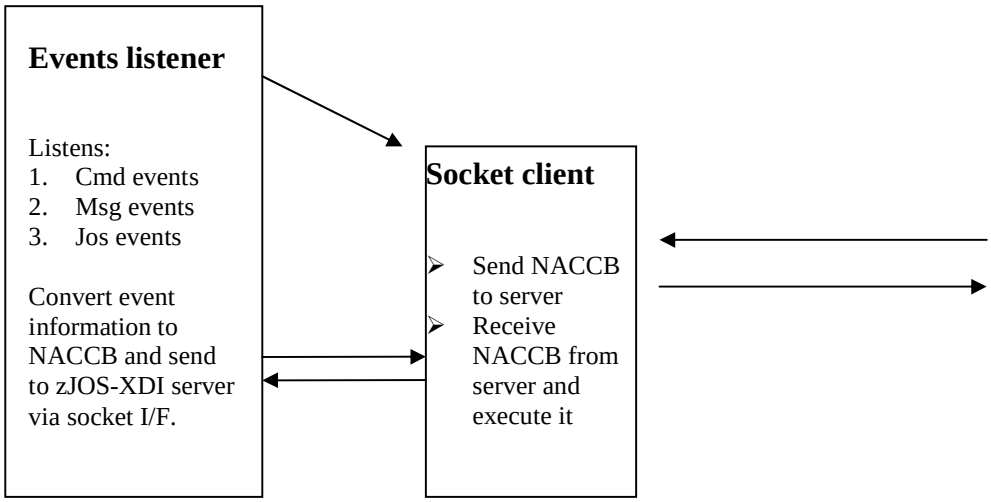


Figure 2. zJOS-XDI agent's major components

## Events Listener

Event listener is a main task of zJOS agent to listen events regarding commands and messages occurrences, and job status. It must capable to capture, collect and send information to server at the time event is occurred. Event information must be converted into network agent command control block (NACCB) format prior to transport to zJOS-XDI server on the mainframe. Otherwise, zJOS-XDI server will not recognize it and purge it as garbage.

Events listener is very OS architectural dependent. It is an OS-level program. Developer must very familiar with internal OS mechanism, especially the way system events are handled by the OS. In mainframe system (z/OS), most of system events, except for job-related events, can only be trapped via subsystem interface (SSI). Job-related events are captured via resource manager. Both SSI and resource manager are kernel-level routines and must run on common segment to make them eligible from all address spaces or tasks on the system.

# Socket Client

Socket client is a purely user-level application program. Socket client is an ordinary socket program which act as a client against socket program on zJOS-XDI on mainframe site. zJOS-XDI uses IP streaming protocol (TCP) to support both IPv4 and IPv6 socket interaction. zJOS agent's socket client must follow socket common rule to establish connection with XDI. Prior to interact with zJOS-XDI server, socket client must do the following steps:

1. Obtain host name and IP address
2. Initialize socket
3. Connect to socket server on zJOS-XDI port.

When connection is established, socket client is then allowed to send NACCB to and receive NACCB from zJOS-XDI on mainframe site. As it is a streaming interaction, no guarantee that data can be sent by once send() service call nor received by once recv() service call. Socket client must put its recv() and send() service call in loop mechanism and perform it until the work is complete, which is indicated by socket ready for next send() or recv().

All socket I/O must be handled non-blocked to let agent freely interact with zJOS-XDI on mainframe site simultaneously. The best way is separating each send / receive pair into a separate task or thread. Hence, developer must familiar with multitasking programming methods.

Socket finish its work when requested to shutdown by either XDI (when XDI is shutted down) or local operator for certain reason. Internally, it does the following steps:

1. Close connection
2. Reset socket

# zJOS-XDI Socket Interaction Protocol

When connection is established, interaction between agent and server must follow the zJOS-XDI socket interaction protocol. This protocol is to simplify communication mechanism. Protocol consists of 2 major parts:

1) Interaction steps
2) Data format

## Interaction Steps

Once connection is established, agent must do the following steps:

(1) Establish connection with zJOS-XDI server on mainframe site.
(2) Send login request. When login information is invalid, zJOS-XDI rejects it. It must correct login information and retry it. Login will also fail when zJOS-XDI on mainframe site is not up.
(3) Receive login acceptance. When login information is accepted, zJOS-XDI reply 4-byte agent identifier (agent-ID). This ID must be used in subsequent interaction with zJOS-XDI server on mainframe site.
(4) Send system events management (EMS) parameters request. Ask zJOS-XDI to send all EMS parameters set related to host on which agent is working.
(5) Send workloads scheduler (SCD) parameter request. Ask zJOS-XDI to send all SCD parameters related to host on which agent is working.
(6) Receive EMS parameters. When receive loop is complete, socket client build EMS table. When EMS is not ready, zJOS-XDI reply with “not ready EMS”. Agent then runs without EMS support.
(7) Receive SCD parameters. When receive loop is complete, socket client build SCD table. When SCD is not ready, zJOS-XDI reply with “not ready SCD”. Agent then runs without workloads scheduler support. When both EMS and SCD not ready, socket client is then shutting down itself.
(8) Notify events listener on current host that socket interface is ready and provides EMS or SCD table or both to be used by events listener.
(9) Wait for request from either events listener (local) or zJOS-XDI server (remote mainframe)
(10) When requested by events listener; Send event information created by events listener to zJOS-XDI
(11) When requested by zJOS-XDI server by means NACCB; Execute the request based on detail information specified in NACCB.
(12) Back to (8), unless requested to shutdown.
(13) If requested to shutdown by events listener or local operator, send logout information to zJOS-XDI server.
(14) Shutdown

# NACCB Data Format

During interaction with server on step (2) up to (13), data stream must be formatted as network agent command control block (NACCB). NACCB consists of 2 parts:

1. NACCB header (as illustrated in figure 3)
2. Attached object/data

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 4-byte NACCB total length | 4-byte Agent-ID | 8-byte local system name | 4-byte NACCB key | 4-byte extended key | 4-byte forward pointer | 4-byte num of object | 4-byte size of object |

Figure 3. NACCB header map

.

***NACCB total length***

Total length in bytes of NACCB header and attached object/data behind the header. This must be in binary fullword format.

**Agent-ID**

4-byte agent ID given by zJOS-XDI server on login acceptance. Login request NACCB is sent without agent-ID. When login request is valid, zJOS-XDI reply NACCB with agent-ID and without object/data attached. This ID then must be used in all subsequent interaction. This must be in binary fullword format.

***System name***

Network name of the host on which agent is working. This must be 8-byte character string left justified and padded with blanks.

***NACCB key***

4-byte binary indicator to express the request or content of the attached object/data. Below is detail description of each byte of NACCB key.

Byte 0-1: Direction

X'0102' → to agent

X'0201' → to server

Byte 2: Request code

- X'01' → execute
- X'02' → save
- X'03' → replace
- X'04' → shutdown
- X'05' → login
- X'06' → logout
- X'07' → reject
- X'08' → acquire
- X'09' → information
- X'0A' → error

Byte 3: Object code

- X'01' → EMS parameter table
- X'02' → scheduler table
- X'03' → message event text
- X'04' → command event text
- X'05' → end-of-job event text (EOJ)
- X'06' → end-of-jobstep event text (EOS)
- X'07' → end-of-task information block (EOTinfo)
- X'08' → job name

Example of special NACCB keys:

```
Nac_to_agent      equ x'0102'        * NACCB is sent to agent
Nac_from_agent    equ x'0201'        * NACCB is sent to server
Nac_agent_login   equ x'02010500'    * agent is logging in
Nac_agent_accept  equ x'01020700'    * agent login accepted
Nac_agent_reject  equ x'01020800'    * agent login rejected
Nac_acq_emsparm   equ x'02010901'    * agent acquire EMS parms
Nac_acq_scdparm   equ x'02010902'    * agent acquire SCD parms
Nac_rcv_emsparm   equ x'01020201'    * agent receive EMS parms
Nac_rcv_scdparm   equ x'01020202'    * agent receive SCD parms
Nac_rep_emsparm   equ x'01020301'    * agent recv EMS parms replacement
Nac_rep_scdparm   equ x'01020302'    * agent recv SCD parms replacement
Nac_rcv_noems     equ x'01020B01'    * agent receive no EMS parms
Nac_rcv_noscd     equ x'01020B02'    * agent receive no SCD parms
Nac_exec_cmd      equ x'01020104'    * agent requested to execute cmd
Nac_exec_job      equ x'01020108'    * agent requested to run job
Nac_info_msg      equ x'02010A03'    * agent send info of MSG event
Nac_info_cmd      equ x'02010A04'    * agent send info of CMD event
Nac_info_eoj      equ x'02010A05'    * agent send info of EOJ event
Nac_info_eos      equ x'02010A06'    * agent send info of EOS event
Nac_info_eot      equ x'02010A07'    * agent send EOTinfo for SCD
```

Notes:

I. All codes above are expressed in hexadecimal notation.

II. The above examples are coded in z/Series assembly terms of equation. Developer should convert to selected language.

***NACCB extended key***

Byte 0: Error code

- X'01' → invalid agent ID
- X'02' → invalid direction
- X'03' → invalid acquired object
- X'04' → invalid request
- X'05' → invalid information
- X'06' → error returned from zJOS-XDI resource manager
- X'07' → error returned from zJOS-XDI EVX component
- X'08' → parameters/table is not available
- X'09' → unknown system name

Byte 1: OS code (reserved)

Byte 2: coding convention (reserved)

Byte 3: unused (reserved)

***NACCB forward pointer***

This field is used by agent internally for performance purposes. Interaction between event listener and socket client is much faster than socket client to zJOS-XDI server. When more than one NACCB is ready on event listener to be sent to server, socket client can only handle one NACCB each the time, whereas the rest must be held on queue. This field can be used to build queue chain, hence, once socket client is notified, it send the first NACCB and continue retrieve the rest on queue chain.

***Object number***

4-byte fullword binary stated number of object/data attached behind NACCB header.

***Object size/length***

4-byte fullword binary stated length of each attached object/data. Only object/data with the same length can be attached together on the same NACCB.

# EMS Parameter Format

EMS parameter for each particular event is placed in an zJOS-XDI control block called event control block (EVBLOK). In accordance to agent request on step (4), zJOS-XDI send a set of EMS parameters, each in EVBLOK format. Collection of EVBLOKs is then tabulated by agent and used by agent's event listener as EMS reference table. All events are ignored by listener until EMS reference table is ready.

Once EMS reference table is ready, when an event occurs, agent's event listener then looks up the table to find matched event key. Only matched events will be processed. Unmatched events are ignored.

EVBLOK format illustrated in figure 4

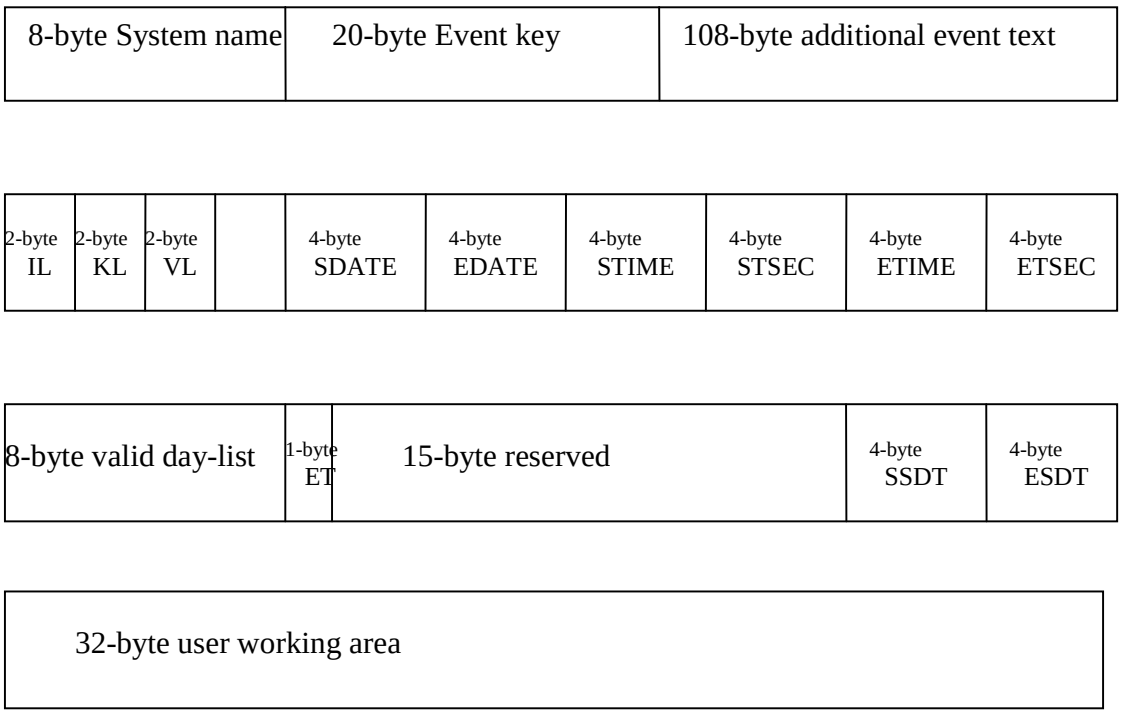


Figure 4. EVBLOK structure map

***System name***

Network name of the host on which agent is working, as one placed in NACCB. This must be 8-byte character string left justified and padded with blanks.

***Event key***

A string key to be matched with information text given by system when an event is occurred. Only event that occurres with matched key is trapped by agent and send to zJOS-XDI server in NACCB format. If the key is not matched, event then must be ignored by agent.

***Additional event text***

Event key is the first 20 bytes of event text. If event text length is more than 20-byte long, the rest is placed in this field.

***Event ID length (IL)***

2-byte binary halfword total length of system name and event key

***Event key length (KL)***

2-byte binary halfword length of event key. Although event key field is 20-byte long, event key length could less than 20-byte, and the actual key length value is placed in this field. Matching is done based on this value. For example if the key for certain message event is only the message code which is only 10-byte long, then the 10-byte message code is placed in event key field left justified padded with blanks, and KL field contains x'000A'.

***Event verb length (VL)***

zJOS-XDI has capability to substitute a portion of event text which is assumed as argument. To do so, zJOS-XDI will only check the event verb. The rest of key is reused as argument to action text. For example, MVS does not have command with verb of KILL to stop a TSO userid. To do so, user must issue console command "CANCEL U=userid". But, user want KILL command is supported. Then, XDI must provide action "CANCEL U=&arg". When "KILL JOKO" command is issued, XDI then subtitute &arg to JOKO and issue "CANCEL U=JOKO", hence TSO user JOKO is then down with abend S222 because it was cancelled. On this case, zJOS-XDI check only the verb KILL with length as specified in VL field.

***Start date (SDATE)***

4-byte start date of automation timeframe. This means, event that occures before SDATE is ignored. SDATE is a julian date and must be in packed decimal without sign zone, which is in hexadecimal notation shown as X'0YYYYDDD'.

***End date (EDATE)***

4-byte end date of automation timeframe. This means, event that occures one day or more after EDATE is ignored. EDATE is a julian date and must be in packed decimal without sign zone, which is in hexadecimal notation shown as X'0YYYYDDD', must not less than SDATE

***Start time (STIME)***

4-byte start time of automation timeframe. This means, event that occures before STIME is ignored, although within valid date range SDATE to EDATE. STIME is a standard time format in packed decimal without sign zone, which is in hexadecimal notation shown as X'HHMMSS00'.

***Start time in seconds (STSEC)***

4-byte start time of automation timeframe, the same as STIME, except it is in number of seconds in binary fullword which counted since 00:00:00. In hexadecimal notation shown as X'ssssssss'.

***End time (ETIME)***

4-byte start time of automation timeframe. This means, event that occures after ETIME is ignored, although within valid date range SDATE to EDATE. ETIME is a standard time format in packed decimal without sign zone, which is in hexadecimal notation shown as X'HHMMSS00', and must not less than STIME.

***End time in seconds (ETSEC)***

4-byte start time of automation timeframe, the same as ETIME, except it is in number of seconds in binary fullword which counted since 00:00:00. In hexadecimal notation shown as X'ssssssss'.

***Valid day-list***

8-byte valid day-list of automation timeframe. Byte 0-6 represent day of week (Sunday to Saturday), and the last byte (byte 7) represents holiday . Each byte can only have value of 1 or 0. For example, day-list X'0001010101010100' means that event will be captured only from Monday to Saturday. If occure on Sunday or in the holiday, it will be ignored by zJOS-XDI.

Note:

To make zJOS-XDI recognize holiday, zJOS-XDI admin must provide holiday calendar.

***Event type (ET)***

1-byte event type code in binary.

X'01' → message event
X'02' → command event
X'03' → time-of-day event (not applicable for agent)
X'04' → end-of-task (not applicable for agent)
X'05' → end-of-jobstep (EOS)
X'06' → end-of-job (EOJ)

***Start date in standard format (SSDT)***

4-byte start date of automation timeframe, the same as SDATE, in standard date notation. SSDT must be in packed decimal without sign zone, which is in hexadecimal notation shown as X'YYYYMMDD'.

***End date in standard format (ESDT)***

4-byte start date of automation timeframe, the same as EDATE, in standard date notation. ESDT must be in packed decimal without sign zone, which is in hexadecimal notation shown as X'YYYYMMDD'.

***User working area***

32-byte free area which can be used by agent as working area, including for forward/backward chaining pointer.

**EMS Parameters in Agent Program**

In agent program, EVBLOKs are only used as EMS reference table by event listener. Event text of matched event is attached to NACCB header, and ask socket client to send it to zJOS-XDI server.

## Scheduler Parameter Format

SCD parameters for each job either scheduled-job or triggering-job, is placed in an zJOS-XDI control block called end-of-task information block (EOTINFO). In accordance to agent request on step (5), zJOS-XDI server send a set of scheduler parameters, each in EOTINFO format. Collection of EOTINFOs is then tabulated by agent and used by agent's event listener as scheduler reference table. All job status events/notifications are ignored by listener until scheduler reference table is ready.

Once SCD table is ready, when a job status event/notification occurs, agent's event listener then looks up the table to find matched job trigger identifier. Only matched jobs will be processed. Unmatched jobs are ignored.

EOTINFO format illustrated in figure 5

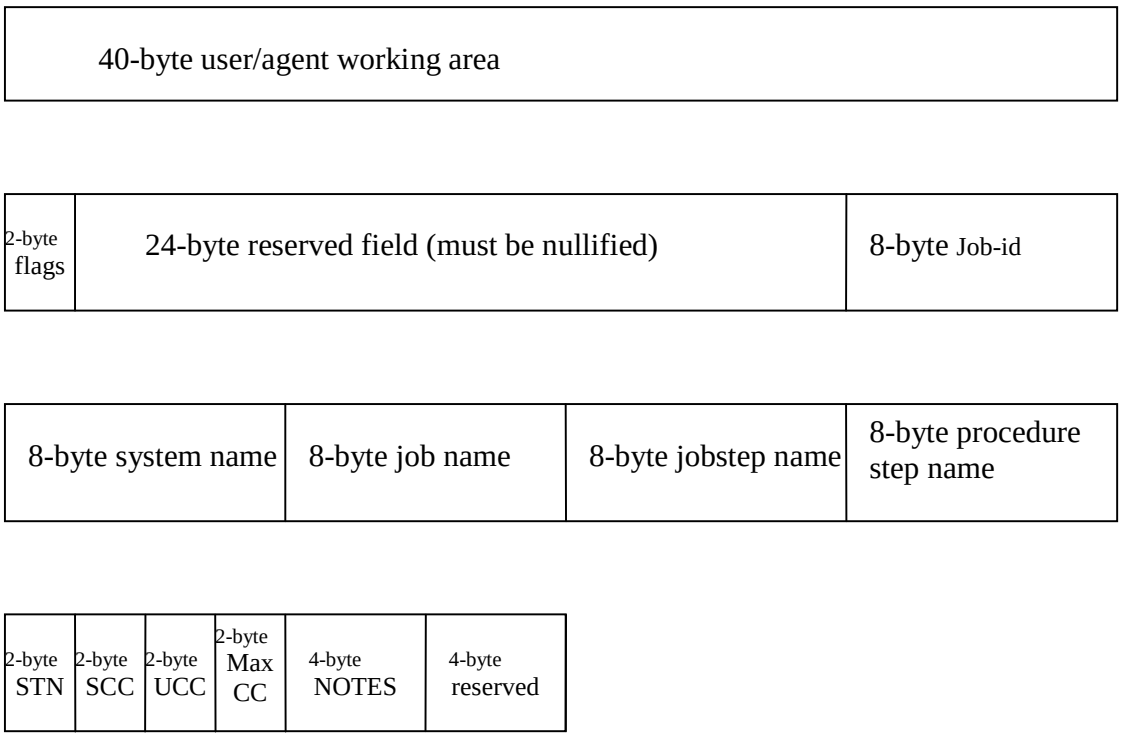


Figure 5. EOTINFO structure map

***User working area***

40-byte free area which can be used by agent as working area, including for forward/backward chaining pointer.

***Flags***

2-byte flags contains status indicators bits. .

Byte 0: Flag1
Reserved for XDI server only.

Byte 1: Flag2
Bit 0 → jobname is capturef
Bit 1 → job-id is captured
Bit 2 → job-step name is captured
Bit 3 → procedure-step name is captured
Bit 4-7 → reserved for XDI server only.

***Job identifier***

8-byte job identifier for JES-like job handler. For OS that does not have JES-like job handler, this field must be blanks. Job-id is 8-byte character string, left justified and padded with blanks. . .

***System name***

Network name of the host on which agent is working, as one placed in NACCB. This must be 8-byte character string left justified and padded with blanks.

***Job name***

In mainframe term, jobname is a name given by user to identify the job. Since job name can be duplicated among users, job handler in mainframe system, then assign a unique identifier called job identifier.

In non-mainframe environment, job name could either be the same as in mainframe, or true unique name of the job or even a program module name or batch script file name. Regardless what it is, as long as it is used by users to identify their job and represent the process, can be used as a job name for scheduler. This must be 8-byte character string left justified and padded with blanks.

***Job-step name***

In mainframe term, job-step name is a name given by user to identify processing step within a job. A job can have up to 256 steps and each must have a unique job-step name within a job.

In non-mainframe environment which support job-step like, then use its name as job-step name in triggering argument if necessary, as long as its status can be detected instantly by event listener. This must be 8-byte character string left justified and padded with blanks. System that does not support job-step like stuff, leave it blanks.

.

***Procedure-step name***

In mainframe term, procedure-step name is a name given by user to identify processing step within a procedure. Like macro in assembler program, procedure is subroutine in a job. All similar`porsion of a job can be simplified as a procedure and then called by job in each place of which it is needed. When job is interpreted by job handler (JES), procedure-step names are generated in addition to job-step name in which procedure was called. It can also be used by scheduler as additional triggering argument, as long as its status can be detected instantly by event listener.

In non-mainframe environment which support procedure-step like, then use its name as procedure-step name if necessary. This must be 8-byte character string left justified and padded with blanks. System that does not support procedure-step like stuff, leave it blanks.

***Job-step number***

2-byte field in which event listener returns job-step sequent number. This is a 2-byte halfword binary number.

***System condition code (SCC)***

2-byte field in which event listener returns system condition code. When a job was abnormally terminated (abend) because of system architectural error, system return system condition code (SCC) as the simplest error description. SCC can be used as triggering argument if necessary.

***User condition code (UCC or CC)***

2-byte field in which event listener returns user condition code. When a job was terminated either normal or abend because of non-system architectural error, system return user condition code (UCC or CC) as the simplest termination description. CC could either return code, when job-step is normally ended, or user abend code, when job was abended itself. UCC can be used as triggering argument if necessary.

***Maximum condition code (Max CC)***

2-byte field in which event listener returns maximum condition code. Max CC of the`job is the highest CC among all reported CC of all job-steps of a job. If necessary, it can be used as triggering argument when triggering job is EOJ status.

***Job notes***

Job notes consist of 4 indicator bytes. All bytes use EBCDIC character symbol. To simplify, agent which work on ASCII host should not convert them to ASCII.

Byte 0 → task status type
'J' for end-of-job (EOJ)
'S' for end-of-job-step (EOS)

Byte 1 → job type (for mainframe only)
'I' for initiated job (run on JES subsystem)
'S' for started, logged on or mounted job

Byte 2 → task termination status
'N' normally terminated
'A' abnormally terminated
'C' not executed, for example: aborted because of JCL error

Byte 3 → task location
'L' run on local host
'R' run on remote host.
Agent must always 'R'.

**Scheduler Parameters in Agent Program**

Unlike EMS, event text is not applicable for scheduler. Copy of matched EOTINFO is updated by listener and attached directly to NACCB header. So in NACCB, the object form is still EOTINFO. Listener then ask socket client to send it to zJOS-XDI server.