# Spark Contribution Guide

Standard guidelines for open-source contributions

March 2018

# Summary

The **Spark** toolkit is a reference implementation of the Common Component Specification (CCS) framework, which was released as version 1 and open-sourced. By default the CCS consists primarily of class interfaces and no functional code, thus the need to illustrate how to build a sample application which utilizes this code. These components are meant to provide features that can be considered “standard” for most web-based applications which operate in a stateless environment.

# Prerequisites

To provide the best experience with the **Progress Application Server (PAS)** and **Progress Developer Studio (PDSOE)** it is recommended that you be on the latest service pack of OpenEdge. The source provided with Spark should be compatible with both OpenEdge 11.6 and 11.7, with the latter being preferable due to significant simplifications of security options and improved support for Single Sign-On and OAuth. Use of a **64-bit Windows** installation and **OE 11.7.2 or later** is assumed, and at least the Progress Developer Studio for OE component present.

Access to the repository is provided through **Git** and you may benefit from having a suitable Git client installed on your workstation. As a suggestion, **Git for Windows** and **TortoiseGit** will provide a seamless integration with Window Explorer. Some automated tasks will be performed using **Ant** which will already be present in your DLC directory if using 11.7 or later.

# Obtaining Code

1. Create a directory **C:\Modernization** for all future application code.
2. **Option 1:** Right-click within the new directory to view TortoiseGit options.
   1. Perform a “**Git Clone**” of [**https://github.com/progress/Spark-Server**](https://github.com/progress/Spark-Server)
3. **Option 2:** Visit <https://github.com/progress/Spark-Server/releases> and download the latest available release as either a .ZIP or .TAR.GZ archive (~90MB).
   1. Expand the archive, making sure the top-level directory is named simply “**Spark-Server**” and contains an immediate “**src**” folder within.
4. Confirm the source code is available by locating the “**src**” directory contents.
   1. This directory can be imported into PDSOE as an OpenEdge project.

# Workspace Options

Before proceeding, it may be useful to enable some options within the Progress Developer Studio for OpenEdge (Eclipse) environment. These options will provide a more consistent experience with the actions to be requested in the remainder of this document. Begin by starting the **Developer Studio** and selecting **C:\Modernization** as your workspace location. If PDSOE has already been started under a different workspace, use the option **File > Switch Workspace > Other…** to choose.

Window -> Preferences

General

Show heap status: checked

Editors

Text Editors

Insert spaces for tabs: checked

Displayed Tab Width: 4

Show line numbers: checked

Search

Reuse editors: unchecked

Workspace

Refresh using native hooks or polling: checked

Refresh on access: checked

Workspace name: "Your Workspace Name Here"

LocalHistory

Maximum entries per file: 1

Progress OpenEdge

Editor

Case: Lower

Expand keywords: checked

Case keywords: checked

Build -> Automatically syntax check: checked

Server

Remove all files and folders when cleaning server pub dir: checked

Update properties from server before starting/launching: checked



Project Explorer -> View Menu (small icon on panel, shown above)

Customize View

Select the filters to apply (matching items will be hidden)

Unselect \*.pl to view Procedure Library files.

# Contribution Types

For purposes of establishing proper guidelines, it would be helpful to divide the most common types of contributions into predefined buckets. We’ll use these to identify use-cases and procedures for submitting changes back to the project.

**Bugfixes** – Corrections to code that has been found to contain a legitimate defect or issue that could corrupt data or cause an unexpected error. Fixes to code should not significantly alter the behavior of the feature it corrects.

**Feature Enhancements** – Additions of a new behavior to an existing feature, or makes it possible to use an existing feature in a new way at a future time. For example, adding a new class interface method, or a method override in an existing class.

**New Features** – Entirely new functionality which may or may not be based on a CCS component, and somewhat generic in nature. In other words, new features should extend the toolkit in a way that is not specific to any proprietary business case.

# General Guidelines

Regardless of the contribution type, the following should be considered when developing a solution and providing content to be accepted back into the codebase. Always consider that reviewers may included volunteers or those who are new to the Spark toolkit, and as such need as much assistance as possible through effective documentation. Please consider including the following pieces of information:

**Business Case** – Always provide a clear and consice reason for any changes. With regard to the contribution types listed above, explain the changes that are being made and the overall purpose for the changes. Creation of diagrams and flowcharts are welcomed and encouraged as well.

**Changelog** – Document the artifacts added, removed, or modified as part of the solution being contributed. Remember to include any configuration files, special server settings, and intended schema to support your solution.

**Test Case** – In order to allow others to replicate the results of any changes, provide instructions for executing the code being provided. In the case of a bugfix, listing the original result of the “bad” code is helpful in illustrating how the change affects output. For new or updated features, a combination of a good business case and test case will help tremendously in understanding the intended behavior.

**Unit Tests** – Inclusion of an ABLUnit or similar test script would be helpful for automation or continuous integration. If possible and necessary, instructions for load testing (via JMeter or similar) would also be encouraged.

# Directory Structures

Within the immediate repository structure are the following support directories:

**cfg** – Contains sample configuration files that are expected to be deployed with a Spark installation in a PAS instance. By default these would be placed within the CATALINA\_BASE\conf\spark folder.

**dist** – Contains the pre-built Ccs.pl and Spark.pl files. When performing a build of the framework, generated output will overwrite these files.

**docs** – Location of generated class documentation, as output by the Progress Compile Tools. Two forms of documentation exist as ABLDuck and PCTDoc, which should be identical in content but differ slightly in presentation. Any additional or relevant documentation for the toolkit overall should be placed here as well.

**src** – The true source code for Spark, as described in the next section.

There are directories within the “src” folder for specific assets and purposes:

**build/preprocess/doctemp** – These are temporary directories that may be created as part of a build process. They contain .r code or transitional artifacts which should not be checked in as part of the repository.

**Ccs** – A copy of the CCSv1 source code, useful for compiling the Spark library and/or viewing the class interfaces. For reference, as of 11.72 a copy of this code has been made part of the OpenEdge installation.

**Spark** – All source code for the Spark toolkit itself. The directory structure here should match the class path as seen in code which utilizes the toolkit.

**tests** – Classes and procedures to aid in creating ABLUnit tests.

**tools** – General catch-all for scripts which may help test certain features or alter an environment quickly for use with the Spark toolkit. Contains a copy of the Progress Compile Tools for use with build processes.

# Build Process

To create a new release of the Spark toolkit or to create updated documenation, an Ant task may be used which utilizes the **build.xml** in the “src” directory. Within the current build script are 4 targets: **build\_ablduck, build\_pctdoc, build\_ccs\_pl,** and **build\_spark\_pl**. All targets take the same -Dversion parameter which should be the current version of the framework being created. For instance, as of March 2018 the latest version number used is 4.2.1 and will be reflected in any generated content. Output will be sent to either the root “dist” or “docs” folder, depending on whether code artifacts or documentation are being created.