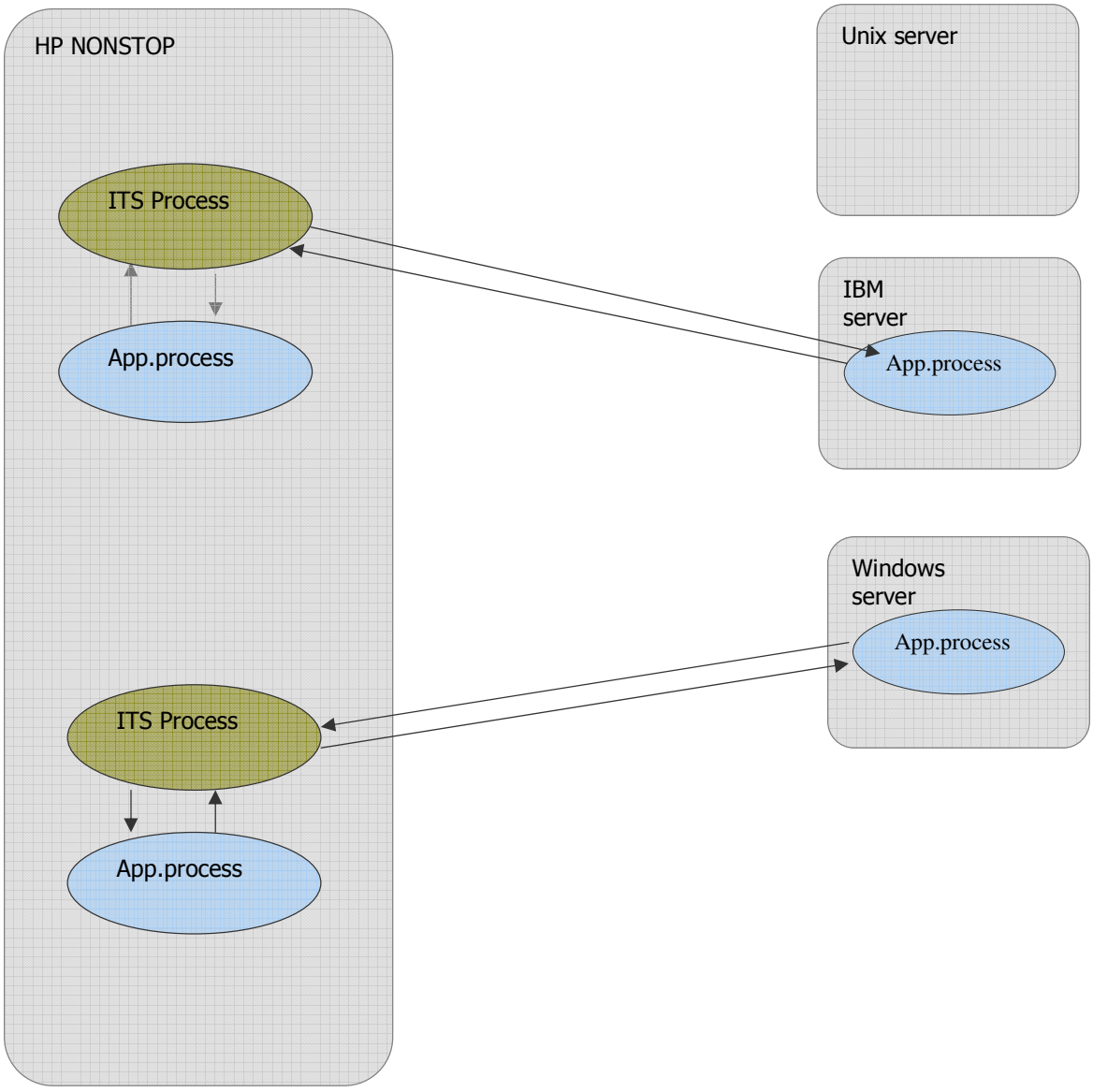
Technical description

Datum	Version	Kommentar
2006-01-10	10	Initial document

Message flows thru the ITS process



ITS handle message flow on tcp/ip socket (streaming) and $receive (message – based).

ITS is multithreaded non-stop process, implemented as watch-dog, due to unrecoverable socket error while it running on a non parallel tcp/ip stack. On a backup taker over, there might be outstanding, not served client request, they must reconnect after the backup process become primary, start a new backup and finally reinitialized its listen port.

General tree switching message type supported.

- **$receive request**: That is: when an non-stop process open, write or writeread and finally close or when the serverclass_send_ procedure is used. This switching message type is used when a process what to send and receive messages to/from a network connected system, example a windows server. A non-stop host application is in this case a requester.

- **socket-to-pathway-server request:** When a Unix application, by using socket sends a request and receives reply to/from a pathway server in to a say, cobol server in a the non-stop system.

- **socket-client-to-socket-server request:** There is a web application, sending a request to a server class in the non-stop system, there might be many instances or active server processes running, then if the request will cause the server to produce more data than what could be sent thru $receive,(due to its 60k limit) in one interaction, it could need many request/replies before all data is sent to the requesting client. If using $receive, in these cases, both the client and server needs to be context safe, the developer has to build that logic into the code. Now if using this message switch technique, this is no longer a issue. ITS will shortcut a free server to a requesting client. Until either the client or server disconnect this connection will exist.
  There is a timeout if no messages is send, if this timer expires, both server and client gets disconnected. The server has then to reinitiate and wait for next request.

ITS process must be started and its listen port opened before any request can successfully be processed by it.

Each request must use the following structure when using this switch message type.

name:	type:	description:
total_message_len	char[6]	User by ITS and the receiving to calculate message length
trans_code	char[4]	User defined transaction code. See reserved codes.
return_code	char[4]	User defined reply code. See ITS reply codes.
user_trans_key	char[16]	User defined transaction key.
pathway_mon	char[8]	Name of the sending Pathway monitor, if any. ex. "$pwy"
pathway_svr	char[16]	Name of the sending Pathway server, if any. ex. "my-svr"
its_ip_adress	char[16]	Address of this system. ex:"192.168.0.100"
its_ip_port	char[6]	ITS listen port. ITS connects to remote servers using this.
its_remote_server	char[3]	Number to remote server, from host-file, number in bold.
user_data	char[220]	User defined field.

```
# ------------------------------------------------------------------
# Hosts file for ITS. USED INTERNALLY FOR ROUTING -
# ------------------------------------------------#-------------
<192.168.0.101/001>                    #   SERVER 1
<192.168.0.102/002>                    #   SERVER 2
<192.168.0.103/003>                    #   SERVER 3
<192.168.0.104/004>                    #   SERVER 4
<192.168.0.245/008/$PWY/SRV>           #   SERVER 5
;
```

ITS, after have this message received, cretate a client socket, connect to the remote servers ip address. (ITS uses it's own listen port to connect on), send the buffer, wait for reply and finally, CALL REPLYX to reply the message back to the originator.

The package contains an example written in c, that show how to do this.

Each request must use the following structure when using this switch message type.

name:	type:	description:
total_message_len	char[6]	User by ITS and the receiving to calculate message length
trans_code	char[4]	User defined transaction code. See reserved codes.
return_code	char[4]	User defined reply code. See ITS reply codes.
pathway_mon	char[8]	Name of the receiving Pathway monitor, ex. "$pwy"
pathway_svr	char[16]	Name of the receiving Pathway server, ex. "my-svr"
user-data	char[ll]	Variable length user buffer

This is conventional request, where ITS, after receiving the client message, perform a SERVERCLASS_SEND_NW call, to the pathway monitor and server specified by the caller.
This called server class now, after received a message on its on $receive, can start process and reply.

Each request must use the following structure when using this switch message type.

name:	type:	description:
total_message_len	char[6]	User by ITS and the receiving to calculate message length
trans_code	char[4]	User defined transaction code. See reserved codes.
return_code	char[4]	User defined reply code. See ITS reply codes.
function_code	char[4]	User defined function code. See ITS function codes.
user-data	char[ll]	Variable length user buffer.

This message handling was invented due to cumbersome context handling when client and server had reason to do several interactions between them before a transaction would become complete. ITS now handle this issue.

server-specifics:
When a socket-server starts, its send all the function codes it handles to ITS, they are then saved in internal memory structures within ITS. When done sending function codes, the server send 'I-am ready-to-receive request transaction message code. Now the server wait for requests.

client-specific:
In the connect to ITS, the client set the transaction code and function code to get-me-a-free-server for-this-function-code request. ITS searches one, if match the request is processed, else client receives error message in the return code.

example:

server	**ITS**	**client**
- connect to its		
- send tree msg's		
- total length = 18 bytes (000018)		
transaction code=7010		
return code=0000		
function code=3010		
function code=3011		
function code=3012		
- send wait for request		
transaction code=7012		
return code=0000		
function code=0000		
send functions-I-can-handle --→		
"000018**7010**0000**3010**"	accept & reply	
"000018**7010**0000**3011**"	accept & reply	
"000018**7010**0000**3012**"	accept & reply	
"000018**7012**00000000"	accept & no reply	
		←- send request
		000031**7018**0000**3011**<xml>122</xml>
	find the server	
	←- send request to server	
process and send the request back -→		
000058**7018**0000**3011**<xml-data><tag1>123456</tag1></xml-data>		

Reserved transaction code:

7000	ITS internal
7002	"
7003	"
7010	Used by socket servers to send function codes
7012	Used by socket servers to send ready request
7014	ITS internal
7016	"
7018	Used by socket server and client when exchange data

Reply codes:

6000	Header len not numeric
6002	Illegal len in header
6004	Message code not numeric
6008	Message not defined
6014	Message buffer too small
6200	Max num. curr. active sock
6202	Max num. curr. $receive
6312	Undefined event state
6316	Undefined remote server
7405	Remote server, from ITSHOST file not found

PC steps:
- Unzip the file **ITS.zip** you've downloaded.
- FTP transfer all files to your nsk system in ascii mode, except these with extension .100 or .700
- FTP transfer files with extension .100 or .700 in binary mode to the nsk host.
- Make sure all files reside in the same sub volume, if not modify the PWCONF and PWSTART files, to reflect specific disc names etc.

Host steps:
- FUP ALTER ITS, CODE 700
- If your TCP/IP process is not $ZTC0, change ITSR and/or PWCONF
- Select the port for ITS to use for listen, default is 3400.
- Home term for ITS is set to $VHS, make sure this virtual home-term-process exists.

Currently the only supported operating system is nonstop kernel nsk.
Porting to Unix and Linux is being considered.

ITS is not freeware, its free to evaluate for maximum six(6) month after the evaluation period, a license fee must be paid.
The fee is based on the number of concurrent sessions.

ITS uses a license key file for this, where it holds the system name , expire date and number of session.