SIEMENS

AXCS Network Communication VC14 onwards

[redacted] AX

AXNserver und AXNclient interface

(Subsystem) Functional Specification

Version 03

Last Change: 2009-01-07

T. Eschenbach
AX PLM-E SC

Archive	Date	Signature	Doc-Number	Doc-Type	Doc-Part	Doc-Version	Misc
			10095636	**EPH**	**01S**	**03**	

History

Version	Date	Author	Change & Reason of Change / Change Request / CHARM	Review-Protocol
00	15-Mar-2007	T. Eschenbach	Previous version is 10046956-EPH-01S-02. New document ID number with migration to Caliber RM. Requiremensts for VC12 and VC13 added.	see SAP EDM(electronic review)
00	26-Mar-2007	T. Eschenbach	Incorporation of review results	see SAP EDM(electronic review)
01	22-Apr-2008	U. Köstner	Update for system version VC14A	see SAP EDM(electronic review)
01	29-Apr-2008	A. Lyncker	Incorporate review results	see SAP EDM(electronic review)
02	03-Dec-2008	U.Köstner	Update for system version VC14B	see SAP EDM(electronic review)
02	15-Dec-2008	U.Köstner	Incorporate review results	see SAP EDM(electronic review)
03	07-Jan-2009	U.Köstner	new req.key for bitfield processing	see SAP EDM(electronic review)

Table of Contents

1 Logical View

1.1 Overview of logical Subsystems

1.1.1 axn_useOfethernet

axn_UseOfethernet 08S86393

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S131191,

Applications can communicate with each other using AXCS. For the AXN software a participant with an APId is an ***AXNclient*** . The AXCS functionality is represented by the so called ***AXNserver*** . Every physical machine can run several AXNclients but there is only one AXNserver. Several AXNclients and one AXNserver on one machine build a so called virtual network.
An AXCS network consists of several virtual networks, which are connected by Ethernet using the TCP/IP protocol.
The only way for an AXNclient to send an AXCS object to another AXNclient is, to send the object to the AXNserver, which forwards the object to the target AXNclient. Only AXNclients that are logged on to the AXNserver can receive objects.
AXNserver and AXNclient communicate via message queues.

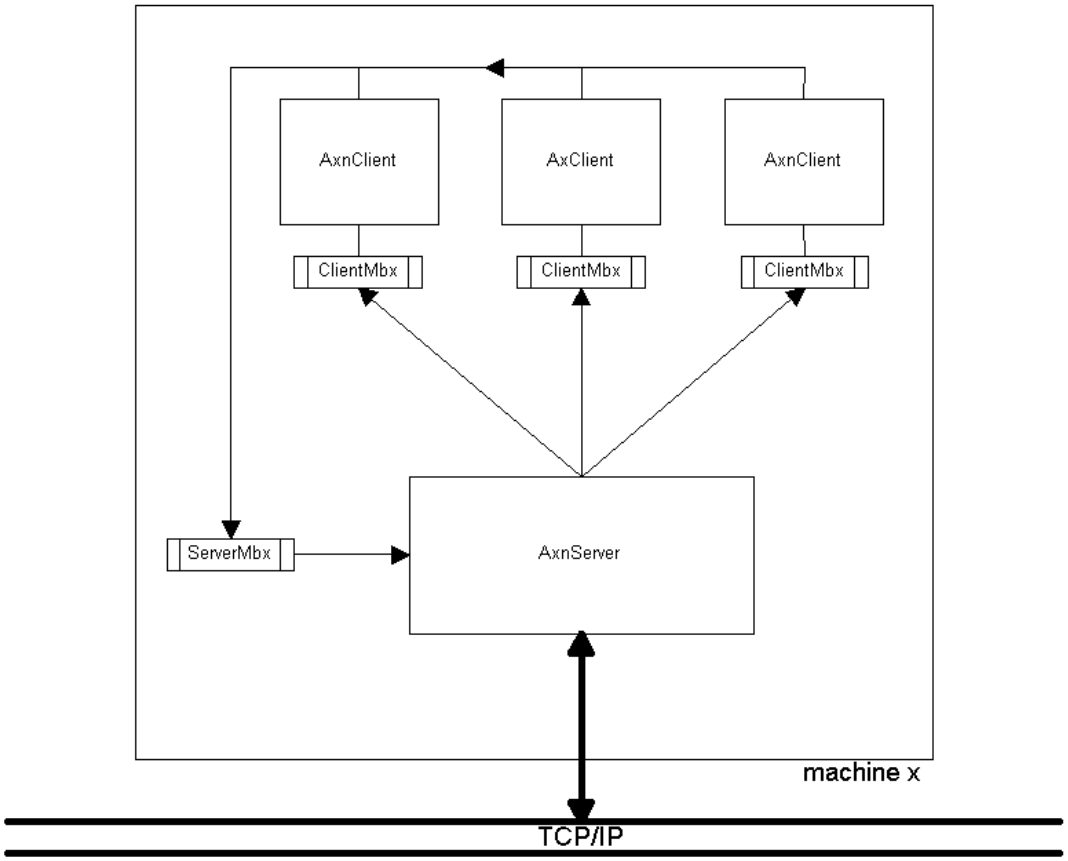


Figure 1: Virtual network on a machine

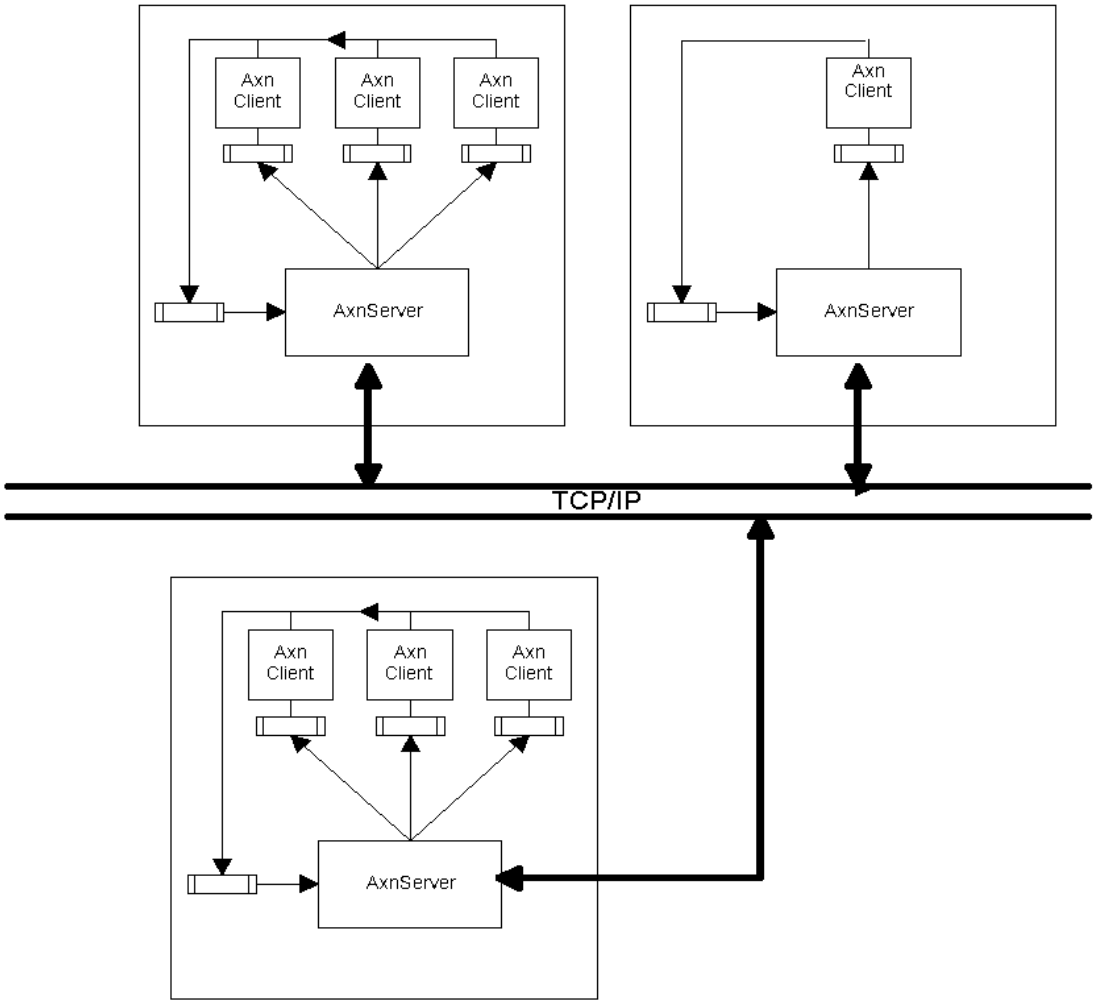


Figure 2: AXCS network with several virtual networks

1.1.2 axn_WindowsService

axn_WindowsService_CHARM144587 08S86394

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S38242,

It is possible to start the server executable on the Microsoft Windows XP and 2000 platforms as Windows Service.

1.1.3 axn_useOfethernetDll

axn_UseOfethernetDll 08S137158

Status:	Approved
Test Level	Subsystem Integration Test

Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S131191,

A client interface packed in a dynamic link library is provided and installed with the AXNserver executable.

1.2 Specification of logical subsystems

1.2.1 Logical Subsystem

1.2.1.1 Interfaces - external visible features

1.2.1.1.1 AXNclient interface (lib only til VC14 version)

axn_ClientInterface 08S86400

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	Supplier

The AXNclient interface is realized as static library. The static library is supported until all VC14 versions. In later versions AXNclient interface is realized as dynamic library.

- Every AXNclient is identified by an APId.
  Every AXNclient has its own receive mailbox.
- The AXNclients communicate by sending and receiving objects over the network using the common software functions SendObjToServer() and recvObject(). The AXNclient initiates the connection with connectToServer()
- Every object is sent as a single object, i.e. it is not possible to send packages of objects.
- The AXNserver reports changes of the network status of AXNclients to all other AXNclients in the network.

2 Deployment View

2.1 axn_AllocatedSubsystems_IAS

axn_AllocatedSubsystems_IAS 08S137284

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S135929,

For the subsystems IAS, IVS and Test the AXN software is ported as Microsoft Windows 32bit and Microsoft 64bit application.
It supports the following operating systems:

- Microsoft Windows XP(32Bit)
- Microsoft Windows 2000
- Microsoft Windows XP(64Bit)

2.2 axn_AllocatedSubsystems_SSW

axn_AllocatedSubsystems_SSW_CHARM213154 08S167744

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	n.a.

For the subsystems service software in RAD a software bridge is created to use the Microsoft Windows 32bit AXCS SW.
The following targets are part of this software bridge:

- acs_com.dll
- axnproxy.exe

2.3 axn_AllocatedSubsystems_DDIS

axn_AllocatedSubsystems_DDIS_CHARM213154 08S167710

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S10572,

For the subsystem DDIS the AXN software is ported to Windows CE 2.12/x86 and Windows CE 3.00/MIPS

2.4 axn_AllocatedSubsystems_ECC

axn_AllocatedSubsystems_ECC_CHARM213154 08S167711

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S10499,

For the subsystem DDIS the AXN software is ported to Windows CE 2.12/x86, Windows CE 3.00/MIPS and Windows CE5.00 (MP277 and TRZPS)

2.5 axn_AllocatedSubsystems_RTC

axn_AllocatedSubsystems_RTC_CHARM213154 08S167743

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	CHARM

For the subsystem RTC the AXN software is ported to QNX 6.3

3 Process View

3.1 Application Control

3.1.1 Objects

Applications can communicate by sending objects to each other. AXNserver does not verify the contents of an object. The sender has to assure that the receiver can interpret the data. An object doesn't need to have object data.
Objects of the type "messages" are special objects that are sent by an AXNserver to an AXNclient. An AXNclient cannot send objects of the type "messages".
A client is only able to receive direct or functional addressed objects, when it is connected to the AXNserver and when it has sent its inlist. A "total inlist reduction" disconnects the AXNclient from the AXNserver and no further objects can be received. One exception to this is the complete reduction of the AXNclient's inlist, sent along with a group index of 0xFF, which indicates to the AXNserver, that the corresponding AXNclient initiated a group change request. For more on this subject, please refer to "Using the group ID".

3.1.1.1 Object header

3.1.1.1.1 Header of AXCS objects

axn_AxcsObjectHeader 08S86420

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

Every object has a header (8 bytes) with the following entries:

Object Length: (2 bytes)

- Total length of the object in bytes (including the complete header with length field itself and the object data). The maximum value of "object length" is 1460.

Mode: (1 byte for type of object)

- Functional Object (0h)
- Direct Object (01h)
- Message Object (02h)
- Monitor Object (03h and 8Xh)
- Service Object (04h)
- Error (05h) (new for AXCS)
- LCM (06h) (new for AXCS)
- Trace (07h)

Target: (1 byte)

- APId of the target AXNclient.

Group-Index: (1 byte)
- Group ID of the target AXNclient(s).

Sender: (1 byte)
- APId of the sending AXNclient

ObjectId: (2 bytes)
- Unique identification number of an object

3.1.1.2 Functional objects

3.1.1.2.1 Functional addressing by inlist

axn_FunctionalObjects 08S86422

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

When an AXNclient logs on, it sends its inlist to the AXNserver. The AXNserver stores this information. The inlist contains Ids of objects, that the AXNclient wants to receive. If an AXNclient sends a functional object to a group (group_index = x), the AXNserver sends the object to all AXNclients of the group x, that have the object Id in their inlist. The AXNclient can change its inlist by sending the object INLIST_ENH (enhancement) or INLIST_RED (reduction). An AXNclient can log off by sending INLIST_RED with an empty inlist. The entry "Target" of the header is not used in this mode. The sending of INLIST_ENH can also indicate a mere change of group IDs by the client, provided that the group index of the INLIST_RED message is set to 0xFF. For more on changing group IDs at runtime, please refer to chapter "Using the group ID".

Inlist objects :
AXNserver handles 2 certain functional objects in a special way:
- INLIST_ENH :
  This object is used for announcing or increasing the inlist of an AXNclient.
- INLIST_RED :
  This object is used if an AXNclient wants to log off (number of Ids = 0) or decrease its inlist or to initiate a group change sequence (group index = 0xFF).

3.1.1.3 Direct objects

3.1.1.3.1 Direct addressing by Target ID

axn_DirectObjects 08S86424

Status:	Approved
Test Level	Subsystem Integration Test

Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

An object can be sent directly to one particular AXNclient by setting 'target' to the APId of the receiving AXNclient.

3.1.1.4 Messages

3.1.1.4.1 Messages of the basic communication layer

axn_BasicLayerMessages 08S86426

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

- TURN_ON / TURN_OFF :
  If an AXNclient logs on / off by sending INLIST_ENH / INLIST_RED, the AXNserver
  sends TURN_ON / TURN_OFF to all other AXNclients.
- INIT_MSG / INIT_CONNECTION :
  A new AXNclient tries to log on by sending an INIT_CONNECTION object to the AXNcerver.
  The AXNserver stores information about the group, the AXNclient wants to belong to (group_index).
  The AXNserver sends an INIT_MSG back to the AXNclient. (*[Reference 86416]* AXNclient logs on to AXNserver)
- ERROR_MSG (NO_TARGET) :
  If an AXNclient sends a functional object and there is no AXNclient with this group_index and this object in its inlist or the target client of a direct object does not exist, then the AXNserver sends the message NO_TARGET back to the sending AXNclient.
- DUMMY_MSG :
  To test if an AXNclient is still alive, dummy messages are sent to the AXNclient. (Alive Mechanism, pg. 21 [86440] Alive Mechanism).

3.1.1.4.2 hm_axn_SetBypassInCaseOfCommunicationInterrupt

hm_axn_SetBypassInCaseOfCommunicationInterrupt 08S86427

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06H97544,

hazard:

Communication line between logical subsystem needed for x-ray is interrupted.
measure:
The AXNServer informs the SYC to set the system in mode "Bypass fluoro"
realization:
The AXNServer implements an alive mechanism to monitor the connection between AXNServers and the local AXNClients.
The AXNServer informs all connected clients with the object TURN_OFF about the disconnected clients. The SYC determines if it was some xray relevant subsystem and set the system to the mode bypass

3.1.1.5 Monitoring

3.1.1.5.1 Support of AcsosNT Monitor

axn_Monitoring 08S86429

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

It is possible, to connect one or more monitors to the net. A monitor records all the AXCS objects on the network. It receives special monitor objects created by the AXNserver. If an AXNclient sends an AXCS object, the AXNserver creates a monitor object, which is sent to all monitors. After sending the actual object to the target AXNclient, a so called monitor send list is sent to all monitors containing all receivers of the actual object.

Object data for the monitor send list :

- Server ID of sender (DIL switch on controller in A01) (1 Byte)
- number of recipient (=N) (1 Byte)
- Recipient 1 APId (1 Byte)
- Recipient 1 Group ID (1 Byte)
- Recipient 1 DeviceId (1 Byte)
- Recipient 1 UnitId (1 Byte)
- ...
- Recipient N APId (1 Byte)
- Recipient N Group ID (1 Byte)
- Recipient N DeviceId (1 Byte)
- Recipient N UnitId (1 Byte)

The DeviceId is

- 0 if the message SEND_LIST is sent via TCP/IP and
- 1 if the message SEND_LIST is sent via the virtual network.

The UnitId is

- 0 if the message SEND_LIST is sent via the virtual network and
- AXNServerId of receiver if the message SEND_LIST is sent via TCP/IP

Functionality:

- The monitor is an AXNclient and therefore has to log on to the AXNserver like any other AXNclient
- The monitor functionality is started with the object MONITOR_ON sent to the AXNserver
- If an AXNserver receives MONITOR_ON, the object is sent to all other AXNserver and the APId of this AXNclient is added to the monitor list.
- If an AXNclient sends an object : 1) First of all a monitor object is created by setting the MSB of mode of the original object. This object is sent to all monitors (all AXNclients of the monitor list). 2) The actual object is sent. 3) Finally the object SEND_LIST (containing all receiver APIds of the actual object) is sent to all monitors
- The monitor functionality is stopped by sending MONITOR_OFF to the AXNserver which distributes this object to all other AXNserver and removes the id from the monitor list
- If a new AXNserver logs on, every AXNserver with a local monitor sends a MONITOR_ON to the new AXNserver
- Before an AXNserver with a local monitor logs off, it has to sent MONITOR_OFF to all other AXNserver
- If an AXNserver recognizes a problem with a monitor, it removes its APId from the monitor list, logs off this AXNclient and sends MONITOR_OFF to all AXNserver
- The monitor AXNclient can be used without monitor functionality to simulate other AXNclients. This means that the inlist is not always empty. Objects are then sent twice to the monitor if the monitor functionality is started (monitor object and actual object)

3.1.1.6 Service

3.1.1.6.1 Service functions

axn_ServiceInterface 08S86431

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

If an AXNserver receives TOPO_NET_RQ, TOPO_VERI_RQ or CLIENT_DATA_RQ from a local AXNclient, it sends this object to all other AXNserver. Then it sends the corresponding reply object with the desired information back to the sender.
AXCS supports 3 service objects :

- TOPO_NET_RQ returns TOPO_NET_RP from each AXNserver with :
  - AXNserver IP address
  - software version
  - number of internal AXNclients
  - APIds of internal AXNclients
- TOPO_VERI_RQ returns TOPO_VERI_RP from each AXNserver with:
  - number of known external AXNclients
  - APIds of these AXNclients
- CLIENT_DATA_RQ returns CLIENT_DATA_RP for each AXNclient of each AXNserver with :

- APId
- AliveTime
- number of inlist objects
- all inlist objects

Request of version is done with sending the object [i.e. TOPO_NET_RQ (sender= request id, mode= ACS_SERVICE, receiver= 0)].All AXNserver answer with corresponding object [i.e. TOPO_NET_RP (ipaddress, version)].

3.1.1.7 LCM (Lab Connection Monitor)

3.1.1.7.1 LCM details

axn_LCM 08S86433

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

Every AXNserver reports all its socket connections to the LCM. The LCM assures, that every AXNserver has a socket connection to every other AXNserver.

- There can be more than one LCM in the net.
- First the LCM has to log on to the AXNserver.
- After startup LCM sends LCM_ON. Its AXNserver distributes LCM_ON to all other AXNserver. LCM sends LCM_OFF before shut down.
- If an AXNserver receives LCM_ON, it sends a CONNECTION(ADD) to the LCM containing all connections to other AXNserver.
- All changes depending connections to other AXNserver are reported to the LCM using CONNECTION(ADD) or CONNECTION(REMOVE).
- If a new AXNserver logs on, the server with the active LCM sends LCM_ON after sending its inlists.
- No connection is stored by LCM until there are CONNECTION(ADD) messages from both servers : e.g. CONNECTION(ADD,S3) sent by S2 and CONNECTION(ADD,S2) sent by S3.
- A connection between two server is deleted if there is one CONNECTION(REMOVE).

3.1.1.8 Error Logging

3.1.1.8.1 Error logger interface

axn_ErrorLogger 08S86435

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

As long as no error logger has logged on, the AXNServer stores the messages for the error logger in a ring buffer.
As soon as an error logger logs on, all AXNServer are informed. All AXNServer send the stored error messages to the error logger and clear their ring buffer. From now on all the error messages are sent directly to the error logger.

- There can be more than one error logger in the net.
- The error logger has to log on to the AXNServer and send ERRLOG_ON before it can receive error messages.
- The object ERRLOG_ON is distributed to all other AXNServer.
- Before the error logger shuts down, it sends ERRLOG_OFF.
- If a new AXNServer logs on, server with a local error logger will send ERRLOG_ON to the new AXNServer.

3.1.1.8.2 axn_DesignDetailledMessageText

axn_DesignDetailledMessageText 08S86436

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S52525,

To localize the root cause of AXCS network problems in the Application Eventlog.
AXNServer should support the following internal message types:
1) AXNServer has a local or internal problem
2) AXNServer has a problem with it's external connection or it sees a problem at another AXNServer
3) AXNServer has a problem with an internal client

3.1.1.8.3 Log messages after connection is lost

axn_ErrlogHandshake 08S140392

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S139625,

The AXNserver shall provide an handshake mechanism to ensure that The ERR_INFO telegram reaches the AXNclient with the role of an errorlogger. This feature shall be compatible to the older AXN-Roots used in VCxx versions.

3.1.1.9 Trace log

3.1.1.9.1 Support of Trace Monitor

axn_TracingLogging 08S86438

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11218,

It is possible, to connect a tracer to the network.. The tracer has to connect/disconnect itself to/from the network by the object TRACELOG_ON/TRACELOG_OFF. Both objects have to be sent using the mode ACS_MODE_TRACE. Connected to the net the tracer gets all objects which are sent functionally or directly.

3.1.2 Using the group ID

3.1.2.1 General Description

axn_GroupIndexAsBitfield_CHARM208611 08S106616

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	CHARM

Some AXNclients use the group ID (group_index of the AXCS header) to send functional objects only to a subset of all AXNclients in the AXCS network (also named "Funktionskreise"). The group_index byte is used as bit field. This results in 8 different groups. AXNclients could be connected to different groups at the same time.

This means, that every AXCS object with
- mode: "functional"
- group_index: x
- id: y

is only distributed to AXNclients, that
- have the object y in their inlist and
- belong to the group x.

3.1.2.2 Setting the Group ID upon logon

axn_SettingGroupID 08S106617

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	n.a.

The group membership of an AXNclient is controlled by the INLIST_ENH object. The AXNclient notifies the AXNcerver to which group it belongs, by setting the group_index of INLIST_ENH to this value.
If the group ID is not relevant for the AXNclients, it can be ignored by setting it to a unique value in all AXCS objects.

3.1.2.3 Changing the Group ID at Runtime

axn_ChangingGroupID 08S106618

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	n.a.

The AXNserver stores this group membership and it is not possible to change it while the AXNclient is connected. This means, if an AXNclient desires to switch to another group, it has to logoff with an INLIST_RED object, which is composed of the following obligatory pieces of information:

- no_objects: 0x00
  (i.e. the reduction of the AXNclient's inlist to {})
- group_index: 0xFF
  (the second decisive criterion for filing a group change request with the hosting AXNserver)

Such an object sent indicates to the AXNserver, that a group change request has been initiated by the corresponding AXNclient. The AXNserver responds by distributing TURN_OFF objects to all connected AXNservers and thus to all affiliated AXNclients on the network as well. The TURN_OFF contains the ID of the client, that initiated the group change sequence.

All AXNservers, with exception to the AXNserver that locally hosts the AXNclient in question, interpret the seen TURN_OFF according to the specifications, i.e. all those AXNservers do remove the AXNclient that is currently conducting its group change sequence completely from their internal databases, which include:

- TurnOnTab
- APTab
- InlistTab

All affiliated AXNclients interpret the delivered TURN_OFFs in full compliance to the specifications. Therefore all connected AXNclients on the network are made aware of the absence of the corresponding client.
The AXNserver that is directly responsible for the AXNclient that requested the group change, removes the AXNclient's entire inlist from its internal database. The difference in handling this group change request, with respect to the internal databases, compared to the remaining AXNservers, can be seen below.

- TurnOnTab: still lists the client, but inlist is empty
- APTab: still lists the client as connected, no modification conducted
- InlistTab: total removal of the client

All those database manipulations of the AXNservers result in a total prevention of delivery of any functional and most of the directly addressed objects to that client.

> **Note:** Objects that are directly sent to that client from within the local virtual network (i.e. the hosting AXNserver of that client and all locally connected AXNclients, as can be seen on page *) are exempted from this restriction, since this AXNserver still does list the client as being affiliated to this server (entries about the client in APTab and TurnOnTab remain), despite having delivered TURN_OFFs to all locally connected AXNclients in the first place and despite having removed this AXNclient's inlist from the database. However, since all AXNclients do receive a TURN_OFF from the client in

question, no AXNclient shall initiate sending a message specifically addressed to that client, according to specifications. Nevertheless, events addressed to the AXNserver that are important to the AXNclients as well, like for instance the start or the logging off of another AXNclient on the network, are relayed immediately by the AXNserver to all affiliated AXNclients, including particularly the client that is in its group change sequence.

After having received its own TURN_OFF, this AXNclient is entitled to transmit an INLIST_ENH object with the following mandatory data fields for completing the group change sequence:

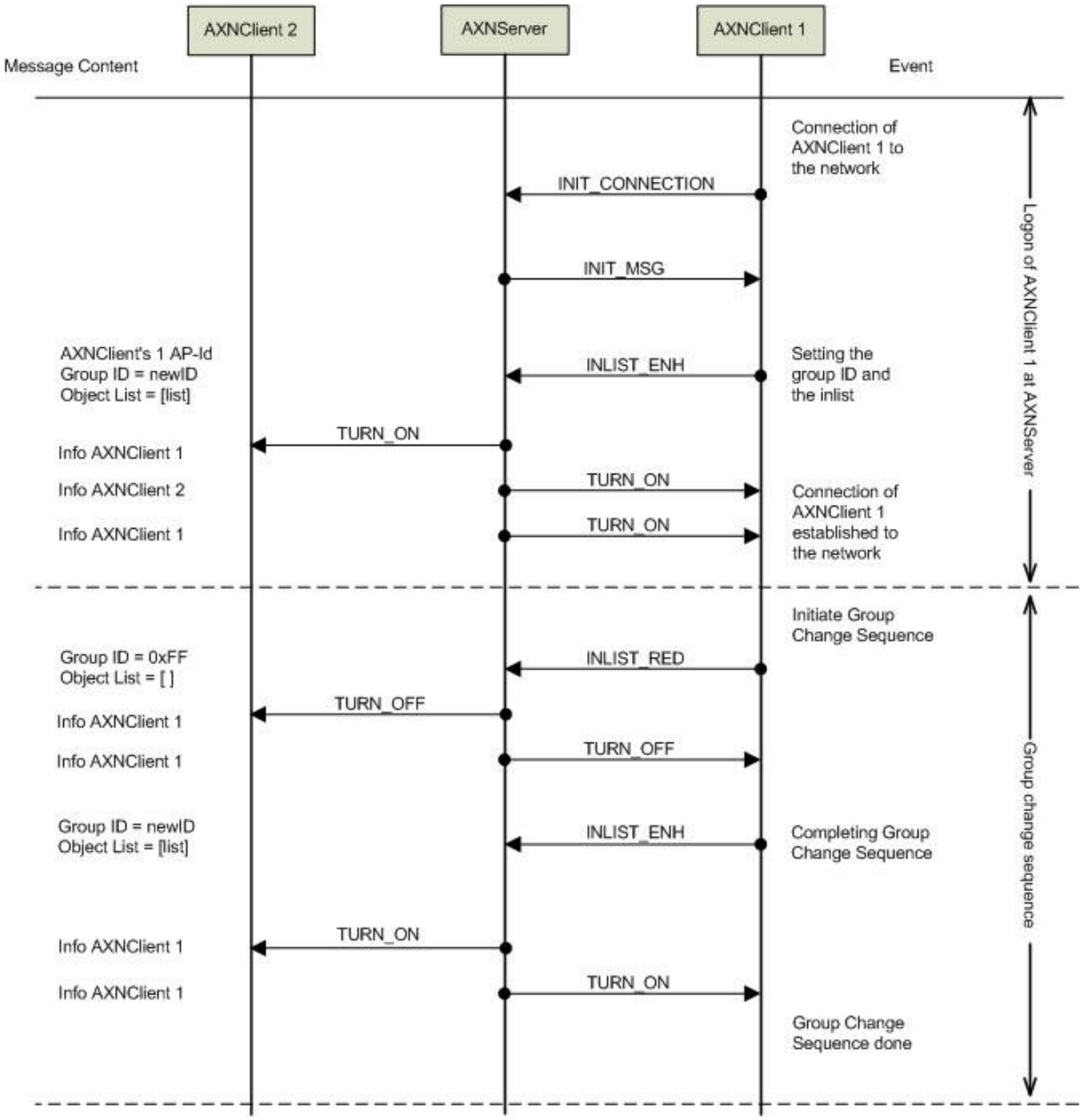
- inlist: {ObjectID_1, ObjectID_2, ... , ObjectID_n}
  The complete inlist of the client, which in the normal case is identical to the AXNclient's inlist prior to the group change sequence.
- group_index: newGroupID
  The new group the client likes to join.

The AXNserver acknowledges this INLIST_ENH object by ...

- ... dispatching TURN_ONs to all other connected AXNservers and thus to all other AXNclients on the network, too.
- ... sending the AXNclient, that is about to complete its group change sequence, its own TURN_ON.

A sequence diagram visualizing the logging on of an AXNclient to the network, as well as a group change sequence can be found below.

Note: A failure by the AXNclient to transmit the INLIST_ENH object in order to complete the group change sequence results in no further actions taken by the AXNserver. The AXNserver does not protect itself or any AXNclient in this case by means of a timer or any other error recognition mechanisms. The disconnection of the misbehaving client by the AXNserver thus is not guaranteed during this phase of the group change sequence.



3.1.3 Alive Mechanism

The identical alive mechanism is used for
- the connection between AXNclient and AXNserver and
- the connection between two AXNserver

It differs only in using different parameters. Parameters are set by AP-ID application (Client<->Server). Parameters are fixed when AXNserver<->AXNserver communication takes place.

3.1.3.1 AliveCheck AXNclient --> AXNserver

axn_AliveCheck_ServerToClient 08S86442

Status: Approved

Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

When an AXNclient logs on to the AXNserver, it also sends a counter which is multiplied by the base rate (= 200 msec), e.g. counter = 3 ==> DummyTime = 600 msec.
A part of the AXNserver (called AliveCheck) assures, that an object is sent to the AXNclient within the DummyTime. If no object is sent, AliveCheck inserts a dummy message. If the AXNclient receives an object, it resets the AliveFlag. AliveCheck tests and sets the AliveFlag.
If the AliveFlag was not reset, the AXNclient is logged off. There is no message from the AXNserver to the AXNclient. The AXNclient has to recognize itself that there is no longer a connection to the AXNserver. This is realized by a timeout when receiving an object from the AXNserver in the method recvObj(). The AXNclient is responsible for further reactions, e.g.

- delete the instance of Class CAxnClientRecv
- switch to state "NO AXCS connection" and wait minimal 2 * 200 * counter in milliseconds
- create a new instance of the Class CAxnClientRecv and log on to the AXNserver again

3.1.3.2 AliveCheck AXNserver --> AXNserver

axn_AliveCheck_ServerToServer 08S86443

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S11505,

The mechanism is exactly the same as the one between client and server. But this time there are two AliveFlags : SendAliveFlag and RecvAliveFlag. If there is no object, AliveCheck inserts a dummy message and sends it to SocketSend. SocketSend reads from its mailbox, resets the SendAliveFlag and sends the object to the other AXNserver. AliveCheck tests the SendAliveFlag and sets it.
If SocketRecv receives an object, it resets the RecvAliveFlag. AliveCheck tests and sets it.
If AliveCheck detects a flag that is still set, the connection to the other AXNserver is closed.

4 Service, Manufacturing and Installation Requirements

For installation:

- For installation procedure please refer to the platform specific installation manual
- As recommendation copy all executables and data files in the same directory
- All files are archived in a separate MSI package generated with INCO

There is no service relevant requirement to AXN.

4.1 MSI_Package

axn_ComponentDeploymentPackage 08S158298

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	none
Rearrange Reason	n.a.
Traces From	06S94915,

AXN creates INCO Database for the AXN Deployment Packages. It is named axnroot.MSI
AXN uses the INCO tool to compile the Deployment Packages

Different installation types are possible:

- AXCS Server for Windows (for product installation)
- AXCS Server for ECC (for product installation)
- AXCS Server for DDIS (for product installation
- AXCS Server for RTC (for product installatin)
- AXCS Server for Development (additional header and library files for development users)

In detail the listed files are imported to the AXN Deployment Package:

- AXCS Server for Windows mas\sources axnroot.cfg
- AXCS Server for Windows mas\production\wnt\exe\Rtccsf\bin axnroot.exe
- AXCS Server for Windows mas\sources broadcast.ini
- AXCS Server for Windows mas\production\wnt\exe\Rtccsf\bin axnrootd.exe
- AXCS Server for Windows mas\sources\Rtccsf\Axn\copy2exe\$MEDHOME\service\bin axnroot_inst.cmd
- AXCS Server for Windows mas\sources\Rtccsf\Axn\copy2exe\$MEDHOME\service\bin axnroot_uninst.cmd
- AXCS Server for Windows _comp\production\wnt\exe\Rtccsf\bin libaxnclientif.dll
- ------
- AXCS Server for ECC _comp\production\wce\exe\Rtccsf axnroot.exe
- AXCS Server for ECC _comp\production\wce\exe\Rtccsf axnrootmips.exe
- AXCS Server for ECC _comp\sources broadcast.ini

- AXCS Server for ECC _comp\production\wce\exe\Rtccsf axnrootMP277.exe
- AXCS Server for ECC _comp\production\wce\exe\Rtccsf libaxnclientif.dll
- AXCS Server for ECC _comp\production\wce\exe\Rtccsf libaxnclientifmips.dll
- AXCS Server for ECC _comp\production\wce\exe\Rtccsf libaxnclientifMP277.dll
- ------
- AXCS Server for DDIS _comp\sources broadcast.ini
- AXCS Server for DDIS _comp\production\wce\exe\Rtccsf axnroot.exe
- AXCS Server for DDIS _comp\production\wce\exe\Rtccsf axnrootmips.exe
- AXCS Server for DDIS _comp\production\wce\exe\Rtccsf libaxnclientif.dll
- AXCS Server for DDIS _comp\production\wce\exe\Rtccsf libaxnclientifmips.dll
- ------
- AXCS Server for RTC _comp\production\nto\exe\Rtccsf\bin\appl axnroot.exe
- AXCS Server for RTC _comp\sources\Rtccsf\copy2exe\etc broadcast.ini
- AXCS Server for RTC _comp\sources\Rtccsf\Axn\copy2exe\$MEDHOME\Service\bin rtc_axnroot_uninst.bat
- AXCS Server for RTC _comp\sources\Rtccsf\Axn\copy2exe\$MEDHOME\Service\bin rtc_axnroot_inst.bat
- AXCS Server for RTC _comp\production\nto\exe\Rtccsf\bin\appl libaxnclientif.so
- ------
- AXCS Server for Development _comp\sources\Rtccsf\Axn\src\axnclientif\gbl AxnClientIF.h
- AXCS Server for Development _comp\sources\Rtccsf\Axn\src\axnclientif\gbl AxnClientIFCreator.h
- AXCS Server for Development _comp\sources\global_includes\gbl axcscom.h
- AXCS Server for Development _comp\production\nto\exe\Rtccsf\bin\appl axnroot.exe
- AXCS Server for Development _comp\production\nto\exe\Rtccsf\bin\appl libaxnclientif.so
- AXCS Server for Development _comp\production\wce\exe\Rtccsf axnroot.exe
- AXCS Server for Development _comp\production\wce\exe\Rtccsf libaxnclientif.dll
- AXCS Server for Development _comp\production\wce\exe\Rtccsf axnrootmips.exe
- AXCS Server for Development _comp\production\wce\exe\Rtccsf libaxnclientifmips.dll
- AXCS Server for Development _comp\production\wnt\exe\Rtccsf\bin axnroot.exe
- AXCS Server for Development _comp\production\wnt\exe\Rtccsf\bin libaxnclientif.dll
- AXCS Server for Development _comp\production\w64\exe\Rtccsf\bin axnroot.exe
- AXCS Server for Development _comp\production\wce\exe\Rtccsf axnrootMP277.exe
- AXCS Server for Development _comp\production\wce\exe\Rtccsf libaxnclientifMP277.dll
- AXCS Server for Development _comp\production\wce\exe\Rtccsf axnrootTRZPS.exe
- AXCS Server for Development _comp\production\wce\exe\Rtccsf libaxnclientifTRZPS.dll
- AXCS Server for Development mas\sources broadcast.ini
- AXCS Server for Development mas\sources axnroot.cfg
- AXCS Server for Development mas\sources\Rtccsf\Axn\axnproxy\acs_com ACS_COM.DLL
- AXCS Server for Development mas\production\wnt\exe\Rtccsf\bin axnproxy.exe
- AXCS Server for Development mas\production\wnt\lib\global libaxnclientif.lib

- AXCS Server for Development mas\production\wce\lib\global libaxnclientif.lib
- AXCS Server for Development mas\production\wce\lib\global libaxnclientifmips.lib
- AXCS Server for Development mas\production\wce\lib\global libaxnclientifMP277.lib
- AXCS Server for Development mas\production\wce\lib\global libaxnclientifTRZPS.lib

5 Constraints

5.1 Design constraint

5.1.1 Support of several operating systems

axn_HandlingSeveralOSPlattforms 08S86460

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	Supplier

All SW-functions are developed to run with the different operation systems named under headline Purpose

5.1.2 Number of external connections

axn_NumberOfExternalConnections_CHARM139889 08S86462

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	n.a.

Actually it is necessary to handle 31 external connections at each AXNserver process. Independently from the platform it is running on.

5.1.3 Number of clients

axn_NumberOfClients 08S86464

Status:	Approved
Test Level	Subsystem Integration Test
Target Release Artis	VC14
Rearrange Operation	NEW
Rearrange Reason	Supplier

Only upto 254 clients are necessary using the ids in hexadecimal 0x01 until 0xFF.

List of Figures

List of Tables

Error! No table of figures entries found.

Table of Requirement Keys

This document has been created using MedBook VA27C.
Für dieses Dokument wurde MedBook VA27C verwendet.

- generated automatically by SAP system **P41** -

Appendix to Document: **10095636 EPH 01S 03**
Sheet generated at : **2009-02-11T11:11:40-01:00**
Originator : **SIEMENS Medical Solutions, P41**

Signatures related to this document and performed in SAP:

Meaning	system date and time	surname, given name of signee
AUTHOR	**2009-02-11T11:08:17-01:00**	**Eschenbach, Tobias**
APPROVAL	**2009-02-11T11:11:23-01:00**	**Lyncker, Andreas**
REVIEW	2009-02-11T11:08:58-01:00	Eschenbach, Tobias
REVIEW	2009-02-11T11:10:44-01:00	Lyncker, Andreas