Custom Logic User Guide

VA 1051000 | Parker Vehicle System Gateway

1051G05.03A

Parker Hannifin Canada
Electronic Controls Division
1305 Clarence Avenue
Winnipeg, MB R3T 1T4 Canada
Office   204 452 6776
Fax   204 478 1749

http://www.parker.com/ecd
ecdinfo@parker.com

Table of Contents

Revision History

Rev #	Release Case	Date
00A	Case 48454	2018-Jul-06
01A	Case 49554	2018-Aug-03
02A	Case 50354	2018-Oct-09
03A	Case 52864	2018-Dec-20
04A	Case 53563	2019-Feb-01

1. Overview

Developers are able to create 3rd party "custom logic" applications that run on licensed PVSGs. There are two types of custom logic applications for the PVSG:

1. Programming protocol plugins (system update and/or module configuration)
2. Full "custom application" logic

These custom logic applications are written in C/C++ and compiled with the Parker-provided custom limited toolchain; It is important to note that use of the toolchain requires access to a machine running Linux, though it is perfectly possible to do the development activity under Windows via an IDE which is capable of remote building. Functionality is accessed via a Parker-provided Software Development Kit (SDK) through the defined API, and must implement and adhere to a Parker defined interface. Custom logic is deployed with a PVSG configuration, which puts the management ability with the customer as opposed to the Parker platform developers.

2. Programming Protocol Plugins

Programming protocol plugins are used to update/modify the software of a node connected to the PVSG using CAN. Custom plugins are installed via the configuration tool under the "Custom Plugins" section in "Module Management", and programming via the plugin is started via the PVSG Dashboard.

3. Custom Application Logic

Custom application logic is different as it will run on startup, and can be used to perform any tasks available through the SDK. The "Custom Application" section in the configuration tool supports adding Custom Applications to a PVSG.

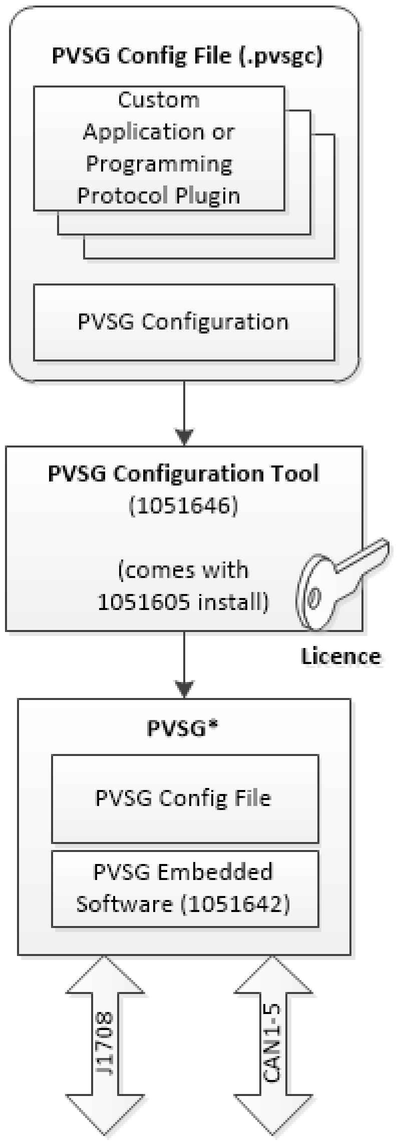

PVSG Config File (.pvsgc)
Custom Application or Programming Protocol Plugin
PVSG Configuration
PVSG Configuration Tool (1051646)
(comes with 1051605 install)
Licence
PVSG*
PVSG Config File
PVSG Embedded Software (1051642)
J1708
CAN1-5

Figure 1: Custom logic overview diagram

* Custom Logic requires a PVSG Variant that supports Custom Applications/Programming Plugins or a PVSG that is licensed to run Custom Applications/Programming Plugins.

4. Usage Guide

4.1. Getting Started

In order to develop custom logic for the PVSG, the following is required:

1. A suitable editing environment. An Integrated Development Environment (IDE) is advised, and Parker recommends the use of NetBeans for development under Windows, which is open-source and free for commercial use. Instructions for configuring Remote Build in the NetBeans environment can be found on the NetBeans website.
2. A copy of the correct build toolchain supplied by Parker. This must be installed on a suitable host machine running Linux, and must be accessible from the development machine via TCP/IP. Contact Parker for more details.
3. A copy of the PVSG custom application template (10516C2). This must be used as the starting point for all projects, since it contains all of the necessary files and settings required to facilitate a successful build. The template is available as 10516C2 (see "Software Release Notes" section below for more version history). The template version should correspond to the version of 1051642 which the developer intends to use. A 'newer' version of 1051642 will support an 'older' version of 10516C2, however the opposite is **not** true.

Core PVSG services are accessible to a custom application by means of a PVSG SDK which has a defined API. An SDK API reference document is located in the 'documents' directory of the custom application template. It is advised that template users familiarise themselves with this documentation prior to continuing.

Each custom application must provide the following entry points with *C-style linkage*:

Entry Point	Return Information	Description
void* GetHandle()	A pointer to the application instance.	Returns a pointer to an instance of the custom application which implements the ICustomApplication interface. If an instance does not exist, it should be created.
void Destroy(void* instance)	Nothing.	Destroys the application instance pointed to by @instance.
int32_t GetSDKVersion()	Returns the SDK version with which the custom application was compiled.	Unless otherwise directed, this function should return the value of the CUSTOM_SDK_VERSION definition, which can be found in PVSGInterface.h.

A custom application/plugin must exist as a single class which implements the ICustomApplication interface. This is defined by Parker, and its definition is provided in the custom application SDK. The ICustomApplication interface is implemented as a pure virtual class, which implicitly causes the developer to implement the requirements of the interface, since otherwise the compilation will fail. The custom application is compiled into a dynamic shared object, which is then converted into a proprietary Parker format. Note that details on the format remain confidential to Parker for security reasons.

The custom application template (10516C2) is configured to provide a template application which has the minimum functionality required to compile. This template should be used as the base for all custom applications.

After a successful compilation, the resulting binary must be converted into a Parker format which can be used on the PVSG. This involves providing a set of metadata, which contains information about the application, and a code signing certificate with which to sign the application. Each application must be signed by a certificate issued by Parker, to ensure the validity, integrity and security of the custom application. A unique code signing certificate will be issued, by Parker, to each customer who is authorised to develop custom applications. Note that certificates are issued in such a way which allows each compiled custom application to be traced back to the developer who signed it. For this reason, it is imperative that the security of any issued signing certificates be maintained at all times. They must not be shared, and should only be used by the persons to whom they were allocated.

To produce the final custom application which can be loaded into a PVSG configuration, the Custom Application Creator tool must be used. This can be found in the 10516C3 PVSG Developer Tools Installer package, which is available from Parker. Note that a licence is required in order to use the developer tools. To obtain a licence:

1. Install the 10516C3 developer tools installer package.
2. Open the PVSG dashboard, open the dock and select "Licences". Click "Developer Tools", and send the generated challenge value to Parker.
3. Paste the provided response into the tool.
4. The authorised features will now be licensed.

The 'make_build.bat' batch file which is provided with the 10516C2 template can be used to create the final build. Note that this batch file is provided as an example, and must be modified to suit each developer. As a minimum, the following must be modified:

- Application name
- Application description
- Signing certificate path
- Signing certificate password

Other fields can be modified at the developer's discretion. When using make_build.bat, the version information must be passed to the batch file when called. To run the script, call make_build.bat <major> <minor> <build> <issue>. The version number will be composed of <major>.<minor>.<build>.<issue>.

Note that the result of running cabuild.exe should be carefully inspected to determine whether the build was successful or not. The return codes and tool command line options can be found by running cabuild.exe --help from the command line.

Following a successful build creation, a file with the extension .pvsgapp will be available. This file can be deployed to a PVSG using the PC configuration tool (1051646). Note that a maximum of 10 custom applications and 20 custom programming plugins may be added to a single configuration. To add a custom application, the "Custom Applications" applet should be used. To add a custom programming plugin, see "Custom Programming Plugins" under the "System Modules" applet.

The PVSG console, via RS-232 connection, can be used to aid custom application debugging. Note that a console login is not required in order to debug a custom application, and as such, no console logins will be provided externally to Parker.

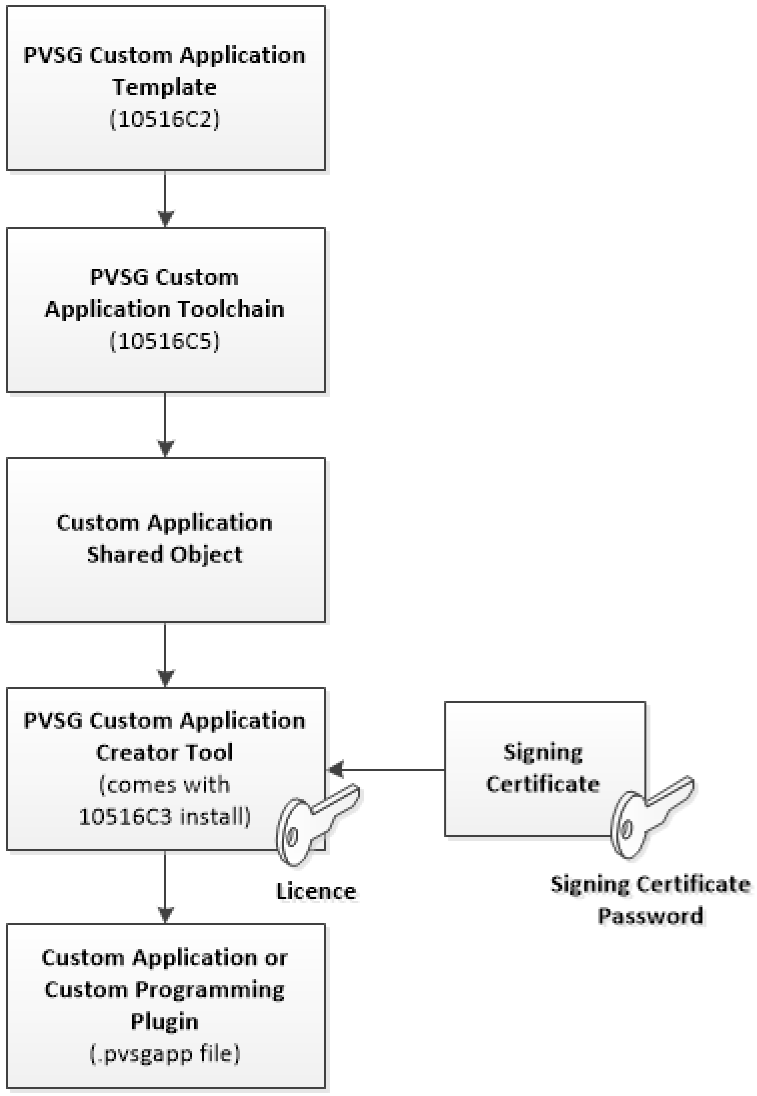

PVSG Custom Application Template (10516C2)
PVSG Custom Application Toolchain (10516C5)
Custom Application Shared Object
PVSG Custom Application Creator Tool (comes with 10516C3 install)
Signing Certificate
Licence
Signing Certificate Password
Custom Application or Custom Programming Plugin (.pvsgapp file)

Figure 2: Custom logic development diagram

4.2. SDK

All SDK services are accessed via an instance of a PVSGInterface object. A suitable instance is passed in the ApplicationInit() function defined by the ICustomApplication interface, via the @pvsg argument. The value of this argument should be cached, and used throughout the lifetime of the application to access SDK services. The application must not create a PVSGInterface instance itself; doing so will result in failure, and creation of an instance which cannot be used. The SDK services available are per the following table, and depend upon the application type.

	Custom Application	Programming Plugin
J1939	Yes	Yes
J1708	Yes	Yes
CAN	Yes	Yes

	Custom Application	Programming Plugin
Virtual EEPROM	Yes	No
Plugin	No	Yes

Attempting to access a service which is unavailable for the application's type will result in an -EACCES error. In general, a return code of 0 indicates success. A negative value generally refers to an error code, the absolute value of which corresponds to a generic POSIX error code. A definition for POSIX error codes can be found in a suitable Linux manual. The function documentation in the SDK API document should be checked to confirm the meaning of the return code for a particular function.

The ICustomApplication interface presents two primary access points into a custom application:

- ApplicationInit() is called during application initialisation. This function should be used for application setup purposes, and should not run general application logic. Prior to returning, this function should schedule at least one task, otherwise the application will terminate.
- ApplicationShutdown() is called by the platform to tell the application to shutdown. The application should clean up its resources here, and terminate gracefully. Note that if the application fails to terminate gracefully within a period of time, it will be abruptly terminated.

4.2.1. Threading

Custom applications are inherently single threaded. It is not permitted for a custom application to spin multiple threads, or fork additional processes. Instead, a basic scheduler supporting the concept of 'tasks' is provided. Each application may schedule up to 10 basic tasks. A task may run periodically, at a period defined by the caller, or singularly, after a delay specified by the caller. It should be remembered that the scheduler effectively runs within a single thread, and operates cooperatively (as opposed to preemptively). This means that application tasks must yield (i.e. return) as soon as they have performed their required operations. If an application task takes too long to complete, the application will be abruptly terminated by the platform.

In addition to basic tasks, up to 2 asynchronous tasks may be utilised by an application. Asynchronous, or "async", tasks operate in a similar fashion to POSIX threads, though with some limitations imposed by the platform. An async task is not required to yield, and the application may remain inside the task context until it wishes to exit, assuming one of the following is done:

1. The async task "checks in" with the platform at least once every 2 seconds, OR
2. The async tasks disables the platform watchdog for that particular task.

Refer to the API reference documentation for details on how to perform the above. Note that the stack size associated with a single async task is limited to 16KiB. If an async task fails to check in on time (or does not disable the watchdog), the application will be abruptly terminated by the platform. CPU utilisation within an async task contributes to the overall calculated CPU usage of a single application.

4.2.2. Locking

Although the custom application task scheduler is inherently single threaded, 'callback' functions from within the SDK are executed in a context other than the primary application context. For this reason, SDK callback's should generally be considered to be akin to 'interrupt context', and should be treated with suitable caution. The application should return from this context as soon as possible, and not start any long running operations. Importantly, SDK services should not be accessed within an SDK callback, since this has the potential to cause a deadlock.

Since it is possible for SDK callback's to be raised in a thread other than the primary application thread, suitable consideration should be given to 'locking'. A lock is an abstract concept, which has the intent of protecting access to a shared resource. Locking is required in order to avoid race conditions and deadlocks (amongst other things) which can occur due to non-atomic operations on shared data structures. std::mutex is one way of providing suitable protection. Any good C++ reference book can be referred to for more details on locking.

4.2.3. Security

For security reasons, each custom application runs in its own sandbox environment in the security context of a restricted user. This means that an application may not directly access system resources, platform services or another custom application. All resource access must be made via the SDK. Custom applications are not allowed to:

- Open, read, write and/or close files on the file system.
- Spin threads using the POSIX threading library.
- Fork additional processes using fork().
- Open, read, write and/or close pipes.
- Allocate RAM beyond the defined threshold.
- Consume the CPU beyond the allocated maximum.
- Shutdown or reboot the system.

For all intents and purposes, the custom application should consider itself to be running on a basic microcontroller system, and not within a Linux platform.

4.2.4. Persistent Storage

Per the Security section, a custom application may not read and write files on the filesystem. If an application requires to store data in a persistent fashion, it must use the virtual EEPROM service provided in the SDK. This gives each application up to 5MiB of storage which will persist, where possible, between product software upgrades and configuration updates. The allocated storage can be used as the application requires; the platform imposes no structure restrictions or requirements. Note that the ability for the storage to persist between configuration updates relies on the custom application UID remaining the same. The UID is allocated by the configuration tool when the custom application object is created. To maintain this, the element must not be deleted from the configuration tool. To upgrade the application, the binary on the existing object must simply be replaced. This will ensure that the UID remains in tact and that the storage will persist. If the PVSG platform detects that a virtual EEPROM store is no longer required, it will be automatically removed.

4.2.5. Resource Control

The PVSG platform services monitor several metric components of running custom applications. This includes, but is not limited to, the following (refer to document 1051G02, *Software User Guide* for more detail on error codes).

Metric	Description	Error Code
CPU usage	Each custom application may not consume, on average, more than 40% of the CPU idle time.	21:2
Memory usage	The total memory footprint (both statically and dynamically allocated) of each custom application may not exceed 5 MiB.	21:2
Last callback time	Each custom application must 'call back' to the platform at least once every 1500ms. This is performed implicitly by the scheduler, on the assumption that the application does not stall in a scheduled task.	21:5
Critical section execution time	Each custom application must return from 'critical sections' within 800ms. If a custom application remains in a critical section for a period of time greater than 800ms, it will be abruptly terminated. A critical section is defined as a piece of custom application code which is being executed from the context of the PVSG platform. This generally means creation and deletion methods of the core application interface.	21:5
Shutdown time exceeded	If the application is requested to perform a graceful shutdown, it must do so within 2000ms. If the application fails to shutdown within this time, its process will be abruptly terminated.	21:5

In the event that one of the above control restrictions are violated, the application will be terminated. Depending upon the violation, a graceful shutdown *may* be requested first. If the graceful shutdown request is not processed in a timely fashion, or if a graceful shutdown is not possible, the application will be abruptly terminated.

4.3. Plugins

A 'plugin' is a special type of custom application. A plugin has a well-defined, single purpose, and its lifetime is limited to a period of time sufficient only for it to perform its required operation. Once the operation has been performed, the plugin instance is terminated. Plugins are developed using the same methods as custom applications, though with some limitations and restrictions which are identified in this document. In order to successfully build a plugin, it must:

1. Be compiled as "type 2" with cabuild.exe (10516C4). See the Custom Application Creator tool help for more details.
2. Make a call to PluginRegisterJobStartNotifier() to register a callback which will be raised when the requesting service (i.e. the service who requested the plugin instance to be created) requires the plugin to start. Note that this implies that the plugin should not start its operation of its own accord, but should wait until requested to do so.

Note that the application type must be observed when loading custom applications and plugins into the configuration tool. It is not possible to load a plugin as a custom application, and vice versa.

Note that due to the architecture of the custom logic system, only a single instance of a particular plugin may run at any one time. This limits the number of concurrent programming jobs which a single plugin instance may service to 1. As such, programming jobs which are handled via a plugin will run sequentially, regardless of the concurrency setting applied in the configuration tool.

4.4. Developing using Linux

For advanced users, it is possible to develop using a Linux environment. An *example* Makefile is provided in the root of the custom application template directory. Note that this is provided as an example only, and should be expanded as necessary by the end user. Note that although development and compilation can occur solely in a Linux environment, the conversion and configuration transfer stage is only possible when using a Windows host.

4.5. Updating the SDK in an existing application

During the course of maintaining a custom application, it is likely that newer versions of the PVSG core software and associated custom application SDK will become available. In such scenarios, it is undesirable to relocate an existing project into a new template. Instead, the SDK should be replaced, per the following instructions.

1. In the root directory of the custom application, remove the 'sdk' directory in its entirety.
2. Untar the new template into a separate directory, and copy the 'sdk' directory into root of the existing application.
3. Carefully inspect the differences between the new and existing versions of the following (Note: Difference determination can be a combination of manual comparison and inspection of the release notes):
   - The project Makefile
   - The NetBeans project files, under the 'nbproject' directory
   - The make_build.bat script
4. If any differences are found, changes from the new template should be merged into the existing application.
5. Fully rebuild the application, and carefully inspect any errors and warnings shown.

4.6. Configuring a build system

Developing custom logic for the PVSG requires a Linux machine to be available which is capable of hosting the build system. Two development methods are available:

1. Develop directly on the Linux machine. This method can be utilised by users who are familiar with the Linux environment, though this is not expected to be the most common development method. See *Developing using Linux* for further details.
2. The Linux machine can be used as a remote build server, with the primary development effort occurring under Windows. This is expected to be the more common development method, and is covered in more detail below.

Note: This guide was written using a system with Ubuntu 16.04.1. Ubuntu is recommended, but not required, as the build server distribution due to its large user-base and accessible support community. Note that if a different distribution is chosen, the instructions below may need to be adapted as required. Whichever distribution is chosen, it is recommended that the latest "long term stable" (LTS) version be used where possible.

The following requirements must be satisfied in order for a Linux machine to be able to build custom applications remotely:

1. A TCP/IP connection must exist between the build server and the development machine(s). Ideally, the build server should be assigned a static IP address.
2. The build server must be configured for SSH and SCP access.
3. The build server must have the Parker PVSG custom application toolchain (10516C5) stored in a known directory.
4. The following build utilities must be available on the build server:
   a. GNU make

SSH and SCP access can be provided on the build server by installing the OpenSSH package:

The toolchain may be extracted to any desired location, but it is recommended to extract it to the following path:

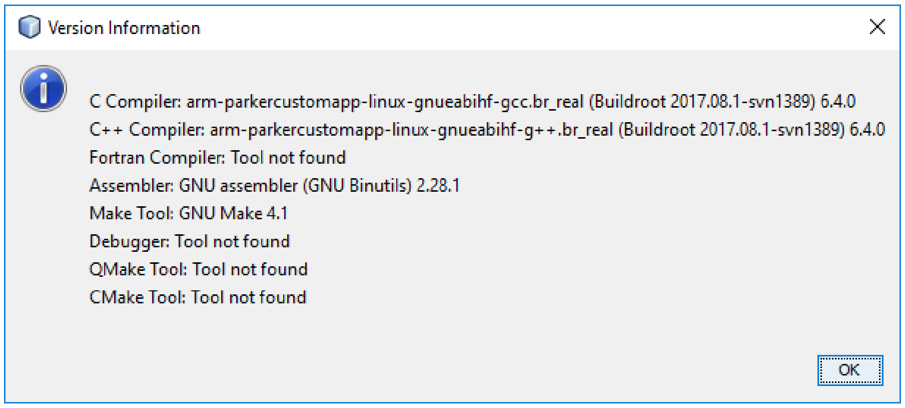
Most of the required build utilities are available in the "build-essential" package:

Note that the availability of the build-essential package may vary by distribution and the package manager in use.

Once the build server is ready, follow the instructions under the heading "Setting up the IDE" in the NetBeans C/C++ remote development tutorial (https://netbeans.org/kb/docs/cnd/remotedev-tutorial.html). Ensure that the build server is set to use "SFTP" to access project files, and that "Automatically find compilers and tools" is unchecked. Next, perform the following (the following instructions assume the toolchain path above was used):

1. Expand the newly added build host under "C/C++ Build Hosts" within the "Services" pane and then expand "Tool Collections" under that. Right-click "GNU" and select "Remove Tool Collection".
2. Right-click "Tool Collections" and select "Add New Tool Collection".
3. Set the Base Directory to "/opt/parker/pvsg-custom-apps-toolchain/bin" and set the Tool Collection Family to "GNU". Set the Tool Collection Name to something like "PVSG", and select OK.
4. Right-click the newly created "PVSG" under "Tool Collections" and select "Properties". Set the following entries (all other entries should be left blank):
   - C Compiler: /opt/parker/pvsg-custom-apps-toolchain/bin/arm-parkercustomapp-linux-gnueabihf-gcc
   - C++ Compiler: /opt/parker/pvsg-custom-apps-toolchain/bin/arm-parkercustomapp-linux-gnueabihf-g++
   - Assembler: /opt/parker/pvsg-custom-apps-toolchain/bin/arm-parkercustomapp-linux-gnueabihf-as
   - Make Command: /usr/bin/make
5. Now, to verify the tool collection has been configured correctly, select "Versions..." in the bottom right-hand corner. After a while, something similar to the window below

should be shown:



```
C Compiler: arm-parkercustomapp-linux-gnueabihf-gcc.br_real (Buildroot 2017.08.1-svn1389) 6.4.0
C++ Compiler: arm-parkercustomapp-linux-gnueabihf-g++.br_real (Buildroot 2017.08.1-svn1389) 6.4.0
Fortran Compiler: Tool not found
Assembler: GNU assembler (GNU Binutils) 2.28.1
Make Tool: GNU Make 4.1
Debugger: Tool not found
QMake Tool: Tool not found
CMake Tool: Tool not found
```

6. Close the versions and properties windows, and open the custom application template project (or an existing custom application project). Right click the project in the "Projects" pane, hover over "Set Build Host", and ensure the proper build server is selected.

When attempting to build a project for the first time, the build may fail with the error "cannot find -lcasdk". This can be resolved by copying "libcasdk.so" from the folder "<project directory>/sdk" on the development machine to the corresponding folder on the build machine (usually found within "<path to home directory>/.netbeans/remote/<development machine IP address>/<development machine local project path>").

Sometimes, the first build will not automatically transfer to the development machine following a successful build. A notification will appear indicating that files have changed on the build server. Select the notification, select the checkbox "Remember my choice", and click OK to automatically download future builds. If this popup doesn't appear, move the build file manually from the server to "(project directory)/output" on the development machine, and try building again.

4.7. Error Codes

The PVSG platform reports custom logic related errors in the system event log and via the dashboard error reporting system. Refer to the software user guide (1051G02) for details on the error codes used and their meanings.

5. Software Release Notes

This section provides software release notes for the distributed PVSG software components referenced by this document.

Each section below provides a summary of the software released that corresponds to a release of this document. For each software item listed the following is provided:

- Part Number: Parker internal software part number
- Software Version: Software version reported by the software
- SDK Version: Internal SDK version. Logic created with one SDK version is compatible with PVSG Embedded Software (1051642) that is the same SDK version.
- Release Case: Parker internal case number associated with a software release

5.1. 1051G05.04A

This revision provided information supporting use of the Parker Vehicle System Gateway (PVSG) with the following software:

Software	Part Number	Software Version	SDK Version	Release Case
Custom Applications Template	10516C2	2.2.62	2	Case 52740
Custom Applications Developer Tools Installer	10516C3	0.22.6.6	2	Case 50122

Summary of changes from 1051G05.03:

- Updated document to reflect current status of the custom logic interface (Case 53309).

5.2. 1051G05.03A

This revision provided information supporting use of the Parker Vehicle System Gateway (PVSG) with the following software:

Software	Part Number	Software Version	SDK Version	Release Case
Custom Applications Template	10516C2	2.2.62	2	Case 52740
Custom Applications Developer Tools Installer	10516C3	0.22.6.6	2	Case 50122

Summary of changes from 1051G05.02:

- Updated makefile of Custom Applications Template (case 52074).

5.3. 1051G05.02A

This revision provided information supporting use of the Parker Vehicle System Gateway (PVSG) with the following software:

Software	Part Number	Software Version	SDK Version	Release Case
Custom Applications Template	10516C2	2.1.56	2	Case 50621
Custom Applications Developer Tools Installer	10516C3	0.22.6.6	2	Case 50122

Summary of changes from 1051G05.01:

- Adjusted J1708 message maximum data size to match product capability (case 50125).
- Added support for asynchronous tasks (case 50153).
- Fixed an issue with the scheduling time on single instance tasks (case 50127).

5.4. 1051G05.01A

This revision provided information supporting use of the Parker Vehicle System Gateway (PVSG) with the following software:

Software	Part Number	Software Version	SDK Version	Release Case
Custom Applications Template	10516C2	2.1.49	2	Case 50122
Custom Applications Developer Tools Installer	10516C3	0.22.6.6	2	Case 50122

Summary of changes from 1051G05.00:

- Added new "Receive All" services for CAN, J1939 and J1708. (Case 49615, Case 49752)
- Added support to send CAN, J1939 and J1708 messages outside the transmit tables. (Case 49614)
- Fixed bug that did not allow user to modify length of J1708 or J1939 messages after initialization. (Case 48984)
- Fixed bug that prevented reading/writing to the last byte of virtual EEPROM. (Case 50036)
- Fixed some timing and cancellation issues with scheduler service. (Case 50035)

5.5. 1051G05.00A

This revision provided information supporting use of the Parker Vehicle System Gateway (PVSG) with the following software:

Software	Part Number	Software Version	SDK Version	Release Case
Custom Applications Template	10516C2	0.21.299.31	2	Case 49259
Custom Applications Developer Tools Installer	10516C3	0.21.4.4	2	Case 49259

- Initial draft release.