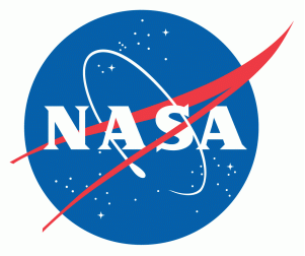
Research Traffic Management Advisor (rTMA)

Up-level Final Report

Prepared for:

NASA/Ames Research Center

Moffett Field, CA 94035-0001



22 September 2013

Prepared by:

PDRC-IADS Research Team

Table of Contents

List of Figures and Tables

1 Identification

The purpose of this report is to describe the up-leveled Research Traffic Management Advisor (rTMA) system and to provide information for researchers who develop and/or run this system.

Throughout this document, when the Time Based Flow Management (TBFM) system terminology is used to refer to the Federal Aviation Administration (FAA) operational system. When the rTMA system is mentioned, it refers to the NASA derived system. NOTE: The 4.0.x briefing material provided to the Traffic Management Coordinators (TMCs) states that in 4.0.x the system name changed from Traffic Management Advisor (TMA) to TBFM. This name change has now been propagated to all Graphical User Interfaces (GUIs), tools, scripts, and documents. Therefore, 'TMA' is no longer used when referring to the deployed metering system.

A number of the sections of this document were also uploaded to sub-pages within the rTMA home on NASA Ames Confluence site. The rTMA home on confluence can be found here: https://atmjira.arc.nasa.gov:9443/conf/display/ctas/rTMA+Home .

The work described in this report is funded through the PDRC-IADS NASA Research Announcement (NRA) NNA11AC17C. Mosaic ATM, Inc. is the prime contractor for this work.

1.1 Document Overview

This document describes the PDRC development, evaluation, and transition strategy.

Section 2 provides the minimal set of information required to initialize rTMA.

Section 3 provides a summary of the major changes to TBFM since the last rTMA up-level.

Section 4 provides a description of the changes required to FAA TBFM 4.0.x to run efficiently in the NASA environment.

Section 5 describes how to build, configure, and run the system.

Section 6 describes how to debug the rTMA system.

Section 7 describes the rTMA testing process and results.

Section 8 discusses rTMA debugging techniques, tools, and convenience scripts.

Section 9 discusses the feasibility of upgrading to a 64-bit version of rTMA.

Appendix A lists the referenced documents and supplementary material.

2 I am in a hurry to run rTMA! What do I need to know?

This section is designed for the busy researcher who doesn't need to know all the details of rTMA and just wants the minimal set of information required to run the rTMA system.

The basic steps for running rTMA are as follows:

1) Obtain the latest source code from Git.
2) Obtain the appropriate adaptation.
3) Determine which processes you want to run.
4) Create/modify an emulator start-up script and initialize the system.

5) Set appropriate environment variables
6) Load shared memory
7) Initialize the system

The following sections discuss these steps in more detail. Section 5 also discusses building, configuring, and running rTMA in greater detail.

2.1 rTMA 4.00.07 source Code in Git/Stash

The rTMA source code can be found in the Git configuration management system in the RTMA project, rtma_4.00.07 repository. To view the rTMA project in Stash go to https://atmjira.arc.nasa.gov:7443/stash/projects/RTMA/repos/rtma_4.00.07/browse

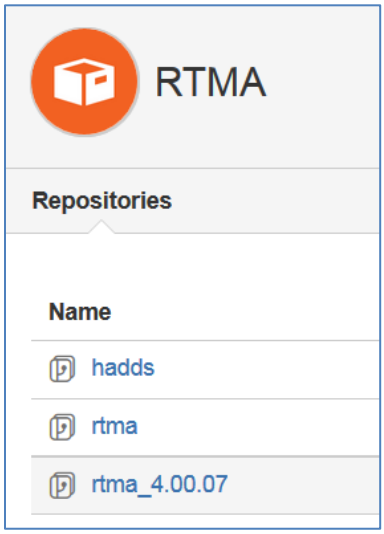


Figure 1 - rTMA 4.00.07 in Stash

Be sure to follow the appropriate procedure to clone the rTMA Git repository from the central server to your own local repository. No clone scripts exists at this time to automate this process.

2.2 Obtain the Appropriate Adaptation

As with CTAS, the adaptation required to run rTMA is not stored in the same repository in Git. In some cases the adaptation may exist in the Git adaptation repository already, or, more likely than not, you will need to obtain it directly from another source (e.g. Clearcase).

The FAA TBFM adaptations that correspond to the 4.0.x TBFM release will be named accordingly. The typical FAA naming convention is <Center>_T<Software>_<Adaptation Version>.

For example, ZFW_T4.0.8_1.0 is a ZFW Center release for 4.0.8 with adaptation version 1.0.

FAA TBFM adaptation is replicated via the clearcase multi-site feed. The clearcase config spec required to obtain the adaptation version mention above is as follows:

element CHECKEDOUT

element * ZFW_T4.0.8_1.0

element * …/main/LATEST

Once you obtain the adaptation, the rTMA system expects it to reside in the apps/adaptation/<Center> folder structure.

2.3 Determine Which Processes and Configuration to Run

A good approach to running the rTMA system is to start with a minimal set of processes and then gradually add in new components until you reach the desired configuration.

A basic startup of rTMA includes the following programs:

- HDIF (Hadds Data Interface—for connect to Host/ERAM)
- ISM (Input Source Manager)
- CM (Communications Manager)
- RA/TS (Route Analyzer, which calls the Trajectory Synthesizer)
- DP (Dynamic Planner)
- GUIs (Planview GUI and/or Timeline GUI)

After initializing the process above, you may want to add in one or more ADIF (ARTS/STARS) data feeds, the WDPD process, and other processes (e.g., CAP).

In addition to determining the appropriate process to run, the user also needs to determine the Terminal Radar Approach Control (TRACON) groups to initialize. A TRACON group is the name TBFM gives to the set of processes that perform scheduling in the system. The TRACON group is typically composed of one or more RA/TS, DP, TGUI, and/or PGUI. For example, Atlanta Center has three available TRACON groups; Atlanta (A80), Charlotte (CLT), and the departure system (EDC). The user can also elect to run one, two, or all three of these TRACON groups within a single rTMA configuration.

2.4 Create/Modify an Emulator Start-up Script

The easiest way run rTMA is to leverage a proven/working emulator example and modify it to fit your needs. If you have emulator scripts that have been used in a previous rTMA version, only minor (if any) changes should be required to allow the scripts to run with the 4.0.x version of the software.

Example start-up emulator files that initialize the system can be found on the Ames rTMA confluence page.

2.5 Set Appropriate Environment Variables

The environment variables required to initialize rTMA are typically placed inside a wrapper script (e.g., runemu.sh). A new environment variable is required in 4.0.x: USER_DEFINED_RELTOP. This new environment variable should point to the top level “apps” directory, for example:

- export USER_DEFINED_RELTOP=…/rtma/apps

2.6 Load Shared Memory

rTMA expects for the system adaptation to be loaded into shared memory. The rTMA tool that accomplishes this task is the shared adaptation manger (SAM) utility. SAM can be used to load memory, view shared memory allocation, and clear shared memory.

The version of SAM used to load the system must match the software version the user desires to run. The adaptation being loaded into shared memory must match the version of software.

It is generally a good practice to include the deletion and loading of shared memory at the beginning of the system start-up scripts so that the start-up process will be more robust to changes in adaptation or version numbers.

For more on working with shared memory, see Section 5.3.

2.7 Initialize the System

Once the steps mentioned above have been completed, you are now ready to initialize the system. This is typically done by executing a wrapper script on an emulator file, for example:

```
./runemu.sh zfw.d10.edc.emu
```

Alternatively, the user can load all the required environment variables inside of their shell and execute the emulator tool directly on the emulator script.

Debugging start-up scripts are often complicated by the fact that rTMA runtime output is not verbose. This is especially true during initialization. If you find yourself frustrated with the lack of rTMA debugging output, please see Section 6 of this document for a number of rTMA debugging tips.

3 Summary of Major Changes Since TBFM Version 3.12

This section describes major changes to the TBFM system since the last NASA merge (version 3.12).

The 4.0.x TBFM software architecture is illustrated in Figure 2.

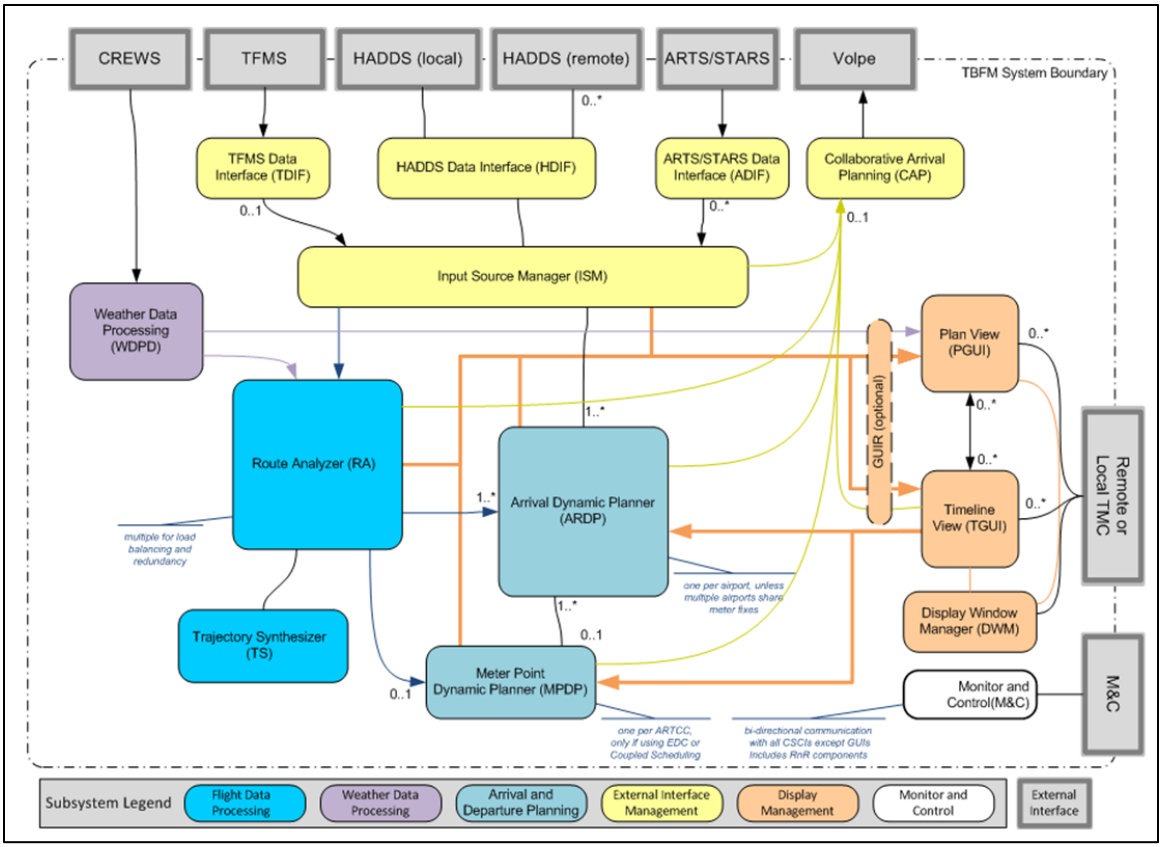


Figure 2: TBFM Functional Architecture (per FAA's System/Subsystem Design Document)

A description of each Computer Software Configuration Item (CSCI) can be found in the System/Subsystem Design Document (SSDD)[3]. A more detailed (software-level) description can be found in the Software Requirements Design Document (SRDD) for each process [4, 5, 8, 9-17]. For example, the CM SRDD[15] contains detailed information on the software design of the Communications Manager.

The only new CSCI since the previous NASA rTMA version (3.12) is the Dynamic Window Manager (DWM) component[13]. The ETMS Data Interface (EDIF) was changed to TFMS Data Interface (TDIF) to reflect the name change of its source (Traffic Flow Management System), and Arrival Dynamic Planner (ARDP) is now distinguished from the Meter Point Dynamic Planner (MPDP) in the architecture rather than having a single DP component.

3.1 Changes from a Software Metrics Perspective

It is often helpful to summarize significant software changes by using industry-standard software metrics. Approximately 15,900 lines of source code that were added to TBFM 4.0x since the prior version NASA rTMA software (version 3.12). The measure of new source code was taken from SCI's 'Understand' tool www.scitools.com. These source code metrics do not include any source code from the new Flight Deck (FDK) Monitor and Control (M&C, which NASA does not use) or any Extensible Markup Language (XML) files.

A positive indicator from the source code metrics was that despite the addition of new source code, the average complexity and average functions/methods per class both went down. This is

not typical; the metrics generally tend to grow slowly worse as a system grows larger. In addition, the overall source code documentation percentage went up almost 9%, which is another positive indicator from the software metrics perspective.

Many developers will want to know the processes impacted by the new source code. In the strictest sense, all TBFM source code files have changed, given a new header at the top of the source files. However, substantial source code changes (more than header) were made to approximately 28% of all unique source code files. The top four preexisting processes with software changes were the PGUI, TGUI, DP, and CM. Many of the changes to the GUI processes were required due to the operational software needing to support old (big endian) and new (little endian) format between adjacent facilities. Therefore, some of the 4.0.x changes are operational artifacts unlikely to benefit researchers, or anyone else for that matter after the nationwide transition to the new architecture.

Deeper inspection of the source code did reveal evidence of refactoring. For instance, some functions/methods were combined, generalized, or renamed, and additional documentation was provided in several legacy areas of the source.

3.2 Hardware Changes

Substantial changes have occurred to the operational TBFM hardware. In fact, it can be said that the new hardware is the biggest change to the TBFM 4.0.x system. Given that these changes do not typically affect research personnel, they will not be discussed at length in this document. For an in-depth discussion of these changes, see the TBFM TechOps Release 4 briefing materials[2].

One interesting (and modern) change to hardware is the use of virtualized processes for some TBFM components (using virtual machines and hypervisor as a machine manager). For more information on the operating system upgrade, see TBFM SSDD[3] or the Re-architecture DDR material[7].

3.3 Flexible Scheduling (TMA Version 3.13)

The focus of version 3.13 of the TBFM system was a capability called flexible scheduling. This TBFM change was motivated by a user-reported desire for a scheduling technique that utilize a range of achievable Estimated Times of Arrivals (ETAs) and Scheduled Times of Arrivals (STAs) and to allow aircraft to take advantage of unused airspace capacity. This change came about as means to help alleviate reported scheduling problems in TBFM, which can be described as three separate issues: 1. Partial Slots, 2. Bounce, and 3. Fit problems. For more information on these problems, see Lockheed Martin's SIG report TBFM004[1].

As described in LMCO's SIG documentation[1], three alternative solutions for this problem were evaluated. The solution that was chosen is called 'general scheduling flexibility'. The generalized scheduling flexibility alternative seeks to reduce delay and unused capacity by allowing adaptable amounts of tolerance in the flight scheduling. These tolerances, which are specified as input parameters (e.g., aircraft class, stream, metering deconfliction point), allow for a greater range of times that a flight may be scheduled to. This approach relies upon the ability to utilize scheduling flexibility within the TBFM scheduling processes and to deconflict ETAs/ STAs using a minimum separation standard, which is smaller than the TBFM configured spacing.

New adaptation files (e.g., flex_values.xml), system messages (e.g., FLEX_OPTIONS), and Dynamic Planner logic was added to the TBFM system to enable this capability.

For more information, see the Flexible Scheduling Detailed Design Review (DDR) slides[6].

3.4 Operating System Upgrade (RHEL 6.1+)

The operational TBFM system has now been upgraded to Intel architecture. The upgrade process was quite extensive due to the large amount of equipment located at all deployed TBFM locations. The 3.14 version of TBFM was the last Solaris/SPARC-based TBFM system. This system was intentionally (virtually) identical to the initial 4.0 TBFM system to allow for exchange of data between neighboring systems as the upgrade commenced across all deployed locations. As of now, all TBFM centers have been upgraded to Intel (TBFM 4.0.x).

The operating system utilized by TBFM is best described as a mix of Red Hat Enterprise Linux (RHEL) 6.1 and 6.2 environment. It is unclear whether the FAA plans to continue to run with this hybrid version of RHEL or upgrade to a later version. For the purposes of rTMA testing, a system was constructed with exactly the same patch level as the operational TBFM system. In addition to this system, a Centos 6.2 and a Centos 6.4 were tested. No noticeable differences to the system capability or software performance were noted between these versions.

The software changes required to upgrade to the new OS were, in the end, relatively minor. The integration testing challenges that were resolved, by evidence of TBFM problem reports, were more significant. These included fixes that helped to 'harden' the system as users found undocumented quick keys that degraded system performance.

3.5 Dynamic Window Manager

A substantial change to the TBFM system's handling of graphical displays was introduced with the DWM capability. DWM capability is intended to give operational users more flexibility to change TBFM displays without the need to restart the system. Prior to this change, it was often necessary for the Traffic Manager to work with local site TechOps personnel in order to change the graphical displays, which required restarting the system using the Monitor and Control.

With DWM, upon initialization of the TBFM system the TMC utilizes the DWM interface to manually launch the required GUIs. A small DWM icon is presented to the users to allow for GUI initialization. When the user selects the DWM icon, a menu is presented based upon the site adapted configurations. New configurations can be created and saved for later recall/loading. Figure 3 illustrates options that are available for Albuquerque and its relevant adjacent center displays.

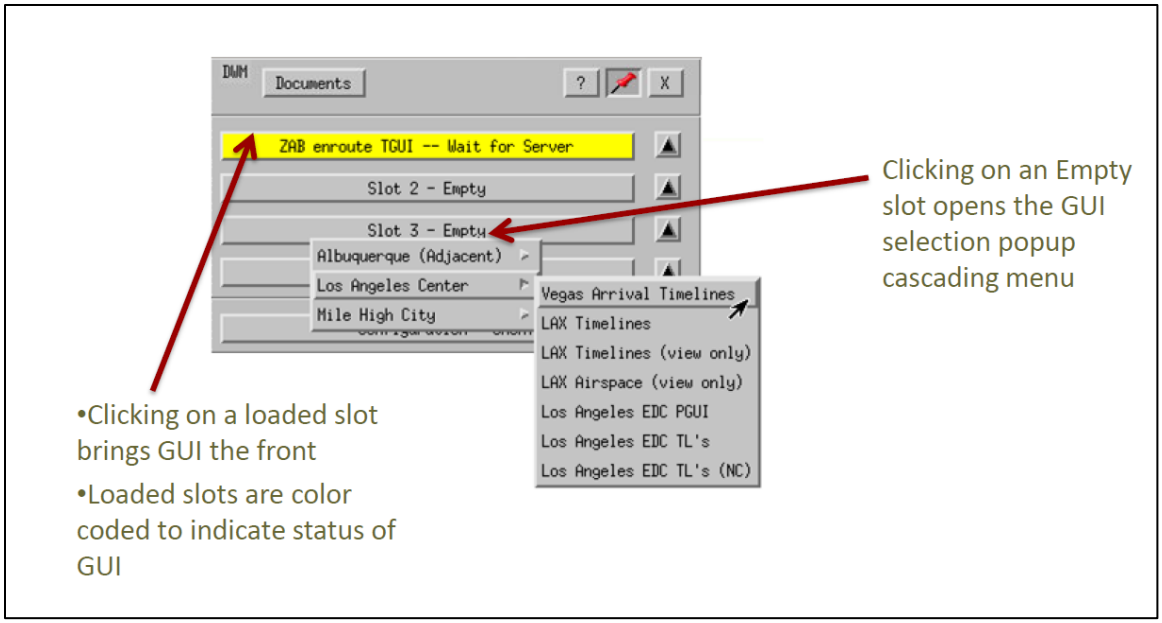


Figure 3: DWM Menu Allows a Range of GUI Selections

Once the selected GUI is initialized, the DWM icon runs in such a way that it no longer obstructs the view of the graphical display, as seen in Figure 4.

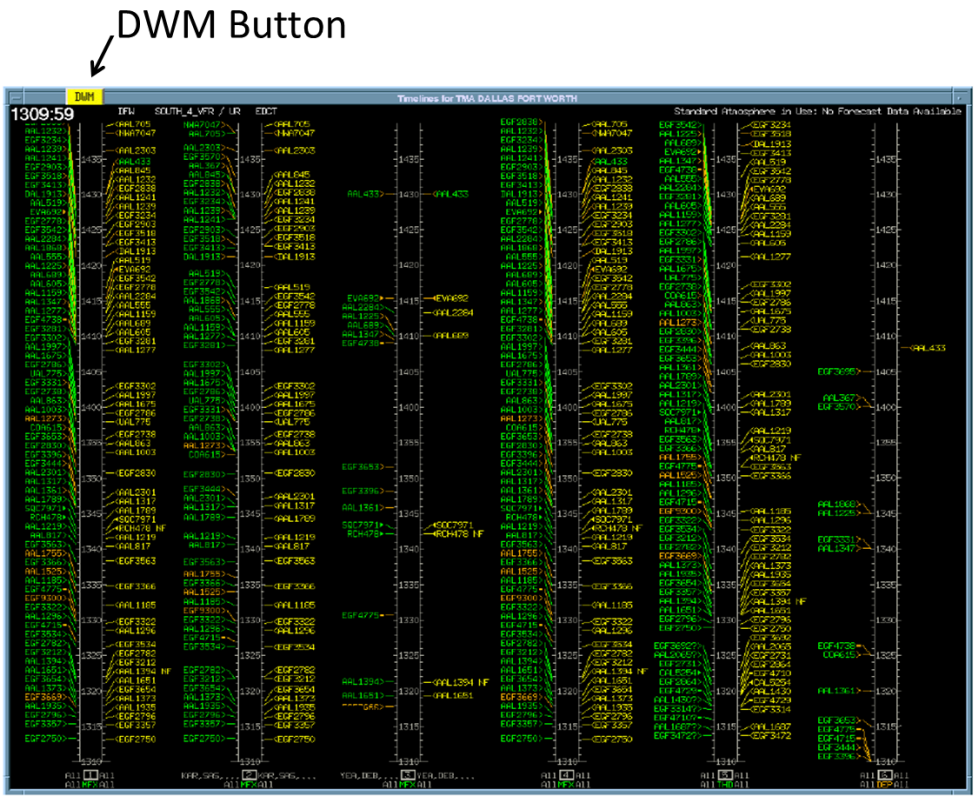


Figure 4: DWM Running in Background Once GUI is Initialized

If NASA researchers desire to run with DWM, it is important to understand that each facility has its own DWM customization files. These files are not replicated to NASA, although examples have been created for ZFW that can be modified for other sites. Similar to user preference files, these customization files are generally maintained by local site personnel.

New configuration files were introduced for DWM. These files are in XML format, and the general format is described in the System Administrator's Manual[18]. The minimal set of configuration files required for DWM is as follows:

- CenterSocket.xml—defines the hostname and port numbers for the CM process (primary and backup)
- WorkstationConfig.xml—defines the location of display workstations for the site and organizes them within highlighting groups
- StartMenu.xml—defines items that a user may assign to slots on the DWM taskbar
- DWMConfig.xml—default configuration to start local and/or remote GUIs.

For more information on running DWM, see Section 5.7 of this document.

3.6 Monitor and Control—FlightDeck

A new monitor and control system is now in use for the operational TBFM system. The system is called FlightDeck. FlightDeck is also in use by other LMCO projects. Figure 5 shows a graphical screen shot of the FlightDeck capability.

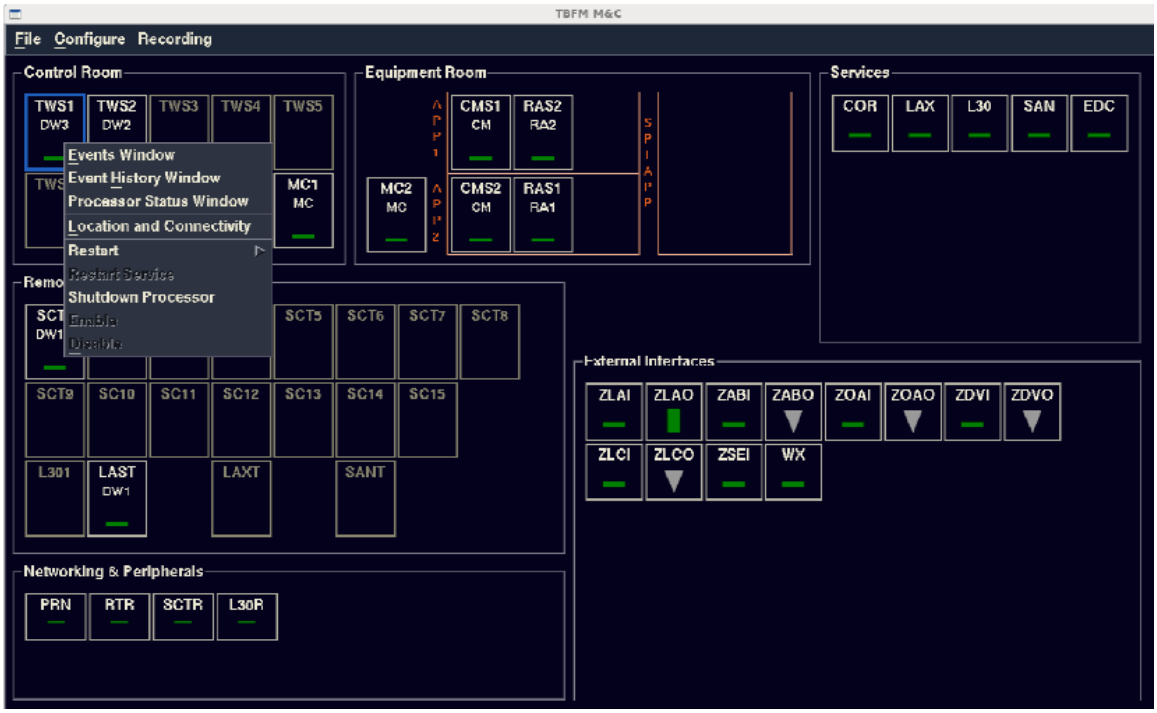


Figure 5: TBFM Monitor and Control User Interface

Thus far NASA researchers have not required a monitor and control interface and instead utilize the emulator (emu) interface to start and stop system processes. Therefore this change has little effect on most research activity. To learn more about the new MnC, see the Re-architecture DDR slides[7], the MnC SRS[4], or the TBFM TechOps briefing[2].

3.7 New System Configuration Files

3.7.1 DataLocation.xml

The DataLocation.xml file allows the user to change the expected location of system files. This is an important file which is required to run TBFM software in the NASA research environment. The required modifications to run in the NASA research environment have been made to this file in the Git repository. For more information on these changes, see Section 4.9.

3.7.2 TimeLineSetup.xml

As part of the change to DWM, a new file was introduced called TimeLineSetup.xml. The contents of this file likely rarely need modification, and it is currently placed in the appropriate location in the timeline directory inside the rTMA Git repository.

3.7.3 ScreenSettings.xml

The ScreenSettings.xml file contains the valid screen sizes the system will consider. This file was introduced to assist with DWM initialization considering the variety of screen sizes that were deployed with the new TBFM architecture. If the user initializes a display with a resolution that is not listed in this file, then the default screen resolution specified in this file will be utilized by the system. Therefore, it is important that all of the possible screen resolutions desired are listed in this file.

3.8 General Changes to Graphical User Interfaces

Some changes were made to the PGUI and TGUI processing. The majority of changes were to allow communication with DWM, including the ability to initialize and determine the status of the GUI. Other changes were described as fixing 'inconsistencies' in the GUIs, as well as to change the way the GUIs receive initial interface status. The PGUI's registration for weather files was also changed.

3.9 WDPD Processing

The WDPD processing was changed to handle critical versus non-critical processes differently. The critical process is the RA, while the PGUI and GUI router are non-critical. These changes have minimal effect to the NASA research environment, but did require some minor source code updates in order to enable the weather file distribution (Section 4.8 has more details).

3.10 Code Refactoring

This activity is not listed in any of the formal materials, but code inspection revealed a number of changes to the source code that appear to be geared toward code refactoring. In many cases additional documentation was provided, functions were consolidated, and variable names changes or other refactoring activities occurred.

4 rTMA Up-level Change Summary

This section provides some notes on the up-level process, including required changes to run TBFM efficiently in the NASA research environment.

In general, the up-level process was relatively straightforward. Most changes that were encountered were described in the system documentation and/or source code. The less straightforward areas of the up-level activity were due to complexities of running the system without DWM which was not possible without software changes, subtle changes in the way some processes initialize with the emulator, changes to environment (OS and/or new variables), differences in operations versus development environment paths, or side-effects of TBFM source code refactoring.

The 4.0.x system was the TBFM 're-architecture', and as such, changes were made to a number of TBFM processes. It is believed that future TBFM versions will be more isolated to a specific CSCI. Thus, the hope is that any future up-levels will be even more straightforward than the 4.00.07 rTMA up-level.

4.1 Upgrade Criteria

This section describes the design criteria, or up-level rules, used by the PDRC team during software development.

An important rule was to change as few lines of code as possible from the TBFM baseline. While this approach may mean more configuration and/or adaptation updates, it helps to preserve the traceability of NASA-derived changes against the FAA TBFM base.

Another rule was to make as few changes to the NASA start-up procedures as possible. Even simple changes to start-up procedures can be costly to NASA projects in training time; therefore, if at all possible the approach taken was to allow the re-use of previous scripts and tools developed for rTMA use.

A rule of the up-level as specified in the Statement of Work (SOW) was to preserve prior rTMA capability that existed on the 'rTMA base', while incorporating the latest TBFM capability.

4.2 Target TBFM Version

The rTMA up-level effort began on version 4.00.05. As the work progressed, the TBFM system went through two updates and stabilized on version 4.00.07, which the team used as the target TBFM version for the up-level. At the time the rTMA software merge was complete, tested, and ready for utilization, all sites were on 4.00.08.

4.3 Target Operating System

An important goal for the up-level activity was to determine the acceptable operating systems for TBFM capability. The process the PDRC team used was to mimic the TBFM operating system as exactly as possible in a baseline rTMA system. The baseline system, which is a mixture of RHEL 6.1 and 6.2 files, was used for comparison against later OS versions to ensure that there was no degradation of capability. In order to match the operational environment, the same packages that existed on the operational system (as obtained from FAA second-level support) were installed on the baseline rTMA.

After successful testing of the near-exact operating system, the rTMA baseline system was then compared to the performance of Centos 6.2 and Centos 6.4, which demonstrated no degradation of capability. Therefore, both the Centos 6.2 and 6.4 operating systems are viewed as providing an acceptable (if not superior) environment for running rTMA.

For the Centos 6.4 operating system test, the team began with the default install of Centos as provided by NTX. This install is believed to closely mimic NASA Ames Centos 6.4 environment. There were several additional Linux packages required by this system for rTMA to function:

- glibc-devel.i686
- zlib-devel-1.2.3-29.el6.i686
- libstdc++-devel-4.4.7-3.el6.i686
- openmotif-devel-2.3.3-5.el6_3.i686
- libXmu-devel-1.1.1-2.el6.i686
- libXt-devel-1.1.3-1.el6.i686
- libX11-devel-1.5.0-4.el6.i686
- libxml2-devel-2.7.6-12.el6_4.1.i686

4.4 Running rTMA with or without DWM

While DWM is useful for operational personnel, research personnel already have the ability to change their displays at will. Thus, there is no underlying research motivation for a change in the way the graphical processes are managed. While evaluating the DWM capability, the question became 'how much overhead, or process change, is required to run DWM in the research environment?' Some requirements and/or limitations of running DWM in the research environment are as follows:

- Multiple machines are needed to run a typical system given that DWM clears and loads shared memory, which can degrade other processes running on the machine. Many developers are accustomed to running rTMA on a single machine, so this would represent a change.
- DWM requires manual interaction to start up a GUI. In the research environment, like that which exists at NTX, this is not a desirable capability for daily display of GUIs to East and West tower, TRACON, and Center locations.
- Multiple configuration files are required in order to allow DWM to properly function. Most of these configuration files are site-specific.
- Running DWM in the research environment requires an additional 'proxySim' process, given the inability of DWM to communicate directly with the emulator. This is typically initialized before the call in the emulator scripts. Therefore, some scripting changing is required.
- Running GUIs in a new manner requires a change to procedure. Although this is relatively straightforward once a user has the required set-up, it violates one of the up-level criteria described in Section 4.1.

While the challenges listed above can be resolved in many cases, it became apparent that allowing an option to run with or without the DWM would be a useful feature. After analyzing the source code, it was determined that software changes were required to enable this. Source code changes were made to the rTMA system to make this possible. The default behavior of rTMA is to run without DWM. For more details on how to run with the DWM process, see Section 5.7.

4.5 Emulator Changes

Based upon changes to message processing, the emulator required a source code change. The processes that would not properly initialize without the emu change were the TGUI, PGUI, and the Multifarious Universal Tester (MUT) playback tool. The GUI issue presented itself as a problem while initializing multiple TGUIs and/or PGUIs. The MUT problem was lack of initializing a playback file. With a surgical source code change to the emulator, both of these issues were resolved.

An additional emulator change that was required was to ensure that the local machine created a unique OARS ID for the en-route interface to work properly. Without this change the rTMA users on the same subnet that use the same HADDS will appear as the same user. The behavior seen is that the second (and all subsequent) users who connect to the same HADDS will be rejected. The software change allows the use of the machine's IP address to create a unique OARS ID, thus eliminating the problem.

4.6 PGUI

The primary changes required to the PGUI were to enable running without the DWM process. An additional change was required to allow for the "-display" X feature to be available. The "-display" command line is important for research users, like NTX, who run multiple PGUIs that are available on different Virtual Networking Computer (VNC) displays. This problem was introduced during TBFM changes to the PGUI required for DWM processing.

4.7 TGUI

Changes were required to the TGUI process to enable it to run without the DWM process. The primary changes required were to allow the TGUI to obtain the necessary fonts from a source other than DWM.

4.8 WDPD

In TBFM 4.0.x, changes were made to the way critical versus non-critical weather is processed. The critical weather processing is considered the RA, while the PGUI and GUI Router (GUIR) are seen as non-critical processes. As part of these changes, the secure copy of encoded weather files was changed. In previous versions a simple script file had been used for this purpose (scp_enc). In this release a source code file (scp_enc.c) was used to copy the weather files. This TBFM secure copy process calls an FAA-provided secure copy tool, which was not available (or required) for the research environment. Therefore, changes were made to WDPD processing to allow the old secure copy method to continue to work.

Another change to weather processing is the location of the processed weather binaries. While the processed binary files remain in the realtime_procs/Output_data/wthr_data/wthr_binary directory, they now are placed under the Center name that is being run. The WDPD process also required a link to realtime_procs inside of the wdpd/src directory. The required links and folders have been created in the NASA Git repository.

4.9 Production versus Development Directory Structure

The TBFM system is primarily geared toward an operational environment (/opt/). While development options exist, some challenges were faced when running in the research environment:

- The relative directory structure of the required executable and preference files does not match the native build structure. In other words, out of the box, the user would generally have to package the executables into a different directory structure than in which they are built. For the busy developer, this is undesirable and can lead to a loss of development productivity.
- Another challenge was the use of the /opt path. This is not used in the research environment. /opt is typically controlled by root and is generally not shared across multiple machines in the research environment in such a way that would lend toward a productive, multi-developer environment.

A handy capability introduced in the TBFM 4.0.x version is the reading of the DataLocation.xml file. In rTMA, this file is located in the common/data/system directory. This file and its associated reader allow the user to specify a relative directory path from the realtime_procs directory. With changes to the DataLocations.xml file, the rTMA system uses the relative location of the required executables rather that the "/opt" structure.

Even with the utilization of the DataLocations.xml file, the expected locations of some system resources are geared toward the deployed operational structure. For instance, the relative locations of directories such as Common_data and realtime_procs that are used in the native build environment do not match the software's expectations. To resolve this issue, and in order to minimize the number of source code changes from the TBFM baseline, symbolic links were used within rTMA. The rTMA Git repository currently has all the required links and configuration files to run in the native research build environment; however, if a new/fresh merge from the TBFM source code is executed it is recommended that the misc/src/configure_system convenience script be used to create the required links. This script creates required Common_data links in the appropriate directories, TS executable link, tsgui link, and Output_data related links. This script also configures the weather directory.

4.10 VNC

The VNC interface has proven to be a productivity-enhancing capability for researchers, especially when geographically separated. Two researchers who are geographically separated but connected to the same secure network are able to view the same screen at the same time. This capability has been used in the NTX production environment where the FAA traffic mangers can quickly 'change the NTX TV channel' to switch between different system displays and/or systems (e.g., DWR, rTMA, SDSS).

A required change to the VNC tool on Linux has emerged as a side-effect of upgrading the operating system. The Real VNC product is no longer natively distributed on Centos and is no longer freely licensed to enterprise; however, the TigerVNC tool is. Given that TigerVNC appears to be a leading open-source VNC utility, it was utilized for running the system during testing. No issues have been identified with the VNC server switch.

4.11 Additional Error Checking

As part of the TBFM 4.0.x refactoring, some additional error checking was added. This is viewed as a positive change and should not be an issue for researchers unless they are adding new functions/methods that require the more stringent error checking. One example of this was the addition of the new data sources (Surface Data Interface and CAP Data Interface) required for PDRC functionality, which requires more stringent declarations for initialization.

4.12 Makefile Changes

The required changes to TBFM makefiles in order to properly build rTMA were minimal. Several tools available in the LMCO development environment (e.g., xmlstarlet, ctdt.exe) that are not required were referenced and needed to be removed. Minor changes were made to the top-level makefile for convenience/productivity. For more information on the build process, see the following section of this document.

5 Building, Configuring, and Running rTMA

This section discusses how to build, configure, and run rTMA.

5.1 Building rTMA

The build_system script, located at the top level of the directory tree (same level as apps), can be used to build the entire rTMA source code. Alternatively, the developer can build each product independently inside of the source directories.

The compiler used to build rTMA is gcc 4.4.7.

To build the entire rTMA system including applications and tools, change to the top-level directory structure and type:

- ./build_system all

Depending on the change it may be necessary to clean out the old executables and object files before building the system. This can be done by typing:

- ./build_system clean
- ./build_system all

If the developer needs to change the version number associated with a given release, perform the following steps:

- mv ./common/api/ctas_version_id.h ./common/api/ctas_version_id.h.org
- cp ./common/api/ctas_version_id.h.org ./common/api/ctas_version_id.h
- chmod 666 ./common/api/ctas_version_id.h
- export the version variable (e.g. TMA_VER=v4.00.07)

NOTE: If compiling the source code on a Centos 5.3 system, it is necessary to comment out the "Wno-address" command in util/api/include.mk. The compiler for the 5.3 OS does not recognize this option, which is used to suppress build warnings.

5.2 Configuring rTMA

This process describes the steps required to configure the rTMA system. Many of the links and directories required for running should already be included in the Git copy of the source code. Therefore, this section only describes those items that are generally required for a developer.

The TBFM adaptation is not stored with the rTMA source code in Git. The operational TBFM adaptations that are replicated to NASA Ames via clearcase multi-site do not require any changes in order to work with the rTMA system. The appropriate adaptation should be obtained and is expected to reside inside the apps/ directory structure, e.g., …/apps/adaptation/ZFW_D10. The adaptation can be an actual folder or a link.

It is common for emulator start-up files to place system start-up output in site-specific directories. Therefore, it may be necessary to create an output directory:

e.g., `mkdir …/realtime_procs/Output_data/adaptation/ZFW_D10`.

To run with weather, the system will need to know where to find the raw weather files. This requires a link to raw weather be updated in the top-level wthr_data directory. A link to the wthr_raw_data is expected in this directory, for example:

wthr_data/wthr_raw_data -> …/Wx/raw

5.3 Working with Shared Memory

The Shared Adaptation Manger (SAM) process is responsible for loading adaptation into shared memory such that it will be available to rTMA applications. There have been no changes to the standard use of the SAM process in 4.0.x from the prior rTMA release.

Problems with shared memory are common. The most frequent issue observed in a multi-developer environment is when one developer loads a specific version of the software from one directory structure and then another developer does not clear shared memory and loads it from a new directory structure. Problems are also seen when switching between adaptation versions.

To properly load shared memory, the adaptation directory or link must be created as specified in Section 5.2. Then the user can change to the sam/src directory and type (for example):

- ./sam –c –A ZFW_D10

Or if adjacent center adaptations are also required in addition to the primary center adaptation, type (for example):

- ./sam –c –A ZFW_D10 –J ZAB_P50 –J ZHU_IAH –J ZKC_T75 –J ZME_MEM

This will load the primary center adaptation and the specified adjacent center adaptation into shared memory.

To show the shared memory that is currently loaded, the user can issue the following command:

- ./sam –s

This will typically result in output similar to that shown in Figure 6. In this example shared memory for both local and adjacent centers are loaded for TRACON groups D10 (DFW), D11 (DAL) and EDC (departure system).

```
id ds tr nam yr/mm/dd hh:mm:ss adaptation dataset   directory pathname
-- -- -- --- -------- -------- -------------------- ------------------
 1  2  0 D10 13/09/13 10:10:09 ZFW_D10              /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZFW_D10/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZAB_P50/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZHU_IAH/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZKC_T75/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZME_MEM/
 2  3  1 D11 13/09/13 10:10:11 ZFW_D11              /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZFW_D11/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZAB_P50/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZHU_IAH/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZKC_T75/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZME_MEM/
 3  4  2 EDC 13/09/13 10:10:13 ZFW_EDC              /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZFW_EDC/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZAB_P50/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZHU_IAH/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZKC_T75/
                                                    /casa/pdrc/iads/snail/PDRC_3.0.4/apps/adaptation/ZME_MEM/
```

Figure 6: Output from the SAM 'show' Command (./sam –s)

To clear shared memory, issue the following SAM delete command:

- ./sam –d

It is good practice to follow up a SAM delete with a SAM show command to verify that shared memory was indeed cleared. Sometimes shared memory may not be cleared if the user is trying to clear it from a different directory structure than was loaded, or if a different user than the one who loaded shared memory.

In some cases when trying to clear shared memory, it may be useful to use the native 'ipcs' tool. This tool reports inter-process communication facilities status. Issuing ipcs on the command line will typically result in a command line output as shown in Figure 7. In this output the shared memory loaded by user pdrc can be seen. To delete a specific segment of shared memory, the 'ipcrm' command can be used on a specific shared memory ID. For example, to delete the first segment in Figure 7, the user could issue the command: ipcrm –m 1114112. NOTE: if the steps described in this section do not clear shared memory, the machine may need to be rebooted.

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 1114112    pdrc       777        9216000    2
0x00000000 1146881    pdrc       777        3936000    2
0x00000000 1212418    pdrc       777        7680000    2
0x01020300 2162691    pdrc       666        18304      1
0x01020301 2195460    pdrc       666        66245968   1
0x01020302 2228229    pdrc       666        66183984   0
0x01020303 2260998    pdrc       666        66525920   0
```

Figure 7: Output from the ipcs Tool

5.4 Working with emu

The M&C emulator (emu) process is used by developers to run the TBFM or rTMA system without the full Monitor and Control (M&C) interface. While initial configuration of these scripts can be tricky, after the user has a few emulator files that work, it is generally straightforward to invoke these scripts and modify adaptation or software as required.

The emulator software can be found in the tools/emu/src directory. The emulator also has a docs directory with a man page describing the available options. The man page describes a number of options that may be helpful for specific scenarios, but many are rarely used.

The typical usage of emu is to invoke an emulator script from a file, like such:

- emu –f <myemufile>

where <myemufile> contains a series of commands that set up and execute TMA.

The primary flow of an emu script is to complete the following steps:

1) Set up an rTMA process with the "-p" option. It is important to remember the name of this process because you will need it later in the script. If the last optional argument is left blank, then it is assumed that the process will be started on the local host, otherwise supply the machine name that the process will be running on.
2) Execute the rTMA process with the "-x" option. It is important to supply the appropriate command line options in this step. Note that in order to see each process' development options, you must use the "-MD –h" option; otherwise, only the operational help menu is available
3) Set up a listener for the processes that require it. This is done with the "-l" option and using the appropriate service name. It is common to set the base port with the "-b" option prior to setting up the listener.
4) Connect the processes to other processes with the "-c" option.

In most cases it is easier to start with an emulator file that is known to work and then modifying it to fit their needs.

5.5 Running rTMA Live

rTMA can be run with live data from En route automation (Host/ERAM), Terminal (STARS/ARTS) or TFM data (ASDI/TFMDG). The rTMA interfaces typically have similar command line options and use fairly consistent processing among them, so once a user becomes familiar with one data source; it becomes easier to include additional data sources.

The live Center data interface (HDIF to HADDS) is the most commonly use data source and is often complemented by Terminal data (ADIF to ARTS/STARs) when more frequent radar updates are required in the Terminal area. Center data can be obtained from ERAM or the legacy Host system. In either case the en route automation data is made available to rTMA through a HADDS system.

The rTMA system can also be run with TFM data. This capability is enabled given TDIF interface. TFM data was added to the operational TBFM system primarily in support of TBFM instances on the US/Canadian border. The FAA TBFM system receives a data feed called TFM Data to Government (TFMDG), which includes more messages than are available in ASDI. However, ASDI data can be used to drive the system with minor changes to adaptation.

5.6 Running rTMA Playback

Running a playback in rTMA is enabled by the MUT utility. This tool reads binary files produced by the system and replays them at the specified rate. MUT mimics the live interfaces in such a way that the downstream processes (e.g., CM) cannot tell the difference between live and playback. An advantage of this playback approach is that no special software needs to be maintained inside of the application logic in order to enable replay. A disadvantage of this playback approach is its inflexible nature. For instance, it cannot skip backward in time, or be readily modified like cm_sim playbacks.

Examples of how to run the rTMA system in playback mode are available on the Ames rTMA Confluence site.

5.7 Running DWM

The DWM process takes special configuration. This is true for both the research and operational environments. Once a user has performed all the required set-up, DWM can be a handy way of initializing different displays. This section discusses how to configure, run, and debug DWM.

5.7.1 Required Configuration Files, Directories, and Links

This following list contains configuration files that DWM utilizes in its processing.

- DwmDefs.xml
  - The DWM process expects to find a DwmDefs.xml file in the dwm/src directory. This file can be copied into the dwm/src directory from the dwm/presets directory without modification.
- Customization directory
  - Each TBFM site requires specific DWM customization files. In rTMA, these customization files are expected in the common/customization directory (as specified in DataLocations.xml). An example of these files has been populated in the Git repository.
  - CenterSockets.xml
    - The user should change the machine name to match the location of where the CM is expected to run that DWM will be connected to.
    - Ensure that the port specified in this file matches the port on which the CM is running. This is typically specified in the emulator start-up script (look at the –l listener setup for CM – e.g. port 2003).
  - WorkstationConfig.xml
    - The machine name listed here must match the location of where DWM will be run.
  - TbfmCustomization.xml
  - StartMenu.xml
    - This file specifies which processes can be initialized by DWM. For instance, if the user wants to bring up the PGUI for DFW, an individual entry must be made in this file.
  - Other configuration files that can be used, but do not appear to be essential, are ChannelConfiguration.xml, Communications.xml, ProcessorConfiguration.xml, and MCView.xml. See the system administrator's manual for more information.
- Configs directory
  - The "configs" directory is used to store user-defined configurations that are saved from the DWM user interface. Saved configuration files can be loaded at a later time.
- Documents directory
  - If the user wishes to load documents (must be PDF) through the DWM interface, store these in the common/documents directory.
- Required links
  - DWM expects a shm_adp_mgr/sam in Common_data directory. This can be accomplished with a link. DWM uses this to load/clear shared memory.
  - DWM expects a link to adaptation at the same level as apps (instead of inside of apps where it resides). Therefore, this link has been created in Git.

    - DWM expects realtime_procs at the same level as apps (instead of inside apps). Therefore, this link has been created in Git.
- Bitmaps
    - DWM expects a pinDown.xpm and pinup.xmp in the Common_data/bitmaps directory. These xmp files can be copied from the apps/dwm/data directory.

5.7.2 DWM's Use of SAM, Requiring Multiple Machines

The DWM process clears and loads shared memory prior to initializing the requested GUI. What this means is that if a separate rTMA process that requires SAM is running (e.g., ISM, CM) then it will fail when loading a GUI with DWM. For this reason, multiple machines are required in order to run DWM. NOTE: The loading of one DWM process does not appear to impact another DWM process, but it will impact the CM and/or ISM processes.

5.7.3 proxySim

The rTMA emulator cannot directory start the DWM process. Instead, the proxySim process is used. This is a new process created for this purpose that can be found in the emu/src directory. To run DWM, issue the following command from the emu/src directory:

- ./proxySim <path_to_dwm_executable> -debug –id 1&

where the path to the DWM executable is …/apps/dwm/src/dwm.

The –debug option can be helpful in debugging any issues that may arise in running DWM.

Debugging output of DWM will be placed in a .dbg file in the dwm-<process_id> directory of the Output_data/<year>/<date> folder.

Additional output that is helpful in debugging DWM issues can be found in the local machine's /tmp directory in the proxySim.<pid>.log file.

6 Debugging rTMA

6.1 Debugging rTMA

6.1.1 More Emulator Output

A good first step of debugging, especially with start-up issues, is to run the emulator in debug mode. While this produces a lot of output to standard error, which admittedly is not all useful, it can help to give greater insight into where the system may be hanging up. To run the emulator in debug mode, just add "-o all,vft" to the last line of the emulator script.

6.1.2 Examine Start-up Output

Another activity that is particularly helpful for start-up debugging is examining the textual output of process start-up output. This output typically goes into the following directory:

apps/realtime_procs/Output_data/adaptation/<site>

For instance, if you are starting arrival rTMA at ZFW, the output for the TGUI would be at apps/realtime_procs/Output_data/adaptation/ZFW_D10/tgui.log.

The process error logs typically only help through initialization. If errors are experienced after startup, the process *.dbg files can be analyzed. The process output is turned on by default and resides in apps/realtime_procs/Output_data/<year>/<day>/<process-id>. The process ID is the UNIX process ID that was running on the system. Typically sorting by time (`ls –altr`) is the best way to identify the latest process. Inside of the process logs are *.dbg, *.sys, and *.msg files. For debugging purposes, the debug (*.dbg) files can have useful output.

6.1.3 Turn on Process Debug and Test Point Output

Another layer of debugging that can be performed is to enable verbose debugging on a process-by-process level. The debug options can be seen by looking at the developer help menu on each process (e.g., ./cm –MD –h). The debug typically uses a –X (or –x) command line option. For instance, CM debug for a specific process is initialized like this:

- -X LIB_ARCS

while debug for all CM processes would be initialized like this:

- -X ALL_CLASS

The output for verbose debug options will go to the *.dbg files mentioned in Section 6.1.2.

Some of the rTMA process support the "-itp" option. This option turns on integrated test points in the software that write to *.tpf output files in the process directory (along with the .msg files).

The output to *.tpf consists of function calls made to the process in question. This output is expected to be run through the automated test point (atp) tool in order to be analyzed. The atp tool requires the process executable and the *.tpf file in order to produce its output. For instance, debugging the cap process, one may issue the following command line options from the process output directory:

- <path>/tools/atp/src/atp *.tpf <path>/apps/cap/src/cap

6.1.4 Add to the Emulator File Progressively

A general rule of thumb is if a large emulator start-up is giving you trouble and initial debugging has not been successful, progressively build the emulator file. This is often done by commenting out processes in the emulator file and re-introducing them one-by-one until the problem is discovered and resolved.

It may be helpful to introduce several emulator files to keep track of the progressive building. For instance:

- hdif.ism.cm.emu (start with just HDIF, ISM and CM)
- hdif.ism.cm.ra.emu (then add in Route Analyzer)
- hdif.ism.cm.ra.dp.emu (add in Dynamic Planner)
- hdif.sim.cm.ra.dp.tgui.emu (add in the Timeline GUI)
- etc. until you have the desired processes

6.1.5 Analyze the System Messages Using Binary Data

If the process debug logs do not help, then another layer of debugging can be done against the *.msg files. These files contain all of the messages passed between rTMA processes and hence can be very useful for debugging and/or analysis. The challenge with using these files is that they

must be translated into textual form, and the user must be familiar with rTMA processing in order to know the standard message processing and meaning of each item within the message. For more information on how to translate the binary into readable ASCII text, see Sections 8.1.2 and 8.1.7.

6.1.6 Email the rTMA User's Group

Lastly, if you have tried these debugging techniques and are still struggling with rTMA, send an email to arc-af-rtma@lists.nasa.gov . In your question, be sure to give specifics of your problem and attach any relevant files (e.g., .emu file, .mut file, relevant output). Also list any debugging steps you have already taken.

7 Testing of rTMA 4.00.07

7.1 Operating System Version and Packages

A considerable amount of testing was performed to ensure that minor variations in operating system packages did not degrade the system. This testing was required due to the desire to move to an open-source operating system (Centos) and to migrate to a later version (Centos 6.4) while also providing assurances that this move did not degrade system performance as compared to the FAA TBFM system.

Operating system testing began by obtaining a complete listing of all FAA packages that are utilized on the operational system. The FAA's second-level support personnel provided this list. On NASA equipment, each of these packages was manually installed until the research operating system was a near-exact match to the TBFM production system. This baseline system was then used to determine whether upgrades to the operating system had any impact on the system performance. The results of this testing were that the later operating systems (Centos 6.2 and Centos 6.4) demonstrated no loss of capability.

7.2 Core TBFM Functionality

Testing of the rTMA capability closely followed the functionality identified in the TBFM 4.0.x operator's manual. The comprehensive nature of the document allowed coverage of a large percentage of the overall capability.

No formal test logs were taken during this testing, but several defects were found and resolved either through software or configuration changes to the system. The process could be described as largely mechanical in nature, in which the tester would go through a section of the operator's manual at a time while ensuring that the rTMA system capability matched the description provided. A special focus was given to Section 8 of the operator's manual, which covers scheduling functions. The following sections describe the testing performed and the results obtained.

7.2.1 TGUI display

This section describes the testing performed on the TGUI. This corresponds to the TBFM functions listed in Chapter 3 and Chapter 6 of the operator's manual.

Examples of the capability tested on the rTMA TGUI are as follows:

- General timeline display
- Color coding of delays
- Broadcast function
- Manual freeze
- Manual assign
- Reschedule single
- Suspend/Resume
- Priority flight handling
- Meter point change
- Scheduling departure features
- Blocked interval
- Create slot
- Quick key functions (F panels, font changes, etc.)
- ETA/STA aircraft tags
- Freeze horizon display

There were several anomalies discovered during TGUI testing:

- Ctrl-U Quick key (Special Use Airspace) does not appear to do anything. This is also shown as an option in the F1 Control Panel of the user's manual but did not show up on the system F1 panel. NOTE: This could be due to adaptation configuration, and was not seen as a reportable error (e.g., JIRA ticket).
- The fonts used for size symbols on the rTMA TGUI do not match those seen in the user's manual. This is believed to be a low priority reportable error.
- Saving changes on the F9 panel results in the TGUI crashing. This is a high-priority error which will be resolved. Note: In most cases the TGUI will immediately come back up given failure recovery in the system. The work-around for this issue to save the desired changes and then load them from the TGUI in a subsequent run.

7.2.2 PGUI Display

This section describes the testing performed on the PGUI. This corresponds to the TBFM functions listed in Chapter 4, Chapter 10, Chapter 14, and Chapter 15 of the operator's manual.

Examples of the capability tested on the rTMA PGUI are as follows:

- PGUI F panels
- Display of metering constructs
- Display of NAS data
- Display of flight data blocks (different coloring options, etc)
- Track history
- Trend vectors
- Save/Load custom files
- Display wind vectors
- Sequence list
- PGUI timelines
- All quick keys

One anomaly was discovered during PGUI testing:

- On the F8 TRACON connection status, the system does not show actual connection time but instead shows 0000:00 01/01/1970 as date/time. This is believed to be an artifact of starting the system with the emulator, and therefore is considered a low-priority issue.

7.2.3 Status and Scheduling Parameter Input

This section describes the testing performed on the various inputs of TBFM that allow for scheduling input, as well as testing of the status panels. This corresponds to the TBFM functions listed in Chapter 7 of the operator's manual.

Examples of the capability tested in this section are as follows:

- Configuration change
- Future configuration change
- Airport acceptance rate
- Future airport acceptance rate change
- TRACON acceptance rate
- Runway matrix separation
- TRACON buffer
- Gate acceptance rate
- Stream class modification
- AMDT adjustments
- Blocked runway
- System settings recordings

Anomalies were discovered during this testing:

- Single-gate free-flow does not properly function. That is, the SGFF graphic does not show up on the timeline, and the gate times do not appear to be affected by it. This is believed to be a low-medium priority issue.
- The 'Coupling' options described in 7.4.5 of the operator's manual do not show up. This is likely adaptation configuration-related. Given that a future TBFM release contains work geared toward coupled scheduling and extended metering, this is believed to be a low-priority issue.

7.2.4 Scheduling Aircraft and Traffic Management

This section describes the testing performed on the scheduling and air traffic management functions of TBFM. This corresponds to the TBFM functions listed in Chapter 8 of the operator's manual.

- Freeze Horizons functionality
- Basic scheduling, re-scheduling of flights (e.g. reschedule all, single flight)
- Broadcast capability
- Aircraft right click options
- Freezing an aircraft
- Changing an aircraft value (e.g., meter fix)—and all after it

No anomalies were discovered during this testing.

7.3 Side-by-side with Operational TBFM

Verification and Validation (V&V) of the rTMA system involved side-by-side analysis of rTMA ETAs and STAs as compared with the operational system. The primary focus was on the ETAs given different constraints that may have been present within the system affecting the STAs, as well as timing differences.

The object of this evaluation was to ensure that the ETAs and STAs were within the same general range, as well as to ensure that all flights were accounted for on both systems with similar meter fix assignments.

An anomaly discovered in this testing was that the rTMA software consistently predicted flight times to be longer than the operational system. This is believed to be due to changes incorporated in the rTMA base back merge. To verify this assumption testing was performed on the rTMA system which did not include the previous rTMA base changes, and the ETAs matched the operational system. Additional testing is required to further quantify and analyze the differences in ETAs between systems.

7.4 Adjacent Center Data Processing

As part of rTMA base testing, all adjacent center data feeds were tested. In previous versions this has functioned as a stress test for the rTMA system given the increased loading it puts on processes due to more flights in the system. The rTMA base system performed well with all adjacent center feeds, only utilizing a small percentage of overall system CPU.

8 Tools and Convenience Scripts

This section describes tools and scripts used with the rTMA system.

8.1 TBFM Tools

The TBFM system is distributed with a number of tools that exist in the tools/ directory at the same level as /apps. This section gives a brief explanation of each tool. In many cases, man page documentation that is no longer distributed with the TBFM was summarized in the following sections.

A number of the tools mentioned in this section assume that shared memory is already loaded. It is generally easiest to use the same system that is running rTMA to process the output data, given the assurance that the shared memory exactly matches that of the system when the data was output.

NOTE: While most of the tools mentioned in this section appear to be maintained by the FAA, it is unclear to what level that is true. Some of these tools—like MPA, APM, and MUT—are required for everyday functions and therefore are fairly well maintained. However, lesser-used tools appear to have command line options that may not be fully supported.

8.1.1 AMP—Algorithmic Message Processor

AMP is a program similar to MPA that reads TBFM binary message files and outputs messages that match user-specified criteria. Unlike MPA, AMP implements a general purpose comparison language for user queries. AMP's capabilities were designed to not overlap those of MPA, as

such, AMP has several limitations that require the use of MPA to filter and pre-process certain types of messages.

AMP assumes that each message contains information about a single aircraft, so for messages that contain information about multiple aircraft, the message file must be pre-processed using "mpa +r -O" to split these multi-aircraft messages into separate single aircraft messages.

AMP builds an in-memory database of information gleaned from certain messages that it encounters as it reads and parses messages. AMP will keep all of the information found in messages that pertain to flight information, namely, Flight_plan_st and CTAS_flight_info_st data, which is obtained from Host flight information messages.

For more information, see the AMP help menu.

8.1.2 APM—Analyze and Parse Messages

APM is a tool used to read and write binary data files (BDF), to read and write adaptation-independent ASCII files (APM), and to generate reports.

Command line options are used to select the format of the input as well as the desired output.

The most common use of APM in the research environment is to convert a binary file to text in its entirety. This can be done in one of two ways.

- ./apm –b –c <binary_file.msg>

The command above converts the file into a comma-separate stream of values that (mostly) match the APM data dictionary ordering. Whereas the command:

- ./apm –b <binary_file.msg>

produces a very large and verbose output file that has each message field name listed in a column on the left, with the value of that field listed on the right.

A handy capability of APM is to output a data dictionary. The data dictionary can be used to look for specific messages or message fields that the user may desire. To print a data dictionary of the target system, issue the following APM command:

- ./apm –d all

For more information on APM, see the help menu.

8.1.3 Delay_profile—Information on Flight Delay

Delay_profile is a tool that extracts delay-related data from CM message logs that have been preprocessed via MPA/APM and provides an overview of the delays given to a specific aircraft. Amendments to the assigned scheduling fix and runway are taken into account in the reported delay.

The delay profile is printed to stdout, so generally the user redirects it to a file.

A typical use of delay profile begins with the use of the MPA and APM tool in order to extract the subset of messages that are relevant. For example:

- mpa -a AAL848 -m ADD_FLIGHT_PLAN -m AMEND_FLIGHT_PLAN -m AIRCRAFT_ETA_AVERAGED -m SCHEDULE -d tgui2 -r –O *N01.msg | apm -b -C off > mpa_apm_AAL848

The command above will extract all the binary messages for flight AAL848 of the type specified (ADD, AMEND, ETA_AVERAGED, and SCHEDULE) that are bound to tgui2 and convert these to text.

Next, the command:

- delay_profile mpa_apm_AAL848 > AAL848_delays

will output a file similar to below.

Delay Profile for Flight: AAL848/T0828 Date: 2009/06/13

Message		Meter Fix			Meter Fix	Runway			Runway
Type	Time	NAME	ETA	STA	DELAY	NAME	ETA	STA	Delay
ADDFP	00:01:01	DIRTY				27L			
AMDFP	00:01:01	DIRTY				27L			
ETAVE	00:01:02	DIRTY	01:53:27			27L	02:05:12		
SCHED	00:01:06	DIRTY	01:53:27	01:53:27	00:00	27L	02:05:12	02:05:12	00:00
AMDFP	00:01:06	DIRTY				26R			
SCHED	00:01:06	DIRTY	01:53:27	01:53:27	00:00	26R	02:05:12	02:03:05	-2:07
ETAVE	00:01:06	DIRTY	01:53:27	01:53:27	00:00	26R	02:03:05	02:03:05	00:00
SCHED	00:01:07	DIRTY	01:53:27	01:53:27	00:00	26R	02:03:05	02:03:05	00:00

8.1.4 DIF—Difference of Two Binary Message Files

Dif is a tool that compares two binary message files. It is similar to the UNIX diff, but it does not assume that messages will be in exactly the same order in the two files. Instead, dif maintains a time window, and if a message appears in the time window in both files, then that message is treated as being the same message.

The default behavior of dif is to compare the message headers of each message in order to conclude that a pair of messages matches. By default, only the message-type, source-process-type, and destination-process-type are compared. A number of command line options exists, which can be seen in the process help menu.

Example: Straightforward use with no command line options

- dif file1.msg file2.msg (simple usage)

Example: Begin difference after finding an IC_TIME_SYNC message

- dif -s IC_TIME_SYNC file1.msg file2.msg

8.1.5 FRD- FRD Location Identified to Position

FRD is a tool used in conjunction with MPA, APM, and Gnuplot to produce graphical output for analysis of flight plan routes, host converted routes, and arc adaptation.

Generally, binary message data of interest is extracted using a combination of MPA and APM filtering to retrieve FH/route and HX/ak_string data items and pipe that into FRD.

A feature of FRD is the ability to dump out selected information from shared memory adaptation. Currently FRD support indexes for gates, meter fixes, and waypoints as well as waypoint filtering and location information.

8.1.6 MAP—TBFM Metering Analysis Program

MAP is a program that displays TBFM metering messages destined to the En Route Automation System. It obtains these messages by reading the ISM binary message log. The user has control over which airport(s), meter reference point(s), FPA(s), and/or sector(s) to display.

MAP outputs a full screen update after each message received. If MAP is run using a recorded message file and the output is redirected into a file, the less program can be used to step through the updates to the meter lists one message at a time.

8.1.7 MPA—Message Post-Processor and Analyzer

MPA is a data analysis tool developed to manipulate TBFM binary message files. MPA is able to process a file, selecting a single message, groups of messages, or classes of messages based on user-specified filtering criteria.

The selected messages may then be printed in a user-specified format. Several categories of output formats are available:

1. Short format: one-line-per-message consisting of only TBFM header information
2. Medium format: one-line-per-aircraft consisting of the short format header information output once for each aircraft ID contained in the message
3. 2-D process-type format: one line per message where all destination processes for the same message are reported on the same line. This makes analysis of CM message logs much easier
4. Dump format: hexadecimal and ASCII dump of entire message
5. Summary format: produce summary reports based on message source, message destination, or message type, the latter being sorted by either message frequency or total bytes.

MPA has a large number of options that can be viewed from the help menu.

The simplest example will show how to initially look at the contents of a message file. The following command

- mpa file.msg | more

will show the messages available in the selected file like below.

```
Iseqno hh:mm:ss srceI destI asize message type

------ -------- ----- ----- ----- ------------------
     1 15:32:10   cm1   mc1   276 APP_INFO
     2 15:32:31   mc1   cm1    28 APP_INIT_LISTEN
     3 15:32:31   cm1   mc1    28 APP_LISTENING
     4 15:32:37   dp1   cm1   204 SEND_HOST_NAME
     5 15:32:37   cm1   dp1    20 VERSION_ID_OK
     6 15:32:37   cm1   dp1   152 INIT_CONFIG
```

In the next example unwanted messages are filtered out, for instance, those that come from DP. Notice how message number 4 in the example above is not in the output, as can be seen by the gap in sequence numbers from 3 to 5.

- mpa +s dp file.msg

Will result in:

Iseqno hh:mm:ss srceI destI asize message type

```
------ -------- ----- ----- ----- ------------------
     1 15:32:10   cm1   mc1   276 APP_INFO
     2 15:32:31   mc1   cm1    28 APP_INIT_LISTEN
     3 15:32:31   cm1   mc1    28 APP_LISTENING
     5 15:32:37   cm1   dp1    20 VERSION_ID_OK
     6 15:32:37   cm1   dp1   152 INIT_CONFIG
```

8.1.8 MUT – Multifarious Universal Tester

MUT is the universal message test tool developed to replay recorded TBFM binary message data. MUT connects to TBFM real-time processes under control from M&C, or emu, performs handshakes, and injects recorded binary message data to appropriate processes as determined by timing information obtained during data recording.

MUT begins playing back data from the beginning of the first input file in a directory. It should be noted that during the time that the data was recording, things happened before the start of this file that may not be captured, such as flight plans. Therefore, it is important that a playback begin early enough so that all required data is available. The best way to do this is to specify an input file that was recorded some time before the time of interest and then fast forward to somewhat before the desired time.

A number of examples exist in the tools/mut/dat folder that can assist in configuring MUT.

Also, see the MUT help menu for more details.

8.1.9 PAM—Parse Aircraft Metadata

The PAM tool is a program that reads TBFM binary message files produced by ISM and upstream sources (e.g., HDIF) and produces a by-aircraft synopsis of significant events. PAM also offers the capability to create a two-dimensional timeline to separate messages from different sources and with differing computer identifiers into separate columns. This greatly increases the ability to understand the interactions between data sources for a particular aircraft.

The PAM tool has a substantial number of command line options that can be used to tailor the output. See the help menu for more information.

8.1.10 TL_PROFILE—TBFM Timeline Profile Extractor

Tl_profile is a tool that generates text versions of the timeline (timeline profiles) for a specified fix or runway based on data from CM message logs. Via command line options, the user indicates the fix or runway associated with the timeline of interest, an optional time interval, an optional airport, and the name of a file that was generated by an mpa/APM run on CM logs. See the tl_profile command line options for details.

8.1.11 XTC—XML Translation and Conversion

XTC is a program to perform format conversions of XML data used within the TMA system. XTC is used to convert ASCII XML data into TMA binary messages of the type XML_MESSAGE. Conversely, XTC is also used to convert the XML_MESSAGE message type back into ASCII XML. The latter conversion will encapsulate the XML data found in the

message within an <xtc>...</xtc> envelope containing attribute data obtained from the TMA message header so that information is not lost. The <xtc> attribute data may then be used in the conversion back to binary messages. If the ASCII XML file does not contain <xtc> encapsulation, then TMA header information may be specified on the XTC command line.

See XTC command line options for more information.

8.2 Convenience Tools/Scripts

The scripts described in this section are located in the apps/scripts directory.

Configure_system

The configure_system script is a convenience routine that creates many of the links required in a new/fresh TBFM install, as well as configures the weather directories. To run the script, set the CTAS_ROOT environment variable to point to the apps directory, and then execute the configure_system script with no command line arguments.

Pref_install

In order for the TBFM system to function, a preference directory must exist for the Centers/ TRACON groups you desire to run. The pref_install script is a convenience script to assist in that purpose. NOTE: You can also copy a previously used pref directory. The scripts directory also has other preference file routines like pref_restore, pref_update, etc.

Scp_enc

This script is called by the WDPD process. Its job is to copy a weather file from a location near the WDPD source structure to a more common location in Output_data, with a unique name so that the file will not be over-written and can be made available to other processes.

8.3 'Mini-TMA' Startup Analysis

During the rTMA 4.0.x up-level, the team re-evaluated the manner in which the system was initialized. The reason for this analysis was to determine whether NASA was starting the system in a similar way to FAA personnel. If not, then this may lead to unknowingly running the system in a state that differs from the operational system and may also cause future maintenance issues if FAA personnel later stopped supporting the startup mechanism NASA uses.

For this analysis, the 'mini-TMA' startup script was examined. 'mini-TMA' is the name FAA personnel have given to the minimal system utilized on TBFM equipment by adaptation development personnel at the site. 'mini-TMA' has been supported for a number of years, and there is no indication that this support is wavering in TBFM. In fact, some updates were required to this start-up process due to the addition of the DWM capability. Therefore, this seemed to be a good candidate for analysis of start-up.

The mini-TMA start-up script is approximately 1500 lines of Korn shell script, which utilized the emulator process to start up TBFM. Mini-TMA is geared toward assisting the user in adaptation evaluation on a TBFM analysis workstation. This script was written and is maintained by FAA second-level support personnel.

The PDRC team obtained the mini-TMA scripts and evaluated it for differences in the start-up process between the standard start-up scripts that are used daily at NTX. The results of this analysis revealed that there was more in common than different between these two start-up

mechanisms. Mini-TMA startup leverages the /opt directory structure, including the logical arrangements of the processes as they are deployed to the field. This makes sense because the operational software is installed as a package on the support workstation in the same way it is on other operational equipment. NTX start-up procedures run the system out of the same structure the source is built within. This makes sense because researchers do not want to worry about the time it takes to package software they just modified and built and would rather use that time for more development. Another difference between the mini-TMA start-up and NTX start-up is that mini-TMA is geared toward testing adaptation, therefore there were options to test adaptations from an alternate directory for comparison against an existing adaptation. This includes some basic error checking. This is not required in the typical NTX start-up, which only changes adaptations when required. Mini-TMA has a handy feature that allows the user to select a playback of a binary data file. It may be helpful to have a similar wrapper script in the NASA environment, especially now that byte order is no longer an issue when replaying operational data (given the move to Intel).

A helpful lesson learned by studying the mini-TMA script was the options with which the emulator is initialized. Inside of the .emu files created by mini-TMA, the last line specifies “"-o all,vft”. This tells the emulator to run all processes in such a manner that they provide output and traceability information. Adding these lines to the NTX start-up scripts has proven to be helpful in debugging certain start-up-related challenges.

In summary, given the similarity between current-day operational use of the mini-TMA script and current NASA start-up procedures, there are no recommended changes to the NASA start-up procedures. Furthermore, if the FAA’s adapters continue to use the mini-TMA script, which leverages the emulator to start up TBFM, then completely moving away from this start-up option seems less likely in the longer term future.

9 Assess Feasibility of a 64-bit Version of rTMA

This section discusses the work performed on the rTMA up-level to perform an engineering assessment of the feasibility of upgrading rTMA to a 64-bit operating system. This section primarily focuses on the recommended process for a 64-bit upgrade rather than specifics of the upgrade itself. To support this work, a copy of the rTMA source was modified to allow building in 64-bit mode. Error logs from the build process were analyzed to discuss initial findings and recommended next steps.

9.1 Current TBFM Operating System

Currently FAA TBFM is running on a 64-bit Centos Operating system. The make utility and compiler are both 64-bit; however, compiler options are used to build the source code in 32-bit. Specifically, the “-m32” option (machine-specific build option) is used in the top-level include.mk file. If this 32-bit-specific build option is removed, the compiler will attempt to build the system as a 64-bit executable.

9.2 64-bit Upgrade Process

A literature search was performed to determine whether there were industry-standard processes for the common challenge associated with upgrading Linux applications to 64-bit. Intel has published an industry standard process for this task[21], which was utilized as the basis for the

process recommended below. Some of the examples and text from this process are included in this report as well. The process outlined in the document calls for making the application '64-bit code-clean', regression testing it in 32-bit, and then optimizing for the target platform.

9.3 Making the Source 'Code-Clean'

The first step is making required changes to ensure that your source code is 64-bit 'code-clean'. One of the larger implications of this requirement is that the code must be modified to support the data model used by the 64-bit operating system. In the 32-bit data model, the fundamental data types int, long, and pointer are each 32 bits in length. The 64-bit architecture differs, int is 32 bits, and long and pointer types are 64 bits. Because of these differences, it is likely to encounter issues during porting.

An analysis of rTMA was performed to determine the estimated percentage of the source that could be subject to data type conversion challenges. This analysis was performed by creating a sandbox rTMA system that removed the –m32 build options, allowing the system to build in 64-bit. In addition, the makefiles included an option to disregard errors in such a way that the build process continued after experience failures. The gcc compiler provided a specific error notice when a data conversion issue was the main cause. Data type errors from the build process were collected and separated by process. The results of this analysis are listed in the Table 1.

Table 1: Potential 32-to-64 bit Data Type Conversion Issues by Process

Process folder	Number of Potential 32-to-64 bit Data Type Conversion Challenges
common/impl	211
apps/cm/src	51
apps/sam/src	38
apps/pgui/src	37
tools/sip/afg/src	28
tools/apm/src	27
tools/sip/program/src	20
tools/emu/src	18
tools/atp/src	14
apps/edif/src	13
util/socket/impl	11
apps/cap/src	11
apps/ism/src	6
util/motif/impl	4
apps/wdpd/src	4
tools/atc/simulation_manager.dir/sm_run.dir	4
util/endian/impl	3
common/ra/impl	2
apps/is/src	2
apps/tgui/src	2
tools/mut/src	2
util/meschach/impl	1

Process folder	Number of Potential 32-to-64 bit Data Type Conversion Challenges
apps/ra/src	1
tools/atc/libraries.dir/atc_library.dir	1
tools/sip/data_monitor/src	1
tools/sip/playback/src	1

The results discussed in Table 1 analysis indicate that the large majority of data conversion-related challenges lies in the common directory structure, followed by the CM process, the SAM utility, then the PGUI.

Note that some data type issues may not have been found by the compiler, and in some cases a single fix may actually result in multiple warnings/errors being removed. Therefore, this list may not provide an exhaustive list of data type conversion challenges. However, this process is believed to be useful in determining where the majority of the data type-related analysis needs to be focused.

Based upon this analysis, a considerable number of data type corrections are required to be fully 64-bit compliant. Once the data type conversion is performed to allow the build process to proceed, additional analysis may be required to pack the messages in an efficient format.

9.3.1 Use ANSI const Instead of #define in Hexidecimal Constants

Another part of the recommended upgrade process from Intel documentation is to change a number of the "#defines" to "const" for hexadecimal assignments. Using const, the compiler can perform the necessary type checking, and the user will get a warning if misuse is attempted. To illustrate this, the following example (from Intel documentation) should be considered:

```
1 #define OFFSET1 0xFFFFFFFF
2 #define OFFSET2 0x100000000
```

In the 32-bit operating system, OFFSET1 is –1 and OFFSET2 is 0. In the 64-bit operating system, OFFSET1 is 4,294,967,295 and OFFSET2 is 4,294,967,296. To avoid the error, instead use ANSI's type constant and properly qualify it with signed or unsigned.

In rTMA, a substantial number of #define Hex constants are used. A count of these were performed that revealed approximately 840 uses in all the apps and tools source code. Not all of the rTMA #define Hex constants may be 64-bit up-level issues, but they could be problematic for the reasons mentioned above. The work associated with resolving the #define Hex issues is estimated to be lower than the data type challenges in the previous section.

9.3.2 Additional Guidelines for 'Code-Clean' Work

The Intel documentation gives additional guidelines for 'code-clean' activities required to ensure a smooth upgrade to 64-bit. The sections of the Intel document that seem most relevant to the TBFM upgrade are as follows:

- Intelligent use of integers and Pointers
- Watch for Packing, Padding, and Alignment issues
- Watch for Truncation

- Use Portable I/O format
- Listen to the Compiler.

Based upon compiler errors seen during the 64-bit rTMA build attempt and/or code inspection, these guidelines should be followed methodically. In some cases, following these guidelines may not resolve a 64-bit conversion issue, but it still may make sense to follow these industry standards from a source code performance or maintainability standpoint.

The point about 'listening to the Compiler' is especially important. There are currently build warnings in the TBFM source code. Some of these warnings are intentionally suppressed/ ignored in the operational build process. The rTMA up-level did not attempt to resolve all of these issues. While work is being performed to upgrade to 64-bit, it may be a good time to resolve or greatly reduce the compilation warnings as well.

9.4 Regression Test 32-bit Version

After the 64-bit 'clean-code' work is done, the Intel process recommended regression testing the same source code on 32-bit. Even if a 32-bit build is not a continuing requirement for the source code, building on 32-bit is important in order to be able to test the system in an apples-to-apples manner.

9.5 Cost/Benefit to Upgrade to 64-bit

This section discusses the cost versus benefit of upgrading TBFM to a 64-bit operating system. While the SOW did not ask for this commentary, this question arose during discussions on this task and therefore is included in the response.

The most common reason for upgrading an application from 32-bit to 64-bit is to allow for more addressable memory space for the application. In the case of rTMA, the memory used by each process is low. As a whole, rTMA uses less than 10% of available memory on a single machine with 8GB memory (conservative estimates, in NTX environment). Therefore, memory usage does not appear to be a driving factor.

A more compelling reason for a 64-bit rTMA upgrade may be obsolescence and concern for continued support of 32-bit library files that the system may be using. As the majority of Linux applications continue to move to 64-bit, the 32-bit libraries may not get the same level of support, requiring a migration.

The cost associated with this upgrade is not believed to be large. It is believed this work could be done in a methodical manner with approximately 6-8 weeks of effort including test and documentation time.

Appendix A – Referenced Documents

FAA Documents

The following FAA documents may be obtained from Shawn Engelland or Michelle Eshow.

Ref #	Document/File Name	Brief Description of Document
1	TBFM004 SIG - Final Delivery- 11 11 2010 - Flexible Schedul.pdf	Flexible Scheduling alternatives and selected approach
2	FAA Tech Ops Briefing Revised Final_pdf.pdf	TBFM Tech Ops Rel4.0 briefing. Hardware focused.
3	Hardware and Software CDRLE04-SSDD - Final 20111021.pdf	High level TBFM System/Subsystem Design Document
4	Appendix_T-MnC-SRS-20120620_Book.pdf	Detailed software information on Monitor and Control
5	Appendix_M-TBFM_srdd_wdpd-FINAL40.pdf	Detailed software information on Weather Data Processing Daemon
6	FS DDR slides Final_post_meeting_20110316.pdf	Flexible Scheduling Detailed Design Review
7	TBFM Re-Architecture DDR - Updated Print Version.pdf	Re-architecture Detailed Design Review
8	Appendix_J-TBFM_srdd_tgui-FINAL40.pdf	Detailed software information on Timeline GUI
9	Appendix_I-PRXY.pdf	Detailed software information on Proxy services
10	Appendix_H-TBFM_srdd_pgui-FINAL40.pdf	Detailed software information on Planview GUI
11	Appendix_K-TBFM_srdd_tdif-FINAL40.pdf	Detailed software information on TFMS Data Interface
12	Appendix_L-TBFM_srdd_TS_baseline.pdf	Detailed software information on Trajectory Synthesizer
13	Appendix_E-DWM.pdf	Detailed software information on Dynamic Window Manager
14	Appendix_D-TBFM_srdd_dp-FINAL40.pdf	Detailed software information on Dynamic Planner
15	Appendix_C-TBFM_srdd_cm-FINAL.pdf	Detailed software information on Communications Manager
16	Appendix_F-GUIR.pdf	Detailed software information on GUI Router
17	Appendix_G-TBFM_srdd_ism-FINAL40.pdf	Detailed software information on Input Source Manager
18	12-0421 TI.6480.8 TBFM_System Administrator_Manual Rel_4.0_Final_FOUO.pdf	System Administrators Manual

Ref #	Document/File Name	Brief Description of Document
19	4.0 Functionality Brown Bag_v6.pdf	General description of 4.0 capability
20	Adaptation_ICD_4.00.0_final.pdf	Adaptation ICD. Helpful for adaptation changes.

Industry Documents

Ref #	Document/File Name	Brief Description of Document
21	http://software.intel.com/en-us/articles/porting-linux-applications-to-64-bit-intel-architecture	Intel's recommended process for upgrading Linux applications from 32- to 64-bit