TGF

InterFacility Communications Technical Document

1.1

Prepared for:

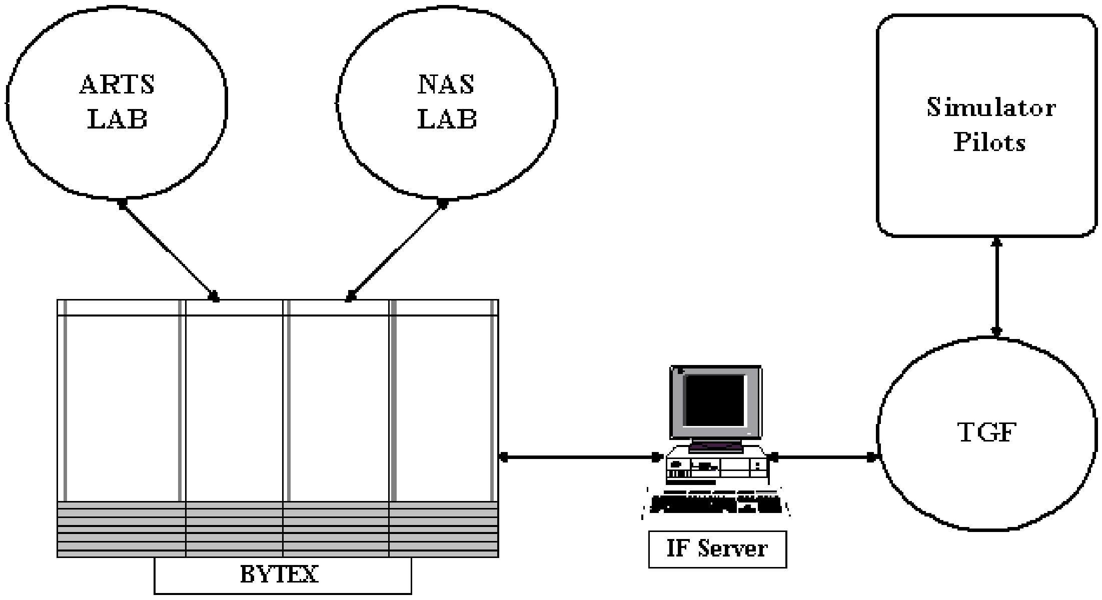
Dan Warburton
ACB-860
Simulation Group (ACB-860)
Real & Virtual Division (ACB-800)

Federal Aviation Administration
William J. Hughes Technical Center
Atlantic City, NJ 08405

Apr. 02, 1996

Table of Contents

1.0 What Is IF?



InterFacility (IF) communications allows 2 adjacent air traffic control (ATC) facilities to communicate with each other. IF is simulated in TGF to allow whatever lab we are responding to (ARTS/NAS) to simulate communications with one or more adjacent facilities.

Target Generation Facility has successfully simulated NAS-NAS & NAS-ARTS with TGF as a NAS facility. TGF has also simulated ARTS-ARTS with TGF as both a NAS center and several adjacent ARTS facilities handling messages from an ARTS lab.

The purpose of this simulation is to implement IF messages to exchange flight plans and hand offs, as well as test and response messages.

2.0 Message Types Implemented

(Messages are sent only once unless otherwise indicated.)

2.1 DA - Data Accept

A DA message is sent in response to another message to indicate that the original message has been received and accepted. It does not necessarily mean that the data in the message was any good or that TGF will do anything with the message. A DA message is sent in response to the following messages that TGF currently handles: DM, FP, TA, TB, TI.

2.2 DM - Departure Message

A DM is sent by an ARTS facility to indicate that an aircraft has departed. DMs are sent from ARTS to NAS only. Currently TGF will accept a DM message and send a DA in response. No error checking or validation is done and TGF will not do anything with the data in the message. TGF does not initiate DM messages.

2.3 DR - Data Rejection

A DR message is sent in response to another message to indicate that the original message was rejected. Rejection can occur for a number of reasons, depending on what type of message was rejected. Primarily, though, a message is rejected for incorrect format. Currently TGF does not send DR messages, although it will accept them. TGF does not do anything with DR messages once it has accepted the message except log that it has arrived.

2.4 DT - Data Test (Response)

A DT message is sent in response to a TR message. The TR/DT combination is used as a test message/response set to verify that IF communication is possible.

2.5 DX - Data retransmit

A DX message is sent in response to another message to indicate that the original message needs to be resent. This can occur for a number of reasons, depending on what type of message was rejected. Primarily, though, a DX is sent because of incomplete data. Currently TGF does not send DX messages, although it will accept them. TGF will not respond to the message once it is accepted, except to log that a DX message has arrived.

2.6 FP - Flight Plan

IF flight plans contain information about an aircraft to be simulated, including departure/arrival time and beacon code. IF flight plans are currently only sent when TGF is simulating a NAS facility and communicating to an ARTS facility. An FP should be responded to with a DA or the FP should be sent again. At the start of a simulation TGF will only send the plans for flights leaving in the next *x* minutes. As the simulation runs FPs will be sent for an aircraft when it's start time becomes *x* minutes away. Currently *x* = 20. FP messages are sent up to 3 times if a response is not received.

2.7 TA - Transfer Accept

A TA message is sent by a facility when it is ready to accept a hand-off from another facility. A TA can not be sent until a TI, and possibly several TUs, have been received. A TA is responded to by a DA. When a facility receives a DA in response to a TA the aircraft in question comes under the control of the facility issuing the TA. The aircraft must also have been among one of the aircraft that a flight plan was sent out for regardless of which facility is handing off and there must be at least 1 ARTS facility involved.

2.8 TB - Terminate Beacon

A TB message is sent by an ARTS facility when it terminates a flight. TGF does not send TBs, but will accept them, log them and respond with a DA. A TB is usually sent for one of two reasons: a flight has been completed, or the simulation is being terminated and TGF is restarting. During the simulation TGF would be recycling but the ARTS lab would terminate all flights.

2.9 TG - TGF specific hand-off

A TG message is a TGF specific message, this it is not a standard message and not covered by official reference material. A TG message is sent from a NAS lab to TGF simulating a NAS lab to indicate a hand-off. When TGF receives a TG message a "contact controller" text message is displayed on the sim pilot station. A TG should be responded to with a DA message.

2.10 TI - Transfer Initiate

A TI message is sent from a site that is ready to hand off an aircraft to an adjacent site. A TI message is responded to with a DA message. TI messages are used with ARTS-ARTS & ARTS-NAS. TGF can send and receive TI messages. TI messages are sent up to 3 times if a response is not received.

2.11 TR - Test data

A TR message is sent to test that IF communications are functioning properly. A DT message is expected in response. The TR/DT combination is used as a test message/response set to verify that IF communication is possible.

2.12 TU - Transfer Update

A TU message is sent to update the status of an aircraft that a TI message has already been sent for. After a TI is sent, Tus are sent at regular intervals until the receiving site sends a TA message to indicate that it is ready to accept the aircraft. TU messages receive no response and TGF will ignore any TU messages it receives. TU messages are used with ARTS-ARTS & ARTS-NAS.

3.0 Set Up:

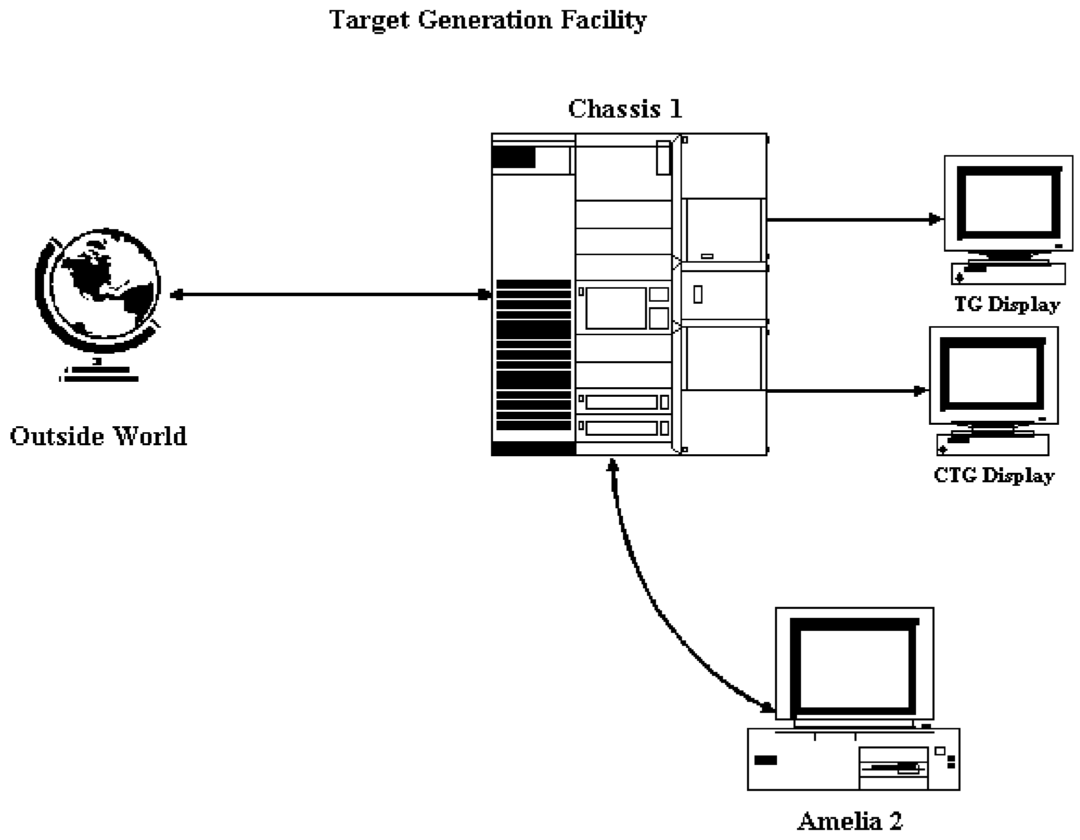
3.1 Old version

Before the upgrade to using the Bytex and a straight connection to do InterFacility communication TGF connected to other labs by using modems and a Motorola box. In order to go back to this style of communications the following must be done. In the tge startup script there should be no reference to any of the current IF setup, i.e. IFmixy, IFcookie, etc. The line

```
setFpProcessor("tge")
```

should be put in. No reference to the new style of IF should be made in the ctg startup scripts either.
This method will soon be phased out, but is currently still supported.

3.2 New Version



In order to have IF communications between labs, several things must be done.

The file "if_adapt.dat" must exist in the current scenario sub-directory of the "ready' area on the server being used (amelia2 for chassis 1 or orville for chassis 2). The format is explained in the file itself. Certain data concerning the sites being simulated needs to be known. Most important is the site IDs, IF will not work properly without this information. Also, the ID for TGF must come first. If simulating multiple sites the main site TGF will be communicating with must come second followed by any additional sites TGF will be simulating.

The startup scripts used when running any simulation must be modified when using IF. In addition to the normal changes to the startup scripts, the files "tge" & "ctg" must have the following changes:
tge:

Under Misc Flags, TG_do_if needs to be set. It should be set to 2 for chassis 1 or to 1 for chassis 2. That is not a typo, and hopefully it will be reversed in the near future.

The IF server needs to be set. Currently the only options are cookie or mixy. To set to cookie use the command IFcookie, for mixy - IFmixy. This command should appear just before the call to tge at the end of the script.
ctg:

The ctg start up script no longer needs to be modified unless the user wishes to use a ctg other than the main one for IF activity. By default, ctg1-3 & ctg2-5 will come up initialized for IF communications and any IF activity should be done from there. However, if the user wishes to use another ctg for debugging, testing, etc. that ctg needs to be initialized with the data in the file "if_adapt.dat" (see above). This is done by calling the function "TGU_CreateIfAdapt()" with a string containing the complete path to the file "if_adapt.dat"

as the only argument. This function can either be entered manually at the ctg desired, or added to the start up script and all ctgs will be initialized.

When the chassis starts up, tg needs to be monitored to ensure that the IF connection is made to whatever server is selected. The following 3 messages should appear before the ctg boots:

```
TEF_if_initl: Status: Connection Completed
about to spawn IF_tx
about to spawn IF_rx
```

If an error message of the form IF_rx error = some large negative number, follows the above than there is a problem with the IF server. A possible solution is to rlogin to the server being used, cd to the scripts area and execute a script called KillServ. After this, reboot the chassis. If the problem repeats then there is something wrong with the IF server you are attempting to use, and a different one should be selected.

The above will connect TGF to the bytex. The lab TGF is working with will also have to be connected to the bytex and our IF server. Usually the other lab handles this. If they do not the following steps need to be taken:

rlogin to the bytex as TGF,

gdisp the IF server being used ex. gdisp cook1

if there are any connections they must be terminated using either pterm or pdisc

pcon TGFs IF server with the port the other lab is using, this will have to be supplied by the other lab

gdisp the IF server to verify that the connection was made

When the "Exercise Ready" message appears on the tge console it will be possible to send and receive test messages with the other lab. Depending on the level of IF communication desired for a particular simulation one or both of the labs should send a TR message to the other. A DT message should automatically be sent back to the sending lab. The format for a TR message send from TGF is as follows:

sendTR(message, sending site, receiving site, scenario number)

message is 8 to 20 characters in double quotes

sending site is the 3 character ID of the sending site, usually, but not always TGF. In double quotes.

receiving site is the 3 character ID of the receiving site, usually but not always the other lab. In double quotes.

scenario number is an integer, 2 for chassis 1, 1 for chassis 2.

example: sendTR("test test","ZCB","BOA",2)

Messages should be sent from ctg1-3 or 2-2 depending on which chassis is being used.

The test messages should be monitored on the above ctg and the tg screens. To verify the messages contents on tg set IF_do_log=2 and on ctg set TGJ_do_log=2. If sending flight plans be sure to set these flags back to 0 before starting the simulation or the excessive I/O to the console will cause the tg to crash.

ARTS labs automatically send TRs every 30 seconds, NAS does not. TGF can send a TR at any time during a simulation but it is not necessary once IF communication has been verified.

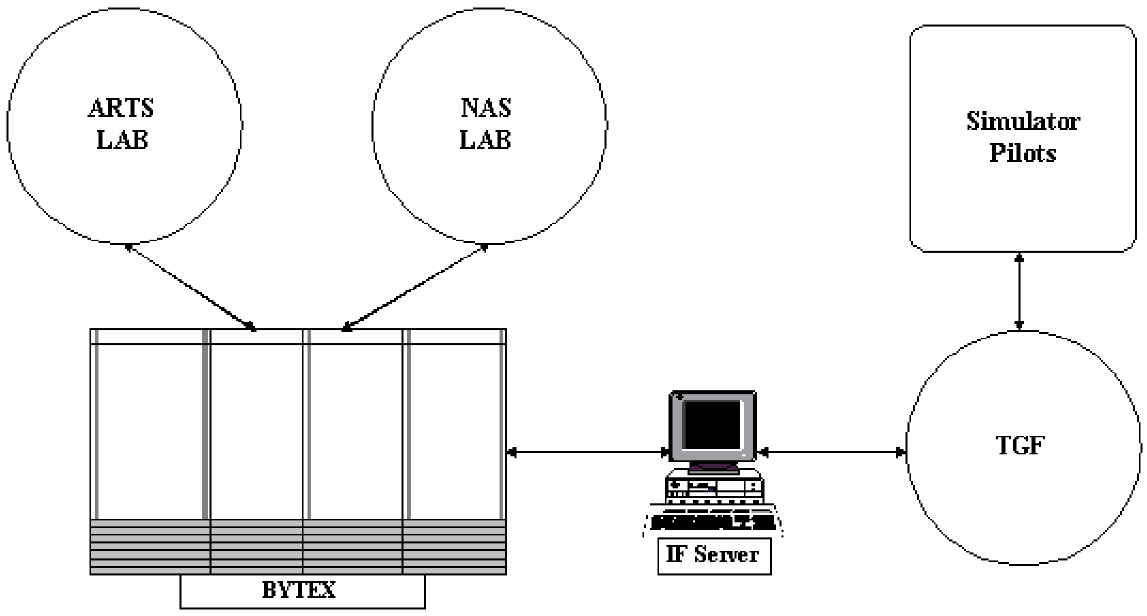
4.0 Hardware:

4.1 Chassis

4.2 IF Server (cookie/mixy/etc.)

4.3 Bytex

4.4 Other Lab (NAS/ARTS)



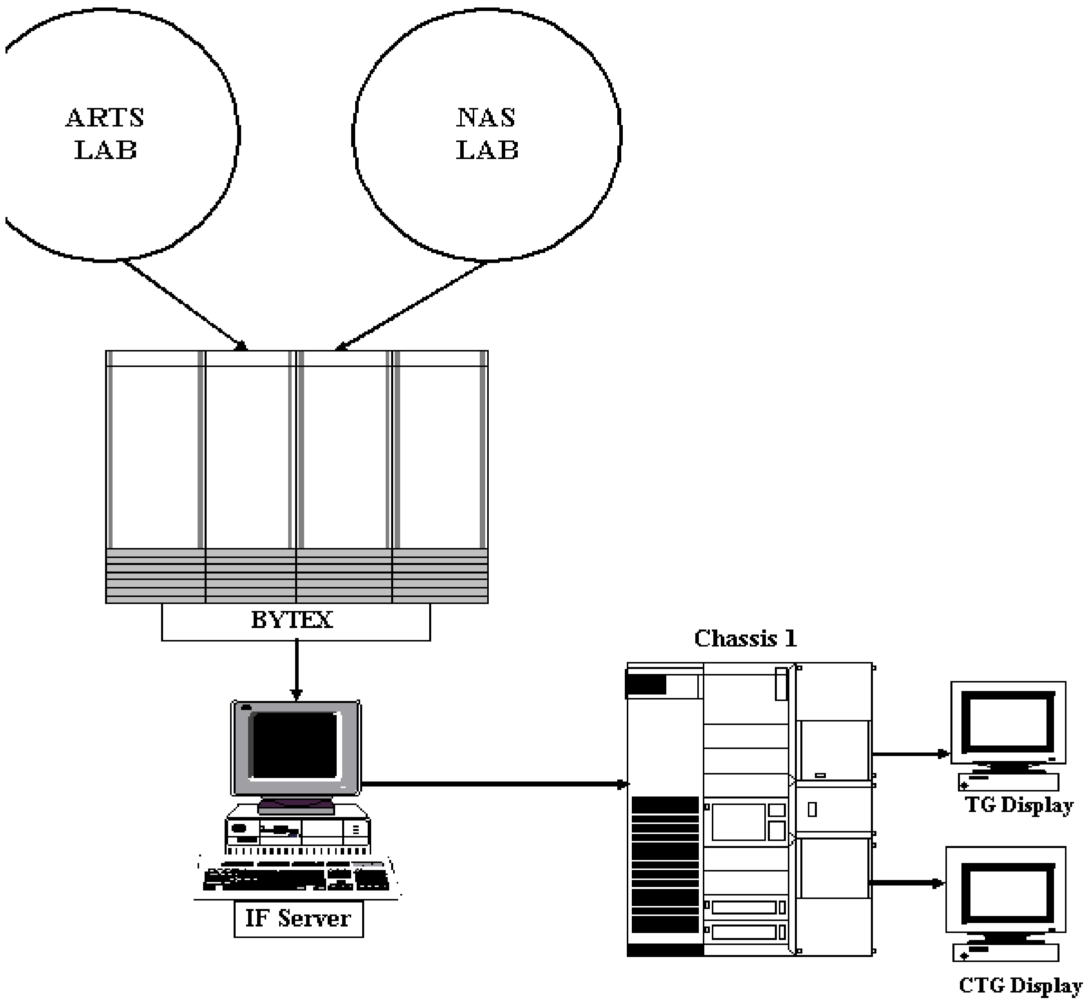
5.0 Software:

5.1 Overview:

Incoming:

When a message comes into the tg along IF lines, it is read in from a socket as a stream of characters. This happens in tef/if_rx.c. The stream is expected to be equal to the size of an IF_MSG struct. If it is, the data is stored in an IF_MSG struct. The message size is byte swapped since we are reading the data in from an Intel architecture machine (generally Cookie or Mixy). The message is then stored in an MHDR struct and passed along the backplane to the ctg for decoding.

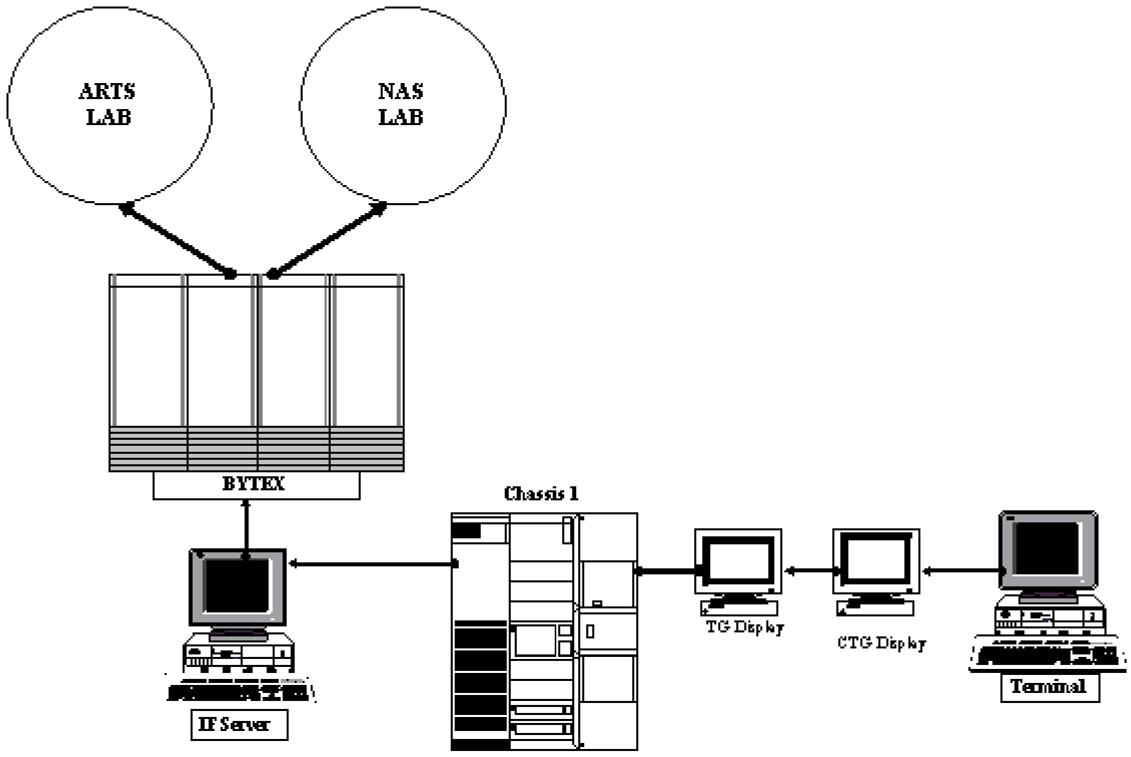
Incoming Message



The MHDR is passed to tgj/TGJ_if_in.c via the TGF_pipe where the message type is determined. The message is copied from the MHDR into an IF_MSG struct where it is translated from EBCDIC to ASCII. The LRC bytes are checked here and removed. At this point the ASCII message and the MHDR pointer are passed to tgj/IfMsgProc.c to be processed.

The first step in IfMsgProc is to determine the IF message type. After this the MHDR pointer and the ASCII message are passed to a message specific function that will handle the message accordingly. Either reply to it, record data from it, or ignore it. See Appendix A for the actions to be taken for specific messages.

Outgoing Message



There are two types of outgoing messages: reply messages and those originated by TGF. The only real difference is in the memory allocation. Reply messages use the same MHDR as the incoming message. Messages originated in TGF have to allocate memory for an MHDR.

tgj/IfMsgSend.c creates outbound messages. There is a specific function for each message type. Every message starts with a header that varies in length and format based on the types of sites being simulated. Immediately following the header is the LRC. This is a calculated field used to insure that data is not corrupted during transmission. There is also one at the end of the message. The rest of the message format depends on the message type being send. Details can be found in Appendix B. Messages are created in ASCII and translated to EBCDIC as the last step before they are shipped to the tg via the backplane. The LRC characters are inserted into the message during this step. The LRC is based on the characters EBCDIC values. There are a few exceptions to this standard and they are noted in the Appendix. After translation to EBCDIC the message is attached to an MHDR and sent to tgj/TGJ_if_out.c

TGJ_if_out.c simply writes the MHDR to the backplane as a character stream. It is read from the backplane into the tg by tef/if_tx.c. if_tx.c byte swaps the size field and sends the stream out to the IF server via a socket connection.

5.2 Code:

5.2.1 /ctg/tgj/ - (entire directory is support, but IF specific is:)

IfMsgProc.c - processes incoming messages

(note: For the below functions the first argument, char*, is the ASCII translation of the actual IF message; the second argument, MHDR*, is a pointer to the MHDR struct passed down from the tg.)

int getRefMsgNum(char*, MHDR*) - This function parses the incoming message and removes its reference number. This number is returned.

int MsgProcess(char*, MHDR*) - This function scans the incoming ASCII message to determine the message type and passes it to the appropriate function to handle that message. Nothing is returned.

int processAM(char*, MHDR*) - Message not used. This function simply logs that an AM message was received. Nothing is returned.

int processCX(char*, MHDR*) - Message not used. This function simply logs that a CX message was received. Nothing is returned.

int processDA(char*, MHDR*) - This function will log that a DA message was received and remove the message that was responded to from the resend list. Nothing is returned.

int processDL(char*, MHDR*) - Message not used. This function simply logs that a DL message was received and sends a DA in response. Nothing is returned.

int processDM(char*, MHDR*) - Message not used. This function simply logs that a DM message was received and sends a DA in response. Nothing is returned.

int processDR(char*, MHDR*) - Message not used. This function simply logs that a DR message was received. Nothing is returned.

int processDT(char*, MHDR*) - This function logs that a DT message has been received. Nothing is returned.

int processDX(char*, MHDR*) - This function logs that a DT message has been received. Nothing is returned.

int processFP(char*, MHDR*) - This function logs that a DT message has been received and sends a DA in response. Nothing is returned.

int processHO(char*, MHDR*) - Message not used. This function simply logs that a HO message was received. Nothing is returned.

int processTA(char*, MHDR*) - This function processes a TA message from another site. After verifying it and extracting the aircraft ID being referenced it passes the information along to TGF. A DA message is sent to the site that sent the TA message. Nothing is returned.

int processTB(char*, MHDR*) - Message not used. This function simply logs that a TB message was received and returns a DA. Nothing is returned.

int processTG(char*, MHDR*) - This function processes a TG message. The aircraft ID is extracted and a message is passed to the simulator pilot station. Upon success a 0 is returned, else nothing is returned.

int processTI(char*, MHDR*) - This function processes a TI message from another site. After verifying it and extracting the aircraft ID being referenced it passes the information along to TGF. A DA message is sent to the site that sent the TI. Nothing is returned.

int processTM(char*, MHDR*) - Message not used. This function simply logs that a TM message was received. Nothing is returned.

int processTP(char*, MHDR*) - Message not used. This function simply logs that a TP message was received. Nothing is returned.

int processTR(char*, MHDR*) - This function logs that a TR message was received and responds with a DT message. Nothing is returned.

int processTS(char*, MHDR*) - Message not used. This function simply logs that a TS message was received. Nothing is returned.

int processTU(char*, MHDR*) - This function logs that a TU message was received. Currently all processing in this function is ignored. Nothing is returned.

int processUNK(char*) - This function logs that an unknown message has been received. Nothing is returned.

void setSites(char*, char*, int, char*) - This function sets the IDs of the sending and receiving sites. The first 2 parameters are changed.

IfMsgSend.c - composes outgoing messages. There are additional support functions in this file that are not listed below.

(Note: There are various levels of log messages that can be printed out by these functions. The level is controlled by the flag "TGJ_do_log". Under normal circumstances it should be set to 0.)

int ebcdicAndLrcIt(char*, int) - This function takes the ASCII version of an outgoing IF message, inserts the LRC prepare/LRC/EOM characters and converts the message to EBCDIC. The first argument is the message to be converted, the second argument is the length of the header. The total length of the message is returned.

void IFSendTA(char*, char*, char*, int) - This function is used to send a TA message to another facility. The first argument is the 3 character ID of the initiating site, the one this message is sent to. The second argument is the 3 character ID of the facility receiving the hand off and sending this message. Both these IDs must appear in the file "if_adapt.dat" (see above). The third argument is the aircraft ID of the plane being handed off. The fourth argument is the scenario ID, 2 for chassis 1, 1 for chassis 2. Nothing is returned.

void IFSendTI(AC*, double, double, double, double) - This function is used to send a TI message to another facility. The first argument is a pointer to the AC struct of the aircraft to be handed off. The next four arguments are the X & Y coordinates and velocities of the aircraft. This function will allocate memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed in the message queue. TI messages are sent up to 3 times if not acknowledged before failing. Nothing is returned.

void IFSendTU(AC*, double, double, double, double) - This function is used to send a TU message to another facility. The first argument is a pointer to the AC struct of the aircraft in hand off. The next four arguments are the X & Y coordinates and velocities of the aircraft. This function will allocate memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed in the message queue. TI messages are sent up to 3 times if not acknowledged before failing. Nothing is returned.

int MakeAsciiHdr(char*, char*, char*, int) - This function is used to create the header portion of an outgoing message. It is separate from the message creation functions since in most cases the message formats are the same regardless of what type of facility sends/receives them. The only difference is in the header format and size. The first argument is where the completed header will be stored. The second argument is the 3 char ID of the site sending this message. The third argument is the 3 character ID of the receiving facility. The fourth argument is the scenario ID, 2 for chassis 1, 1 for chassis 2.. The message number is returned.

int sendDA(char*, MHDR*) - This function is used to send a DA message in response to one of several possible messages sent from the other site. Part of the incoming message is used in the DA message. The first argument is the ASCII translation of the IF message. The second argument is the MHDR struct passed down from the tg. This function will allocate the memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed at the top o the queue and go out immediately. There are no retries for a DA message. Nothing is returned.

int sendDT(Buf, mhdr_ptr) - This function is used to send a DT message in response to a TR message sent by the other facility. The header from the incoming message is used as part of this message. The first argument is the ASCII translation of the IF message. The second argument is the MHDR struct passed down from the tg. This function will allocate the memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed at the top o the queue and go out immediately. There are no retries for a DT message. Nothing is returned.

int sendTR(char[], char*, char*, int) - This function is used to send a TR message from a TGF simulation site to another simulation site (may be TGF simulated or another lab). The first argument is a character string of variable length depending on who is receiving, generally 8 - 20 characters. The second and third arguments are the 3 character IDs of the sending and receiving sites, respectively. These sites must be listed in the file "if_adapt.dat", (see above). The last argument is the scenario id, this is usually 2. This function will allocate the memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed at the top o the queue and go out immediately. There are no retries for a TR message. 0 is returned.

int SendFP(char*, char*, char*, int, int) - This function will send a single flight plan from TGF to the main receiving facility defined in the file "if_adapt.dat". The general use of this function is to be called as part of a loop that is reading the file "dl_fp" for a particular scenario. The first argument is the flight plan. The second and third arguments are the 3 character Ids of the sending and receiving sites. These sites must be listed in the file "if_adapt.dat", (see above). The fourth argument is the delay in seconds before the flight plan is to be sent. The fifth argument is the scenario ID. This function will allocate memory for, and create the MHDR used to pass the message along the backplane. After the MHDR is set it will be placed in the message queue. FP messages are sent up to 3 times if they are not acknowledged before failing. 0 is returned.

TGJ_if_in.c - translates messages received from tg

int TGJ_if_in() - This function continually reads the raw character stream from the tg via the backplane and stores it in an MHDR. Then, based on the MHDR->ID calls the correct function to process the stream. Nothing is returned.

void TGJ_if_process(int, MHDR*) - This function takes the incoming MHDR, translates the IF message portion from EBCDIC to ASCII and passes the MHDR and the ASCII message onto IfMsgProc. The first argument is irrelevant. The second argument is the MHDR containing the incoming message.

int TGJ_IF_msg_to_ascii(unsigned char*, int, int, unsigned char*) - This function converts the incoming IF message from EBCDIC to ASCII and verifies that the LRCs are correct. The first argument is the IF message in EBCDIC. The second argument is the length of the message. the third argument is the length of the message header. The fourth argument is the IF message translated to ASCII. 1 is returned on failure, 0 is returned if translation is completed.

TGJ_if_out.c - prepares messages to go to tg

int TGJ_get_fp_from_file(char*, FILE*, int, int, int) - This function reads a flight plan from a file and formats it for processing. This includes removal of bad trailing characters and the insertion of the relative universal time. The first argument is the actual flight plan, the second argument is not used, the third and fourth arguments are the Universal Time hours and minutes at the start of the simulation. and the fifth argument is the number of seconds after the start of the simulation that the flight plan should be sent. A negative number is returned upon success, 0 is returned if universal time couldn't be inserted and a 1 is returned at end of file.

int TGJ_process_fp_file(int) - This function opens the file containing IF flight plans, if it exists, sets the universal time at the start of the simulation and loops through the flight plan file sending a flight plan at a time to SendFP() so they can be sent to the receiving site. The argument is the scenario ID. Nothing is returned.

int TGJ_send_ifmsg_totg(MHDR*) - This function takes an MHDR containing an IF message and maps it to a CTG_GBL struct. It then fills in some fields in the CTG_GBL and sends the message to the tg via the backplane. The argument is the MHDR to be sent. Nothing is returned.

TGJ_if_util.c - utility functions

void TGJ_send_IF_TI(AC*, char*, char*) - This function calculates the location and magnetic declination for a particular aircraft and then causes a TI to be sent for it. The first argument is the AC struct for the aircraft being handed off. The second and third arguments are the 3 character IDs for the sites sending and receiving the hand-off, respectively. Nothing is returned.

void TGJ_send_IF_TU(AC*, char*, char*) - This function calculates the location and magnetic declination for a particular aircraft and then causes a TU to be sent for it. The argument is the AC struct for the aircraft being handed off. Nothing is returned.

void TGJ_TA_Received(char*, char*, int) - This function notifies TGF that a TA message has been received for a particular aircraft. The first argument is the ACID of the aircraft being handed of. The second argument is the 3 characters ID of the site that sent the message. The third argument is the scenario ID.

void TGJ_TI_DA_recv(AC*) - This function notifies TGF that a DA has been received for a particular aircraft that a TI had been sent out for. This causes TGF to stop sending TU messages for that aircraft. The argument is the AC struct for the aircraft that has been accepted by the other site.

void TGJ_TI_Received(char*, char*, char*, int) - This function notifies TGF that a TI message has been received for a particular aircraft. The first argument is the ACID of the aircraft being handed of. The second and third arguments are the 3 characters IDs of the site that sent the message and the site that received the message respectively. The fourth argument is the scenario ID.

5.2.2 /ctg/tgu/

IF_adapt.c - reads adaptation file "if_adapt.dat"

int TGU_CreateIfAdapt(char*) - This function reads in the adaptation file, if_adapt.dat, and sets various fields that are to be used throughout the simulation. The argument is a complete path, including filename, to the version of if_adapt.dat that is to be used for a particular simulation. After the file is opened the name, type, and possibly other data about each site is read in, stored and echoed with log messages. 0 is returned.

Below are short utility functions that are stored here so tg and ctg can access them.

int setInterFacPort(int) - This function sets the port number that IF will use. Nothing is returned.

int getInterFacPort() - Returns the IF port number.

int setInterFacHost(char *) - This function stores the IP address of the IF server to be used. The argument is the IP address of the server. Nothing is returned.

int setIFcookie() - This function sets cookie as the IF server. Nothing is returned.

int setIFmixy() - This function sets mixy as the IF server. Nothing is returned.

char *getInterFacHost() - This function returns the IP address of the IF server.

char *get_TGF_fac_type() - This function returns the type of facility TGF is simulating, NAS or ARTS.

char *getPrimeFPsend() - This function returns the 3 character ID of the primary site in the simulation. Currently always TGF.

char *getPrimeFPrec() - This function returns the 3 character ID of the primary site receiving flight plans from TGF if flight plans are used. Otherwise returns the ID of the primary site TGF is communicating with.

int getHeaderSize(char*, char*) - This function returns the header size to be used in an outgoing message based on the types of facilities the IF message is going between. The first argument is the 3 character ID of the sending site and the second argument is the 3 character ID of the receiving site. The header size is returned.

int get_fac_type(char*) - This function determines the type of facility based on the 3 character ID. The argument is the 3 character ID. The facility type is returned.

5.2.3 /tg/tef/

if_initl.c - initializes IF lines

TEF_spawn_IF_procs(char[], int) - This function will spawn a process for a particular scenario. The first argument is the last 2 letters of the process to be spawned (TX or RX). The second argument is the IF socket to be used. Nothing is returned.

int KillIf() - This function will connect to the IF server and kill the IF process running on it. Nothing is returned.

int TEF_if_initl() - This function will attempt to create a socket and establish a connection through the defined IF port to the defined IF host. If successful it will spawn the IF receive and transmit processes. Next an MHDR will be created for the IF periodic and it will be written to the backplane. Nothing is returned.

int IForville() - This function sets orville up as the IF server and sets some flags. 0 is returned

int IFbooboo() - This function sets booboo up as the IF server and sets some flags. 0 is returned. booboo is no longer a reliable IF server, check it's availability before using it.

int IFcookie() - This function sets cookie up as the IF server and sets some flags. 0 is returned.

int IFmixy() - This function sets mixy up as the IF server and sets some flags. 0 is returned

if_tx.c - transmits messages from TGF to outside world

int TEF_if_write_all(char*, int, int) - This function will write the x characters (argument 1) from the character stream (argument 0) to the socket (argument 2). 0 is returned.

int TEF_if_tx(int) - This function will continually read the backplane for messages from the ctg. After some processing the message will be sent to TEF_if_write_all() to be sent to the IF server. The argument is the socket for TEF_if_write_all() to write to. 0 is returned.

if_rx.c - receives messages from outside world

void print_msg_inascii(IF_MSG*) - This function takes the MHDR just read off the IF line and prints the message portion of it in ASCII. If the IF_do_log flag is greater than 2 it will print the message one character per line listing the EBCDIC value, the ASCII value and the ASCII character. The argument is the IF_MSG struct read from the IF server. Nothing is returned.

void TEF_process_msg(IF_MSG*) - This process takes the IF_MSG struct read from the IF server and makes it part of an MHDR struct. The MHDR is then written to the ctg via the backplane. The argument is the IF_MSG struct read from the IF server. Nothing is returned.

bool TEF_rx_read(int, char*, int) - This function is not used.

int TEF_if_rx(int) - This function will continually read from the socket passed in as the argument for IF data. If data is read it is stored in an IF_MSG struct and passed to TEF_process_msg. 1 is returned.

5.2.4 /tgf/include/

if.h - header file

This file contains many #defines used for IF processing.

It also contains the following structs:
CallbackStruct;
IF_MSG;
IF_ADAPT;
IF_SEND_FP;

Appendix A

Message formats

Formats of the messages TGF sends.

Note: These messages are built in ASCII and translated to EBCDIC before being transmitted. Some of the characters in these messages are not meant to be translated. LRC prepare, LRC (0xB3), and EOM (0xB1) characters are not meant to be translated. Other exceptions are specifically noted.)

DA:

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (DA)
a space
3 byte numeric CID (TCID/ECID depending on site types)
a space
incoming message header
a space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

DT:

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (DT)
a space
incoming message header
a space
TR message message (a TR message contains a field with a text message, echo it here)
space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

FP:

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (FP)
a space
the flight plan (read in from a file)
space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

TA:

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (TA)
a space
receiving ARTS site ID
a space
3 byte computer ID
a space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

TI:

(Note: NAS-MD-640 Appendix A lists this incorrectly. The correct format can be found in the text portion of NAS-MD-631 A3.06)

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (TI)
a space
3 character output routing - generally receiving ARTS site ID
a space
3 byte computer ID
a space
8 byte binary field - 2 bytes per: X & Y coordinates, X & Y velocities (* Not translated to EBCDIC *)
a space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

TR:

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (TR)
a space
3 character ID of the receiving site
a space
1 character clear weather symbol (`)
a 4 to 20 character text message - some systems require a minimum of 8 characters
a space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

TU: (same as TI except message type field)

(Note: NAS-MD-640 Appendix A lists this incorrectly. The correct format can be found in the text portion of NAS-MD-631 A3.06)

message header (4-13 characters, length & format based on types of sites involved)
a space
0xB3 (LRC prepare character)
calculated LRC character
a space
message type (TU)
a space
3 character output routing - generally receiving ARTS site ID
a space
3 byte computer ID
a space
8 byte binary field - 2 bytes per: X & Y coordinates, X & Y velocities (* Not translated to EBCDIC *)
a space
0xB3 (LRC prepare character)
calculated LRC character
0xB1 (EOM character)

Appendix B

Message Header Formats

NAS to NAS

Header Length of 4 Bytes:

1 Byte - last character of sending sites 3 character ID
3 Byte Message Number - incremented with each message

NAS to ARTS / ARTS to NAS

Header Length of 10 Bytes:

3 Byte Sending Site ID
4 Byte Time - currently 0000, field not used
3 Byte Message Number - incremented with each message

ARTS to ARTS

Header Length of 13 Bytes:

3 Byte Sending Site ID
4 Byte Time - currently 0000, field not used
3 Byte Message Number - incremented with each message
3 Byte Receiving Site ID

Appendix C

Start Up Checklist

(Section 3.1 covers this in detail, this section is meant as a quick do-list.)

1) Modify startup scripts (tge, ctg, eie, cd2)

tge: One of the lines that specifies an IF server (IFcookie, IFmixy, etc.) must be uncommented.
TG_do_if must be set
ctg: No changes are needed However TGU_CreateIfAdapt() may be called with the path to the proper if_adapt.dat file as the argument.
eie: No changes should be needed.
cd2: No changes should be needed.

2) Make connection to Bytex

Generally this will be handled by the other lab. If not, see section 3.1 for details on connecting.

3) Verify file if_adapt.dat

Look in the ready directory for the scenario being run. Verify that the file is there and the correct site IDs are listed.

4) Verify flight plan file

If running with ARTS be sure that there is a flight plan file called dl_fp in the ready directory for the scenario being run.

5) Reboot chassis

6) Start ECO

Unless otherwise instructed take all the default settings and select the proper scenario.

7) Send test message

After seeing the Exercise Ready message on the tge console and confirming that the other lab is ready for IF communications, send a TR message. A DT message should be received in reply.