Windows® XP Embedded Goes to Space, Twice! The Importance of Architecting Windows for Devices.

By Sean D. Liming
Edited by John R. Malin
Annabooks – www.annabooks.com

July 2025

Yes, you read the title correctly. Windows XP Embedded (XPe) went into space. In 20+ years, I have consulted with many different clients on many different projects, programs, and industries. The unique people and different company cultures behind all these projects have made for a unique career experience. There have been some standout projects, but there is one project that was literally out of this world.

My typical technical articles and books provide step-by-step instructions on how to use a Windows feature, technology, or software solution. The goal is to help the developer understand the topic and get up the learning curve quickly. What I have never covered is the pre-work that goes into selecting the features, technologies, and software solutions. The project requirements and pre-project discussions that drive the system architecture are the key to success. As a change to my typical technical discussions, this article will focus on the preliminary work and the development process that follows. What better way to cover the topic than a real project that was put in the most extreme environment possible.

Training Class to Consulting Project
A NASA contractor attended one of my Windows Embedded Standard 7 (WES7) training courses in 2013. It wasn't clear what he was working on at the time, but since this was a Windows Embedded course, I assumed it was some ground-based test system.

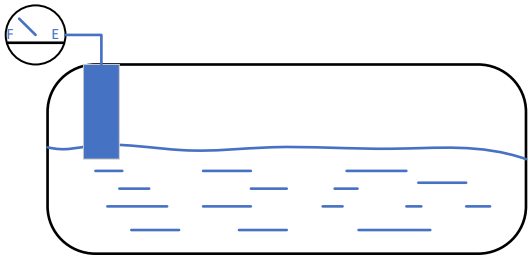
A few years later, the same NASA contractor gave me a call asking about the minimal hardware specifications for WES7. He shared the hardware platform that they were looking into. I knew the processor and boot drive size were not a good fit for WES7, and I made other processor recommendations that would be a good fit. The reply was that the system has tight temperature and power requirements to meet flight operations, and he wasn't sure my hardware suggestions would work. After a few more email exchanges, it became apparent that this project was not ground-based test equipment or something that was going into an airplane. Whatever the project was, it was going into space. I made a few more hardware suggestions and recommended testing Windows XP and other RTOSes. In the end, the hardware specification requirements limited the choices, which dictated the operating system selection. Windows XP Embedded (or the Windows Embedded Standard 2009 (WES2009) version of Windows XP Embedded for those keeping score) was the chosen operating system. Since WES7 and WES2009 are built with different tools, I offered my consulting services to help create the custom Windows operating system builds.

Radio Frequency Mass Gauge (RFMG)
The first step was to learn what the project was all about. NASA always has forward-thinking projects with regard to space flight. Creating spacecraft and satellites is expensive. Reusability and extending missions to save on cost was becoming a new design standard. The current endeavors of going back to the moon and eventually to Mars have brought different ideas about transport to and from distant locations and Earth. One idea was to have a spacecraft cycle back and forth between Earth and Mars to keep supplies going. The "Hermes" in the movie *The Martian* was an example of this type of spacecraft. Fuel becomes a major factor to consider, and the ability to measure fuel remaining and refuel such spacecraft was the key to success.

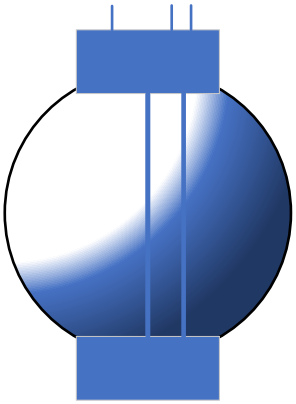
The project study that this NASA team was working on was part of the Robotic Refueling Mission (RRM). More specifically, the measurement gauge of how much cryogenic fuel is in the tank. Of

course, one could ask why such a study be needed, since pressure and quantity gauges already exist? Everyone knows about the Apollo 13 oxygen tank explosion. Not every propellant can be measured the same way, and low gravity (low-g) presents some complications.

So as not to get too far into the science and divert from the main topic, it's best to simplify the problem and the solution under investigation. Think of the fuel tank in a car. The fuel does slosh around a bit, but Earth's gravity at 9.8 m/s^2 is keeping the fuel in the bottom of the tank. A simple float can measure the tank capacity in a car, because the float and the fuel are both subjected to Earth's gravity.



In space with low-g or microgravity, the cryogenic fuel can float everywhere[1]. A simple float measurement is not going to work.



A float measurement could be made when the spacecraft is applying thrust, accelerating, and the fuel settles to one end of the tank. The float would be subjected to the same acceleration as the rest of the spacecraft, and the entire system would act as if it were subject to a gravitational constant the same as the spacecraft's acceleration. Burning fuel to measure the fuel is not ideal or efficient. A solution is needed to measure the fuel in the tank at any time and any state of acceleration during a mission. The solution under investigation was called Radio Frequency Mass Gauge (RFMG)[2,3]. RFMG combines chemistry, physics, material science, and electrical engineering to get an accurate, instantaneous measurement. The science is a little complex, so links are provided in the references below. Very simply, knowing the materials inside the tank and the fuel type, a simple RF chirp is picked up by two antennas inside the tank. The RF response is turned into data that software uses to determine the capacity.

The first tests were performed in the lab on the ground. The second tests were performed in the reduced-gravity simulator aircraft, or "vomit comet". The testing showed that RFMG was a viable solution, so the next step was to test RFMG in space. NASA had already launched and tested two Robotic Refueling Missions (RRM)[4] to demonstrate different technologies. The RFMG test was slated to be part of the RRM3 mission[5,6].

Project Inquiry Drives the System Architecture

With all that background covered, we can now get to the heart of the article. There was some confidence that creating the custom Windows operating system could be done, but for a device that is going into an extreme environment, there were some doubts. The Windows operating system had to be carefully architected to meet the requirements of the operating environment. The pre-work for every project starts with a few questions:

1. What does the system do?
2. What is the hardware specification?
3. What operating system features need to be in the system?
4. How will the user interact with the system?
5. How is the system supported in the field?
6. What is the manufacturing process?
7. Are there any runtimes, database engines, third-party tools, etc., to be installed?
8. What lockdown and security features need to be implemented?

NASA shared the basic system concept: the Windows system will power on, the Windows operating system will boot, the system will auto-logon to the main account, and launch Windows Explorer Shell. A run registry key will start the main application. The operational sequence sounds simple, but addressing the little items is important to avert problems. For example, a sudden power loss and disk corruption can kill the mission. Engaging in discussions that consider failure modes, system constraints, and timing helped drive the system requirements.

The first set of questions focused on what other systems would be connected to the RFMG system and if there were failure modes that could cause any harm or damage to the connected systems. This system was only to be connected to the RRM3's internal flight computer via RS-422, and the Windows system would not be connected to the internal International Space Station (ISS) network. The system would only be used for RFMG testing. From time to time, I question clients about why they want to use Windows Embedded/IoT just to see if a different operating system or RTOS could be used and might be a better fit. With the safety concerns addressed, the big question still remained: why use Windows? As it turns out, the RFMG probe only came with a Windows device driver. All the testing and development to this point was completed using Windows. The timetable to create a solution for the RRM3 launch was short, so there was no time to develop a device driver and all the application software for a different operating system or RTOS. The selected hardware for the mission was limited to 1GB of drive space; so, in the end, Windows XP Embedded (WES2009) was the only option.

The next set of questions focused on user operation, such as human interaction and field support. These are typical questions asked for any project. The RRM3 was to be attached outside of the ISS where there would be no direct human interaction. Everything would be handled remotely. There would not be a spacewalk, if something were to go wrong. Windows would have to survive a hard shutdown, as a safe shutdown would not be possible. Also, there would not be a display. Based on this information, the first set of XPe/WES2009 features that went into the custom operating system were being fleshed out:

- The XPe write filter technologies have to be enabled and configured to protect against any sudden power loss. The Windows OS getting corrupted and not able to reboot would brick the instrument, as there would be no way to repair an isolated system. Putting in the write filter technologies would help mitigate corruption in case of a power loss. A write-through folder would be needed for the data.
- Disk caching would have to be disabled so all data would be flushed to the disk and not held in RAM, again, addressing any sudden power loss.
- Message Box Intercept (aka Message Box Default Reply) would have to be enabled to address any pop-up dialogs. Since this would be a headless system, any pop-up dialogs

could take up system resources. MBI would kill the dialog and record the information to the event log.
- Network components would be needed for TCP/IP communication.
- Remote Desktop Protocol would be needed for remote access during development, since there was no display when final integration and test are being performed.
- Password timeout would have to be disabled, as there would be no need to force a password change.
- The system will auto-logon to an account with administrative privileges.

The next questions turned to the application itself. Specifically, if there are any runtimes, databases, or support components needed:

- Basic system components like Device Manager, Networking, Registry Editor, and Task Manager were to be included.
- Visual C++ redistributable package.
- The main application would be launched via a registry key.
- Windows Explorer shell would be the shell for the system.

My typical final questions turn to security, manufacturing, and field support. Windows 10/11 IoT Enterprise has a lot more security features like virtual base security, secure boot, application control, device driver blocking, BitLocker, etc. Windows XP Embedded only had basic security. Even though the system was isolated, some security features were added. There would be nothing to add or architect for field support, since the system would be working autonomously. The system would not be mass-produced, so the system clone tool would not be needed, but the big issue was how to deploy the Windows operating system. XPe/WES2009 didn't build as an installer like the new Windows Embedded/IoT releases do. The actual operating system files were the output from the development tools. The operating system files need to be copied to the target to run through First Boot Agent (FBA). A WinPE image based on Windows 7 would be used to deploy the pre-FBA image to the target hardware. This turned out to be a useful process for development. A pre-FBA image could be sent for any change requests without the need for hardware.

Sometimes clients get a little embarrassed when they don't know the answers to the questions. Not having all the answers is fine since this is why we want to have the upfront discussions. There is always going to be something that was not thought of. The remaining items get fleshed out during the development phase. For this project, no one has ever created a tank measurement system using Windows that is going into space. There were sure to be some changes along the way.

From Project Requirements to Development Process

The Windows IoT development workflow is emphasized in my latest Windows 10 IoT Enterprise book. I have seen clients make Windows Embedded/IoT development more complicated than it has to be. The Windows Embedded/IoT development workflow started with Windows NT Embedded and continues to evolve even today. XPe/WES2009 have a bit of a twist compared to Windows 10/11 IoT Enterprise. Windows XP Embedded was broken down into 10,000 components. 1,000 components were for the operating system, and the rest were device drivers. Image sizes could vary from 100MB to a couple of gigabytes, depending on what was added. Component creation is required for XP Embedded. For Windows 10/11 IoT Enterprise, the distribution share is used rather than components.

Even though the device is going to an extreme environment, the same development workflow was applied. The project started with the installation of Windows XP on the target system to gather all the device drivers and some other items. Device drivers and runtimes were then put into components. Two macro-components were created to manage all of the components put into the Windows image. One macro-component was for the drivers, and the other macro-component was for the OS and application components. Architecting the build process this way allowed for quick changes during development. The ability to control what gets put into the operating system from

build to build helps to create a stable, reliable operating system. Unneeded components for things like games, Netware, Microsoft Mail, and many others were not included in the build.

Custom Windows Embedded/IoT development is an iterative process. A minimal XPe/WES2009 image was built and tested first. Once the device drivers and runtimes were verified, the next step was to add more components and support applications that were required by the specification. Finally, the focus turned to the lockdown features. This iterative process creates checks along the way to verify that the system was operating as expected. After the final tests were completed, the first custom Windows operating system was sent to NASA.

As expected, over the next 7 months, NASA came back with some additional changes as they developed the main application. Items like turning off daylight saving time, adding a network share, adjusting the write filter cache size, determining libraries to be installed, and applying device driver updates were among the activities. With the exception of the file folder share, the changes were easy to make using the component architecture. Each iteration moved closer to the bullseye, and eventually, the custom operating system met all of the project's needs.

My part of the project ended in December 2015. NASA continued development and integration for the next couple of years. I kept in touch to see if any issues cropped up. The Windows operating system was working well, and they put the system through a few power cycles to check its robustness. The system architecture was working.

First Mission: RRM3

On December 8, 2018, RRM3 launched on the SpaceX CRS-16 resupply service mission[7]. I didn't hear anything for the first couple of months, and eventually reached out. The RFMG data gathering was working perfectly. The XPe/WES2009 image was working as expected, but with one exception: time. For some reason, Windows time was jumping forward several hours. A PC's time can drift, but not this significantly. There were no reports of time issues other than the daylight-saving time and the write filters. Daylight saving was turned off, so this was not the problem. My first thought was that something was wrong with the RTC or the battery. NASA informed me that the RTC battery was not installed because it cannot survive the environment in space. Since the RTC was not battery-backed up, the time was set from the RRM3 computer after system boot. The issue didn't show up in ground testing, so something happened in space. Different problems could be at play, such as a cosmic ray hit or the board's clock crystal not working in microgravity. The rest of Windows and the application were still working normally. The simple solution was to have the RRM3 internal flight computer set the time more frequently. Eventually, the mission came to an end when a different part of the RRM3 failed, and the fuel under measurement had to be released into space. RFMG and the Windows operating system continued to operate until the end.

Second Mission: Intuitive Machines Nova-C Lunar Lander

The RRM3 proved the RFMG technology was viable. The next big step would be to test the technology on a spacecraft. A year or so later, I was told there was a possibility that the RFMG and the Windows operating system would be going to the moon. I didn't hear anything more until February 2024, a few days before launch. RFMG and the Windows operating system were integrated into Intuitive Machines Nova-C Lunar Lander[8]. RFMG was used to measure the liquid oxygen and methane tanks[9]. The original image, sent nine years ago, was on board. The IM-1 was launched on February 15, 2024, and had a rough landing on the moon on February 22, 2024. The RFMG and Windows operating system survived the whole trip. The propellant tanks were measured from launch to orbit, to cruise, to landing, and continued to work even when the IM-1 tipped over. The full RFMG results report is part of the references below[10]. There was still the issue with time, but NASA was able to address it. With the technology proven, RFMG technology will move on from using a Windows operating system to a more integrated computing solution.

Solid Development Process, Solid Architecture, Successful Results

A successful project starts with asking all the right questions to drive the system architecture, and then follow up with the right development process to build the custom Windows image. The NASA

RFMG project put a Windows device into the most extreme conditions possible, and the custom Windows operating system completed the mission it was designed for. It doesn't matter if the project is a medical device, industrial control, entertainment system, or something going into space, the same Windows Embedded/IoT development workflow is performed.

Even though I have not made it into space myself, as my astronaut application seemed to have gotten lost, at least I made a tiny contribution to our space program.

References

1. Investigation of Propellant Sloshing and Zero Gravity Equilibrium for the Orion Service Module Propellant Tanks
2. Propellant Tech Could Fuel Long-Duration Missions
3. An RF Sensor for Gauging Screen-Channel Liquid Acquisition Devices for Cryogenic Propellants
4. RRM 1 & 2 - NASA
5. Robotic Refueling Mission 3 - NASA
6. C1Po1B-01 RRM3 Overview Presentation V3
7. Robotic Refueling Mission - Wikipedia
8. NASA Tests New Spacecraft Propellant Gauge on Lunar Lander
9. Intuitivemachines Instagram
10. Results from the Radio Frequency Mass Gauge Technology Demonstration on the Intuitive Machines Nova-C Lunar Lander / PowerPoint Presentation