OEM6® Family
Application Programming Interface (API)

USER GUIDE

OM-20000140 **Rev 2** **June 2014**

OEM6 Family Application Programming Interface (API) User Guide

Publication Number: OM-20000140
Revision Level: 2
Revision Date: June 2014
This manual reflects firmware version 6.400 / OEM060400RN0000

Proprietary Notice

The software described in this document is furnished under a licence agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or non-disclosure agreement.

Information in this document is subject to change without notice and does not represent a commitment on the part of NovAtel Inc. The information contained within this manual is believed to be true and correct at the time of publication.

OEM6 and NovAtel are registered trademarks of NovAtel Inc.

FlexPak6 and ProPak6 are trademarks of NovAtel Inc.

All other brand names are trademarks of their respective holders.

Table of Contents

Notices

Scope

This document contains sufficient information on working with the Application Programming Interface (API) to be able to develop applications for NovAtel's OEM6 family of receivers. It does not provide specific details of the functions offered by the API but rather information on building and loading applications. Details of the API functions can be found in the API header file.

Conventions

The conventions used throughout this document are:

< > Text displayed between < and > indicates a variable parameter.

In tables where no values are given, such fields should be assumed to be reserved for future use.

	Information to supplement or clarify
	Caution—an action, operation or configuration may result in incorrect or improper use of the product
	Warning—an action, operation or configuration may result in regulatory noncompliance, safety issues or equipment damage

Customer Service

If you have any questions or concerns regarding the OEM6 API, please contact NovAtel Customer Service using any of the following methods:

NovAtel GPS Hotline:	**1-800-NOVATEL** (U.S. & Canada) (1-800-668-2835) 403-295-4900 (International)
Fax:	403-295-4901
E-mail:	support@novatel.ca
Website:	www.novatel.com
Write:	NovAtel Inc. Customer Service Dept. 1120 - 68 Avenue NE Calgary, Alberta, Canada T2E 8S5

Chapter 1 Introduction

1.1 Overview

The Application Programming Interface (API) allows you to develop specialized C/C++ applications to further extend the functionality of your OEM6 family receiver. By using the functions provided by the API, along with the commands and logs already provided by the OEM6 firmware, a wide variety of applications can be created.

1.2 Features

The OEM6 API provides the following features:

- The ability to open physical ports on the receiver to interface with external devices
- Support for three virtual ports, to directly send commands to and receive logs from the receiver firmware
- Support for multiple tasks, with varying priority levels
- Message queuing functionality supporting inter-task communication
- Support for semaphores and mutexes
- The ability to control many of the receiver's general purpose input/output (GPIO) lines
- Access to receiver time
- Controller Area Network (CAN) protocol functionality (for receivers that support it)
- Pulse-width modulation control (for receivers that support it)
- SoftLoad interface, which provides functionality for upgrading the receiver firmware
- Ability to store data in Non-Volatile Memory (NVM)
- Network Socket API based on BSD Sockets (available for receivers that support Ethernet)
- File IO APIs for accessing data files within the internal flash memory device (available for OEM638 receivers only)

Refer to the API header file for more information on the features provided by the API.

1.3 Materials Provided - API Development Kit

The API Development Kit supports building applications for the two different processors found on NovAtel OEM6 family of receivers.

The build tree for OEM628 and OEM615 receivers is found in the top level folder "iMX31". The build tree for the OEM638 receiver (also in the ProPak-6 enclosure) is found in the top level folder "OMAP".

Be sure to use the appropriate build tree for the receiver type. Applications built for one type of receiver processor will not run on the other type of receiver processor.

The following is provided in the OEM6 API Development Kit:

- Example code projects for the Green Hills Software (GHS) multi compiler environments
- NovAtel's *WinLoad* utility for loading the application onto the receiver
- Each processor specific build tree contains the following:

- An OEM6 API library file (liboemapi.a)
- An OEM6 API header file (oemapi.h)
- A folder containing the directory structure and files needed to build a proper Green Hills Software (GHS) executable
- Two Windows command line utility programs, *TOSREC* and *DATABLK*, which format the executable for use with a NovAtel receiver

1.4 Requirements to Build and Run an Application

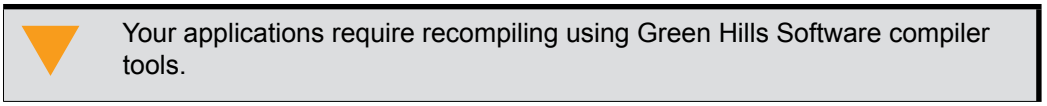
In addition to the items listed in the API Development Kit (see *Section 1.3*), the following is required to build and run an application on an OEM6 family receiver:

- A NovAtel supported complier. Currently, only Green Hills Software (GHS) C/C++ compilers, are supported. The *GHS Software C/C++* compiler is available in the *GHS Multi 6.1 Integrated Development Environment*, for the ARM processor family (version 2012.1). For more information about obtaining GHS compiler tools from NovAtel, please contact your NovAtel sales representative or email: sales@novatel.ca
- OEM6 family receiver running OEM060400RN0000 (6.400) firmware or higher, loaded with an API enabled software model. (See *Section 3.1* on *Page 11* for more information about setting up the receiver for loading an application.)
- A computer with a serial port and a serial cable to load the application onto the receiver. Versions of the compiler are available for both Linux and Windows.
- In addition to this manual, the following manuals can be downloaded from out web site:
  OEM6 Family Firmware Reference Manual (OM-20000129)
  OEM6 Installation and Operation Manual (OM-20000128)

1.4.1 Firmware Compatibility

Applications built for previous versions of NovAtel OEM products, will not run without first being recompiled with the new API. Although every effort was made to retain functional compatibility, it is important to check the Release Notes for the list of changes to the API since the last release.

In particular, applications that were built using the Developer Kit for 6.2XX versions of the receiver firmware will not run on receivers loaded with firmware revisions 6.300 or later. Attempting to start such an application on the receiver will product the result: “ERROR: ELF format error".



A description of the changes made to the API since the last version can be found in the Release Notes. To determine what version of the API your firmware supports, log the `APPLICATIONSTATUS` message, which is described further in *Section 4.2* on *Page 14*.

Chapter 2 Designing and Building the Application

2.1 Designing the Application

When designing an application for an OEM6 family receiver, it is important to understand some of the key features of the API as explained in the following sections.

> Review the sample applications provided with the API for more information on how to design your application to work with the receiver firmware.

2.1.1 Working with the Virtual Ports

When communicating through one of the physical or virtual ports, the application must be designed to either:

- Read **all** data sent to the port or
- Configure the port to not transmit data and disable response generation

This is necessary because any incoming data remains in a buffer until it is read and is not automatically discarded if more data arrives, resulting in a build up of waiting data.

To disable transmission and response generation at the port, use the `INTERFACEMODE` command with the `NONE` mode for the *txtype* field and `OFF` for the *responses* field. Refer to the OEM6 Family Firmware Reference Manual (OM-20000129) for more information on this command.

2.1.2 Using the GPIOs

Many of the OEM6 family receivers provide LV-TTL general-purpose input/output (GPIO) signals that can be used by your application. The API header file provides more details on the functions available to control and read these GPIOs. The sections below indicate which GPIOs are provided on each receiver type.

> Refer to the OEM6 Installation and Operation Manual (OM-20000128) for the GPIOs electrical specifications and pin locations.

2.1.2.1 OEM638™

There are two GPIO pins available on the OEM638:

- GPIO_ERROR
- GPIO_PV

2.1.2.2 OEM628™

There are four GPIO pins available on the OEM628:

- GPIO_USER0
- GPIO_USER1
- GPIO_ERROR
- GPIO_PV

To use GPIO_USER0, be sure to disable COM3, which shares the Tx line with USER0 on pin 19.

Refer to the INTERFACEMODE command in the OEM6 Family Firmware Reference Manual (OM-20000129) for more information.

2.1.2.3 OEM615™

The following pin is available on the OEM615:

- GPIO_PV

2.1.2.4 FlexPak6™

There are two GPIO pins available on the FlexPak6:

- GPIO_ERROR
- GPIO_PV

2.1.3 Input Parameters

There are two methods that can be used to pass parameters to an application.

2.1.3.1 Command Line Entry

The application can be designed to accept a single unsigned, 32-bit parameter, which is then entered as part of the command string for starting the application. This parameter could be used, for example, to set the output serial port used by the application.

For more information about entering a parameter when starting the application, see *Section 3.3.1* on *Page 12*.

See *Section 2.1.4* for a note about the use of command line entry input parameters with auto-start applications.

2.1.3.2 DATABLK Entry

Alternately, the `<SNKey>` field, set when the *DATABLK* utility is run on the executable, can be used to store an input parameter, which would then be read from the `VERSION` log by the application.

For more information about the `<SNKey>` field, see *Appendix A.4.2.1, Using DATABLK* on page 22. *Section 4.3* on *Page 16* provides more information about where the field is stored in the `VERSION` log.

2.1.4 Auto-Start Applications

An application loaded onto an OEM6 family receiver can be set to automatically start whenever the receiver is powered up. This option is set using the `<ComponentEnum #>` field of the *DATABLK* utility, which is described in *Appendix A.4.2.1, Using DATABLK* on page 22. For convenience, the macro *START_OPTION* in the customApp.gpj file is used to supply this field value.

When an application is configured to automatically start, the input parameter is fixed as `0`, the priority is set to `PRIORITY_LEVEL1` and the stack size is `10,000`. See *Section 3.3* on *Page 12* for more information.

2.2 Building the Application

There are three basic steps required to build the application and convert it into a format that can be loaded onto a receiver.

1. Compile, link and integrate the application into a binary .elf file
2. Convert the binary file into S-Records
3. Add additional S-records necessary for loading the application

The example build project framework that is supplied under each build tree (iMX31 and OMAP) contains Green Hills build projects that perform all three of the above steps. The sections that follow briefly describe the three steps, with more detailed information found in *Appendix A.4, Creating Loadable S-Records* on page 21 and *Appendix A.4.1, S-Records* on page 21. It is important to review these appendices to determine any special variations for your situation. In most cases the required changes can be made via predefined "Macro" variables in the top level build project file (customApp.gpj)

With the Green Hills compiler tools installed and your PATH set correctly (see *Appendix A.2, Obtaining and Installing GHS Compiler* on page 17), invoke the following command from within the processor-appropriate build tree:

```
gbuild -top customApp.gpj
```

A clean build will result in the file ddProject.hex which can then be loaded onto the receiver.

2.2.1 Compile, Link and Integrate the Application

Compiling and linking of the application follows the steps familiar to developers of C/C++ applications. The Green Hills build project download_as0.gpj contains the list of source files and folders to search for include files and libraries to link against. Normally, one need only add or change the list of source files. The example project provided uses a single source file: download_as0.cpp.

The next step performed by the Green Hills project files are to "Integrate" the executable into a form that the operating system can work with on the receiver. The project file "download.gpj" performs this step and the contents of this file must not be changed. *Appendix A.3, Dynamic Download Object (DDO)* on page 17 contains more detailed information on dynamic downloads, the build folder structure and the steps automatically invoked to create the final executable.

2.2.2 Convert the Binary Executable to S-Records

The Green Hills build project automatically invokes the supplied utility *TOSREC* to convert the Dynamic Download Object into a format that can be loaded onto the receiver. For more detailed information about this process see *Appendix A.4, Creating Loadable S-Records* on page 21.

There should never be any changes required to this process.

2.2.3 Adding Information for Loading the Application

The WinLoad utility is used to load the application .hex file onto the receiver and reads information from the .hex file in order to determine how the application should be loaded. This information is included in a special set of S-Records placed at the beginning of the .hex file loaded onto the receiver.

The provided *DATABLK* utility is invoked by the Green Hills build project "download.gpj" to add these special S-Records. It is very likely that some of the information included in these S-Records will need to be customized for your particular application. For example, the name of the application, whether it is to be started automatically and the serial number key for the application.

The file customApp.gpj contains a set of macro definitions for the values that are most likely to be customized for your project. For most customers, this will be sufficient for their needs. However, for more detailed information about the *DATABLK* utility and the options that can be provided to it, see *Appendix A.4.2, Adding Information for Loading the Application* on page 22.

Chapter 3 Loading and Controlling the Application

Once the application is built and converted to S-record format, with the necessary S-records added, the application can be loaded onto the receiver.

A Windows based utility named *WinLoad* is used to assist with loading the firmware.

3.1 Setting up the Receiver

In order to load and run an application, the receiver must have:

- A model with the API option enabled
- Version OEM060400RN000 (6.400) or higher of firmware loaded

The following sections provide information on how to determine if your receiver meets these criteria and how to update it if it does not.

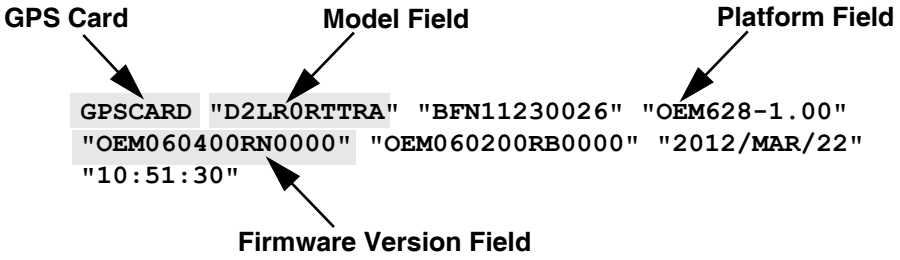
3.1.1 Determining the Current Model and Firmware Version

To determine the current model and firmware version of the receiver, read the `VERSION` log. To do so, send the following command to the receiver:

```
LOG VERSION
```

Read the output provided, specifically the 1st and 4th fields after the word `GPSCARD`. The first field provides the model and the fourth field indicates the version of firmware loaded on the receiver.

Example:



In this example, the last character in the model field ends with an "A" and the firmware version field reads "OEM060400RN0000", therefore an application can be loaded and executed on the receiver. If an update to the firmware and/or model is needed, instructions are provided in *Section 3.1.2* and *Section 3.1.3* of this document.

3.1.2 Updating the Firmware

To update the firmware to a version that supports the API, obtain the following from NovAtel Customer Service:

- The firmware update file, with version OEM060400RN000 (6.400) or higher
- An update authorization code

WinLoad is also required to load the firmware onto the receiver and is included with the API package. Follow the procedure in the `HowTo.txt` file provided with the update file to upgrade the firmware.

3.1.3 Authorizing a Model with the API Option

To authorize a model with the API option enabled, contact NovAtel Customer Service to obtain the necessary authorization code and use the AUTH command to add the code to the receiver. Refer to the *OEM6 Family Firmware Reference Manual* (OM-20000129) for more information on this command.

3.2 Loading the Application

Once the receiver is set up with the necessary model and firmware version, the application can be loaded in to NVM on the receiver using version 1.0.134.000 or higher of the WinLoad utility. This utility is provided as part of the API package or can be obtained from NovAtel Customer Service.

For information on using *WinLoad*, follow the procedure given in the *OEM6 Family Installation and Operation User Manual* (OM-20000128), making adjustments for the selection of the application file, rather than the firmware file and the absent authorization code prompt.

Only one application can be loaded onto the receiver at any one time. However, functions are provided to allow for multiple tasks running within the application. Please see the provided oemapi.h file for more information.

- The application HEX file should be selected, rather than a standard firmware HEX file
- An authorization code is not needed

3.3 Controlling the Application

The operation of the application can be controlled by using the APPLICATION command as discussed below.

3.3.1 Starting the Application

Once the application has been loaded onto the receiver, enter the following command string to start the application:

```
application start <parameter> <priority> <stack>
```

The values that can be entered when starting an application are described in the table below.

Table 1: APPLICATION START Parameters

Parameter	Valid Values	Description
<parameter>	Any ulong value	Optional field to specify an input parameter for the application. If a value is not specified, the default 0 is used
<priority>	Any long value from 0 to 21	Optional field to specify the priority of the application in relation to system tasks. See the API header file for details. In order to specify the priority, a value must be entered for the <parameter> field as well. If a value is not specified, the default 1 is used
<stack>	Any long value from 200 to 20000	Optional field to specify the size of the stack to be used by the application in bytes. In order to specify the stack size, a value must be entered for both the <parameter> and <priority> fields as well. If a value is not specified, the default 10000 is used

3.3.1.1 Auto-Start Applications

If `5` was entered for the `<ComponentEnum #>` field when running the *DATABLK* utility, the application is set to auto-start so when the receiver is first powered, the application begins to run.

An alternate method for setting an application to automatically start is to use the `SAVECONFIG` command after the application has started. When this is done, the application automatically starts whenever the receiver is powered up, as long as the saved receiver configuration is present in memory. However, as soon as the `FRESET` command is issued, the configuration is lost and the application will not automatically start. When an application is configured to auto-start using the method described in *Section 2.1.4* on *Page 8*, it is not affected by the state of the receiver configuration.

3.3.2 Stopping the Application

To stop the application, enter the following command:

```
application stop
```

This command can be used to stop either standard or auto-start applications.

3.3.3 Removing the Application

To remove the application from NVM, enter the following command:

```
application remove
```

The application is not recoverable when removed from NVM.
It is not necessary to remove an existing application from NVM before loading a new application.
When loading a new application, any existing application will be removed from NVM automatically.

Chapter 4 Additional Information

The following sections provide additional information that may be useful when working with OEM6 applications.

4.1 Determining the Version of the Loaded Application

When an application is loaded onto the receiver, the VERSION log provides information about the application. When an application is loaded, an additional entry is displayed in the VERSION log, with the type field showing DB_USERAPP for a standard application or DB_USERAPPAUTO for an auto-start application. All the parameters given in that entry apply to the application loaded on the receiver. An example of the log with a DB_USERAPP entry is shown below.

```
[COM1]<VERSION COM1 0 52.0 UNKNOWN 0 50.046 004c0020 3681 45632
<     2
<          GPSCARD "D2SR0GTT0A" "BFN10470121" "OEM628-1.00"
"OEM060400RN000" "OEM060200RB000" "2014/May/23" "19:45:13"
<          DB_USERAPP "apiexample" "321" "" "1.2" "" "2013/Jun/06"
           "17:01:51"
```

The version log reports a snapshot of the application in memory at the time the device was powered up. Removing the application (refer to *Section 3.3.3, Removing the Application on Page 13*) does not update this snapshot. So, the version log continues to indicate the presence of the application even though it has been removed from memory.

For information on how to capture the VERSION log or the fields it contains, refer to the *OEM6 Family Firmware Reference Manual* (OM-20000129), *Section 4.3* on *Page 16* of this document provides information on how some of the VERSION log fields are set by the *DATABLK* utility.

4.2 Logging the Application Status

A log has been created to capture the details of any application loaded onto the receiver (refer to *APPLICATIONSTATUS API Application Status Information* on page 15). All formats and standard fields, such as the header, are explained further in the *OEM6 Family Firmware Reference Manual* (OM-20000129). Many of the fields are set to values entered for use by the *DATABLK* utility, as described in *Section 4.3* on *Page 16* of this document.

The time indicated in the log header is the time when the application status was last changed, for example, the time when the application was started or stopped.

APPLICATIONSTATUS API Application Status Information

Log Type: Asynch **Message ID: 520**

Field	Field Type	Data Description	Format	Binary Bytes	Binary Offset
1	header	Log header		H	0
2	api version	The version of the API that the currently loaded firmware supports	Ulong	4	H
3	running	Flag indicating whether the application is currently running, where 0 = FALSE 1 = TRUE	Bool	4	H+4
4	base address	The base address is an address in the kernel memory space and is not related to an actual address accessible within the VAS	Ulong	4	H+8
5	size	This value is an over estimation, formed by summing the various code and data areas used by the VAS where the application is running. It is a worst case memory usage measured in bytes	Ulong	4	H+12
6	name	The name of the application	Char[16]	16	H+16
7	version	The version of the application	Char[16]	16	H+32
8	compile date	The date the application was compiled In the format yyyy/mmm/dd, where mmm is three letters for the month (eg. JAN)	Char[12]	12	H+48
9	compile time	The time the application was compiled In the format hh:mm:ss	Char[12]	12	H+60
10	xxxx	32-bit CRC (ASCII and Binary only)	Hex	4	H+72
11	[CR][LF]	Sentence terminator (ASCII only)			

Recommended Input:

```
LOG APPLICATIONSTATUSA
```

ASCII Example:

```
#APPLICATIONSTATUSA,COM1,0,72.5,UNKNOWN,0,8.181,00000020,3EFF,754;1
4,FALSE,00000000,00000000,"SAMPLEOEMVAPP","1.0","2006/JAN/
31","14:23:54"*77BB5E52
```

4.3 DATABLK Parameters Used in Logs

Many of the fields captured in the APPLICATIONSTATUS log and the DB_USERAPP or DB_USERAPPAUTO entry of the VERSION log are set to the values entered for parameters used by the *DATABLK* utility. The table below provides a list of these parameters and the matching log fields. For more information about the *DATABLK* utility, see *Appendix A.4.2, Adding Information for Loading the Application* on page 22.

Table 2: DATABLK Parameters Used in Logs

DATABLK Parameter	Matching Log Field	
	Version[a]	Application Status
<Name>	model	name
<SNKey>	psn	N/A
<Version>	sw version	version
<Compile Date>	comp date	compile date
<Compile Time>	comp time	compile time

a. Only valid for the DB_USERAPP or DB_USERAPPAUTO entry in the log.

Appendix A Green Hills Software Compiler Tools

This appendix contains steps to build the Green Hills Software (GHS) project which creates a Dynamic Download Object (DDO). The NovAtel GNSS receiver only supports user applications in the form of a GHS DDO. Although an ELF executable (for ARM processors) created by other tools can be loaded onto the card, such applications will fail to run when an attempt is made to start them.

The ELF file containing the user application is converted to a relocatable S-record file and is loaded into the receiver's flash memory using the *WinLoad* utility. To start the application the receiver loads it from flash at runtime. Once running, the application communicates with the firmware via the NovAtel API, provided in the liboemapi.a archive file.

A.1 GHS Compiler

The specific format of the Dynamic Load Object requires the use of the GHS software tool chain for ARM processors. For compatibility reasons, NovAtel recommends using version 2012.1 or later of the compiler tools.

A.2 Obtaining and Installing GHS Compiler

Obtain and install the GHS compiler. Customers who do not already have the required GHS tools can obtain them from NovAtel. You will receive a link to the NovAtel support web site to download the tools. Follow the instructions provided in the download to install the GHS tools.

After installing the GHS compiler tools, follow the instructions to obtain and install a password from Green Hills Software to activate the compiler. You will need to indicate you are a customer of NovAtel, which GHS will confirm at the time of the request.

A.3 Dynamic Download Object (DDO)

A GHS Dynamic Download Object (DDO) is a self-contained application that runs within a separate Virtual Address Space (VAS) on the NovAtel GNSS receiver. This address space is outside the kernel address space and protection barriers exist to prevent unintentional accesses from corrupting the operation of the receiver. When `application start` is invoked, a VAS is created for the application. This VAS exists until the `application stop` command is issued at which point all tasks within the VAS are halted and the VAS and everything in it are deleted.

A.3.1 Building the Dynamic Download Object

Once the compiler tools have been installed they can be used to build the user application which produces a DDO in the form of an ELF file. A template directory structure, with an example application, is provided to assist with the creation of a user application. The contents of the template directory are described below. In most cases, customers need only to modify one or two of the files besides the source code file.

A.3.1.1 Download Structure

The important folders and files are as follows:

- Folder **src**: contains the source files for the project. For the example application, a single file, download_as0.cpp is provided. Additional source files can be added to this folder and the project file updated with the additional entries (described below). One of the source files must contain a function MainTask() which is the entry point for the application.
- Folder **include**: the example project has this folder in the search path for #include files. As shipped, the folder contains the **oemapi.h** file, which describes the API interface.
- Folder **libs**: the example project searches this folder for library files. As shipped, the folder contains the library liboemapi.a which contains the API implementation.
- File **customApp.gpj**: This is the GHS top level build project file. The top of this file contains several “macro” definitions which are passed to the *DATABLK* utility. These definitions can be changed to tailor the application to your needs. The lines that follow these macro definitions should not be changed.
- File **download.gpj**: This is the GHS project file that transforms the executable into a VAS DDO. This file should never require modification.
- File **download_as0.gpj**: This is the GHS project file that builds the application. This file can be modified to add or change source files or libraries. Source files are added to the end of the file. Additional libraries would be added in the compiler options section.
- File **tgt/download.int**: This is the Integrate file for the user application. This file contains information regarding the memory structure of the application and the VAS. To customize the amount of memory used by the VAS, this file needs to be modified (refer to *Section A.3.2.2* on page 20).

A.3.2 Building the Project

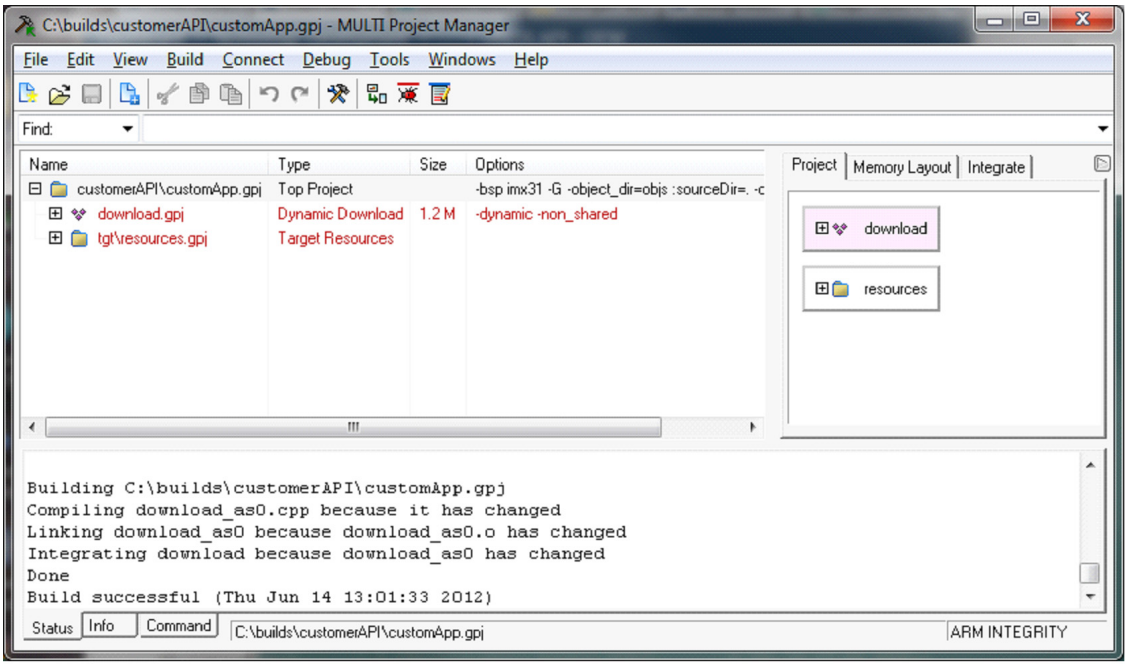
There are two options to build the project. The command to build the project can be invoked directly on the command line. Alternatively, if the graphical GHS MULTI tools are available, the project can be opened in the MULTI Project Manager and built from there. MULTI is available from Green Hills Software at additional cost. Not all of the features available in MULTI (for example, run-time debugging) can be used on the NovAtel Receiver due to security restrictions in how the operating system has been set up.

To build using the command line, the following command must be invoked from the base of the template directory `gbuild –top customApp.gpj`.

```
$ gbuild –top customApp.gpj
Building C:\builds\customerAPI\omap\customApp.gpj
Generating headers from download.int because download.time does
not exist
Compiling download_as0.cpp because download_as0.o does not exist
Linking download_as0 because it does not exist
Integrating download because it does not exist
Output from integrating download.gpj:
*
* datablk - NovAtel Inc. data block utility n
* Executable Version: 2.21
* Header Version: 2
*
Processing download.srec to ddProject.hex
Done
```

The result of a successful build is the file **download**. This is an ELF file containing the DDO. After successfully building the DDO, the project build script invokes the *DATABLK* and *TORSEC* utilities to prepare the file for loading onto the receiver (see *Section 2.2.2* on page 9 and *Appendix Adding Information for Loading the Application*, starting on page 22). The prepared file is then loaded onto the receiver using the *WinLoad* utility (*Section 3.2* on page 12).

To build using the MULTI Project Manager, double click on the customApp.gpj project file to open the MULTI Project Manager. Select customApp.gpj and press F7 to launch the build process.



A.3.2.1 Memory

There are two user application memory limitations to be aware of:

- The size of the download ELF file cannot be larger than 1 MB (1048576 bytes). This refers to the DDO file, not the ASCII .hex file.
- The size of the user application VAS cannot be larger than 1 MB (1048576 bytes)

When the user application is first started the entire ELF file is loaded to receiver memory. If the receiver does not have enough memory to load the ELF file, or if the ELF file is larger than 1 MB, the application will not start. An error indicating there is not enough memory is issued.

The second memory limitation is the size of the VAS. All of the memory that will be used by the user application must be allocated up front as part of the VAS.

It is important that the size of the VAS be correct. If there is not enough memory allocated at the start, the user application may encounter errors at runtime when it attempts to allocate additional memory and no free memory is left. If the VAS is too large memory is wasted and in some scenarios, the entire GNSS application may suffer degraded performance. As a safety measure, the receiver limits the maximum VAS size to 1 MB.

The following contribute to the size of the user application VAS:

- User application text, data and bss segments
- User application heap

- User application Free Memory Pool

The Free Memory Pool is memory allocated in the VAS that is used by the operating system to store task objects, semaphores, task stacks and other operating system objects created by the user application. The size of the Free Memory Pool is set in the download.int Integrate file (described in the following section).

A.3.2.2 File Download.int

The Integrate file is located at tgt/download.int and contains three values of interest to the developer. The following variables control the amount of "dynamic memory" allocated for use within the VAS and may be adjusted to fit the application's needs. Because the basic size of a memory page in the MMU is 4096 bytes, all of these values should be multiples of 4096 to avoid waste (the Integrate application will round up and issue a warning in any case). Each of these values affects the total size of the VAS. If the VAS is too large, it may not load and no customer application will run.

- MemoryPoolSize is the size of the Free Memory Pool for the VAS
- HeapSize is the amount of memory to allocate for the heap when the VAS is created. This space does not consume memory from the Free Memory Pool
- HeapExtensionReservedSize is the amount of virtual memory address space to reserve contiguous to the initial heap area. Once the heap is exhausted additional memory for the heap will automatically be allocated from unused pages in the Free Memory Pool and assigned virtual addresses in this space. It is permissible for the heap extension size to be 0, in which case virtual addresses used for additional heap memory may not be contiguous with the initial heap area. The size of this potential additional heap usage should be accounted for when calculating MemoryPoolSize

The variable StackLength sets the size of the stack for the initial task in the VAS. This is not the size of the stack for MainTask and should not be changed. No other variables in this file should be changed.

A.3.2.3 Guidelines For Sizing The Memory Variables

If the user application attempts to use more memory than the VAS can provide, the request will fail.

If the sum of MemoryPoolSize + HeapSize + Application Text + Application Data + Initial Stack is larger than 1 MB the VAS will not load and the user application will not run.

By default, the amount of memory allocated for the VAS Free Memory Pool is 0.5 MB. This space must be large enough to contain all of the stacks for tasks created by the user application, the HeapExtensionReservedSize (assuming the heap is fully used) and any objects created by the INTEGRITY operating system (for example, Task Control Blocks, operating system object and the like). Each task requires one page (4096 bytes) plug 64 bytes for each operating system object it creates.

There is a single heap area within the VAS which is shared between all tasks running in the VAS. The default size of this heap is 128 KB, with a contiguous extension of 4 KB. These values can be changed as required. Within the API, the message queue module allocates memory from the heap and must be taken into account. The amount of memory required is (NumOfMessages * (MaxMsgSize + 8))+128

None of the API calls require more than 1 K of stack space for internal automatic variables.

A.4 Creating Loadable S-Records

When loading the application onto the receiver, the S-Record format, which is described in *Section A.4.1*, is used. Therefore, once the application has been built, the resulting binary ELF file must be converted to S-Record format. The *TOSREC* utility is provided to complete this conversion.

A.4.1 S-Records

The S-Record format is an industry standard for encoding programs or data files in a printable form that allows for ease of transfer between devices.

An S-Record is an ASCII character string consisting of five fields, in the format shown below.

```
<type><length><address><data...><checksum>
```

The S-Record fields all use hexadecimal format, except for the `<type>` field. The fields are described in *Table 3* below.

Table 3: S-Record Fields

Field	Length (Characters)	Description
`<type>`	2	The type of S-Record, as described by *Table 4*
`<length>`	2	The number of character pairs in the record, excluding the `<type>` and `<length>` fields
`<address>`	4, 6, or 8	The 2, 3 or 4-byte address at which to load the contents of the data field in memory
`<data>`	variable	Executable code or memory loadable data
`<checksum>`	2	The least significant byte of the one's complement of the sum of the values represented by the length, the address and the data fields

There are 3 types of S-Records used for OEM6 applications, as shown in *Table 4* below.

Table 4: S-Record Types

Type	Description
S0	Header record is used by WinLoad to determine how to load the application as described in *Section A.4.2.2* on page 23
S3	Data record
S7	End-of-file record

Typically, each S-Record file consists of one or more header records, followed by one or more data records, concluded with a single end-of-file record.

A.4.1.1 Using TOSREC

The *TOSREC* utility is a Windows command line program. To run the application, enter the following in a command window:

```
tosrec <infile>
```

where `<infile>` is the name of the file to be converted. The name of the S-Record file to be generated can be specified using the following optional parameter:

```
-o <outfile>
```

where <outfile> is the name of the output file.

Examples of command strings to run the utility are shown below.

```
tosrec download
tosrec download -o output.hex
```

A.4.2 Adding Information for Loading the Application

The *WinLoad* utility is used to load the application onto the receiver and read information from the input file in order to determine how the application should be loaded. This information is included in a special set of S-Records placed at the beginning of the file. Once the application data has been converted to an S-Record format, the *DATABLK* utility is used to add these necessary records.

A.4.2.1 Using DATABLK

The *DATABLK* utility is a Windows command line program. To run the application, enter the following string in a command window:

```
datablk <In SREC File> <Out SREC File> <Compress> <Block #>
<ComponentEnum #> <Name> <Version> <SNKey> <Platform> <Compile Date>
<Compile Time>
```

Each of the parameters are described in the table below.

Table 5: DATABLK Parameters

Parameter	Valid Values	Description
<In SREC File>	Any	File name of input file
<Out SREC File>	Any	File name of output file
<Compress> [a]	Raw	Specifies the application is to be loaded as is
<Block #>	2	The data block in memory in which the application will be loaded into
<ComponentEnum #>	1 or 5	The type of application, where 1 specifies a standard user application and 5 specifies an auto-starting user application
<Name>	15 non-null characters or less	A string indicating the name of the application
<Version>	15 non-null characters or less	A string indicating the version of the application
<SNKey>	15 non-null characters or less	A string indicating the serial number or key for the application. Can also be used to set an application parameter. See *Section 2.1.3.2* on page 8 for more details
<Platform>	15 non-null characters or less	Indicates which hardware platform this application is to be loaded on. This string should be the same as the “hardware platform” returned in the “hwversion” field of the LOGVERSION command
<Compile Date>	Any valid date in the format yyyy/mmm/dd, where mmm is three letters for the month (e.g., JAN)	Optional field to specify the date the application was compiled. If no value is provided, the computer’s current date is used

Parameter	Valid Values	Description
<Compile Time>	Any valid time in the format `hh:mm:ss`	Optional field to specify the time the application was compiled. If no value is provided, the computer's current time is used

a.Only use `Raw` as the value for this parameter. `Compress` is not a valid option for Dynamic Download applications.

An example command string to run the utility is given below:

```
datablk input.hex output.hex raw 2 1 SampleApp 1.00 1234 OEM628
```

See *Section 4.3* on page 16 for more information about how the values entered for the *DATABLK* parameters are used in the receiver logs.

A.4.2.2 Records Used By WinLoad

DATABLK adds three header records that are required by the *WinLoad* utility when loading the application. The records provide the following information to *WinLoad*:

- The target platform of the OEM6 family receiver (provided via parameters)
- The version of the application
- In which data block to load the application (block 2)