Satellite Control Software (SCS)

Mission Data Client Extensibility User Guide

Prepared by:

Florian George

Stéphane Billeter

swiss space center

Space Center EPFL

Lausanne

Switzerland

06 November 2013

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

RECORD OF REVISIONS

ISS/REV	Date	Modifications	Created/modified by
1/0	03/08/2011	Initial release	Florian George
2/0	06/11/2013	Updated for QB50. Release ICD for CDR.	Stéphane Billeter / Muriel Richard

RECORD OF REVISIONS **2**
ABBREVIATED TERMS **4**
INTRODUCTION **5**
1 IACTIONMODULE MODULES **7**
2 WINDOW AND FRAMEWORKELEMENT MODULES **8**
3 ACCESSING LIVE DISTRIBUTION **9**
4 ACCESSING ARCHIVE DISTRIBUTION **11**
5 CONFIGURING THE MDC TO LOAD MODULES **13**
6 CUSTOM REPORTING DATA **14**
7 FIGURES AND TABLES **15**
8 REFERENCES **16**

ABBREVIATED TERMS

Ack	Acknowledgement
CCSDS	Consultative Committee for Space Data Systems
CRD	Custom Reporting Data
COM	Component Object Model
ECSS	European Cooperation for Space Standardization
EGSE	Electrical Ground-Support Equipment
ESA	European Space Agency
MCS	Mission Control System
MDC	Mission Data Client
MDR	Mission Data Repository
Nack	Negative Acknowledgment
OSI	Open Systems Interconnection
PUS	Packet Utilisation Standard (see [N1])
SCOE	Special Checkout Equipment
TC	Telecommand
TM	Telemetry
VC	Virtual Channel
WPF	Windows Presentation Foundation

INTRODUCTION

This document describes the extensibility infrastructure of the Mission Data Client (MDC). The MDC can load modules which can perform a multitude of tasks ranging from new data displays to launching external programs. Figure 1 shows the simplified data flow of the built-in and custom reporting data. It is first processed and distributed by the Mission Control System (MCS), custom reporting data is serialized by the distribution modules. Then the data is distributed to the Mission Data Client where custom reporting data is deserialized. Processing, distribution and client modules are therefore usually developed together.

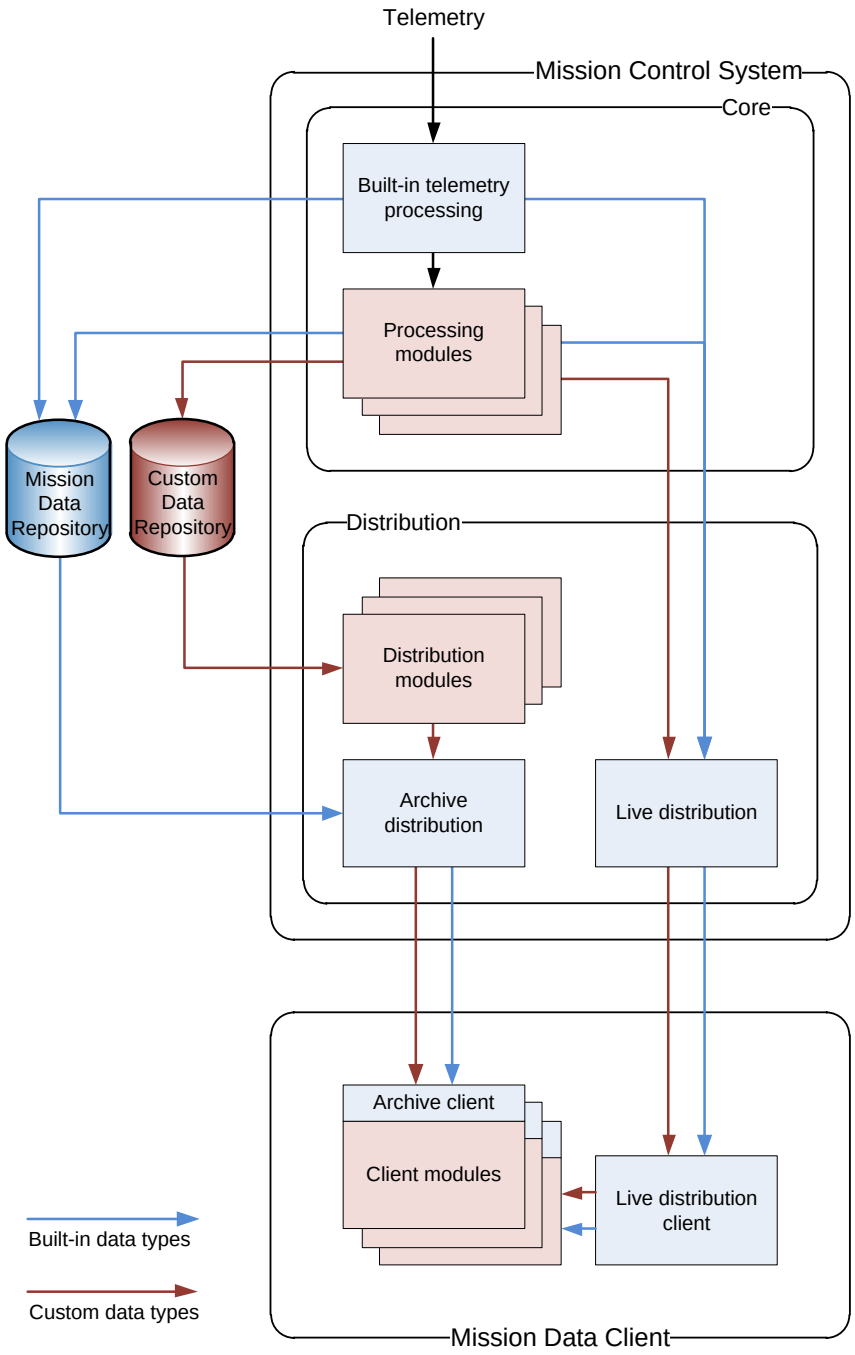

Figure 1 - The MCS and MCS extensibility architecture

The MDC is entirely built using the .NET Framework[1], modules are no exception. A module is a class implementing a specific interface and configured to be loaded by the MDC. Multiple modules can be present in a single assembly. Chapter 6 describes the modules in details.

The modules list is built at MDC's start-up and the modules are only instantiated when launched by the user.

Modules must be a class that follows at least one the following requirements:

- Implement the *IActionModule* interface[2].
- Derive from the *Window* class from WPF.
- Derive from the *FrameworkElement* class from WPF.

[1] .NET Framework: http://www.microsoft.com/net/

[2] *Epfl.SwissCube.GroundSegment.MissionDataClient.Components.IActionModule* in *MissionDataClient.Components.dll*

1 IACTIONMODULE MODULES

Module can implement the *IActionModule* interface[3] shown in Figure 2. If they do, the MDC will call the *PerformAction* method after the module instantiation. The argument passed with this method is the string specified for the *argument* attribute in the MDC's configuration (see chapter 5 *Configuring the MDC to load modules*).

```
namespace Epfl.SwissCube.GroundSegment.MissionDataClient.Components
{
	public interface IActionModule
	{
		void PerformAction(string argument);
	}
}
```

Figure 2 - IActionModule interface

[3] *Epfl.SwissCube.GroundSegment.MissionDataClient.Components.IActionModule* in *MissionDataClient.Components.dll*

2 WINDOW AND FRAMEWORKELEMENT MODULES

If the module's class is derived from the *FrameworkElement* class (*System.Windows.FrameworkElement* in the *PresentationFramework.dll* assembly of WPF), it is instantiated and added as the root element of the newly created module window. Module windows extend normal bare windows with the addition of a zoom slider which allows the user to increase the size of the window content as well as a spacecraft selection dropdown list.

If the module's class is derived from the *Window* class (*System.Windows.Window* in the *PresentationFramework.dll* assembly of WPF), it is used directly as the window for the module (therefore without the added functionalities of a module window).

The module's life-cycle (whether it is considered as running or not) is directly tied to the life-cycle of the module's window itself (whether it is a provided window or not).

3 ACCESSING LIVE DISTRIBUTION

Window and *FrameworkElement* modules can access the live distribution to receive the incoming mission data of their choice. To do that, the modules can retrieved an instance of the *LiveDistributionClient* class[4] from the application resources through which all the interaction with the live distribution can be performed. Figure 3 shows how to retrieve this instance and how to register for the reception of housekeeping values.

```
liveDistributionClient = (LiveDistributionClient)FindResource("LiveDistributionClient");
liveDistributionClient.HousekeepingValueReceived += ...
```

Figure 3 - Retrieving the *LiveDistributionClient* instance example

Figure 4 shows the definition of the *LiveDistributionClient* class. Modules can register to the various **Received* events to receive the needed mission data. Modules shall only register to the events corresponding to the data they need. Indeed, if no module requests a specific mission data category, the data is not transferred from the distribution server to the MCD to reduce the network traffic. Unnecessarily registering for mission data would prevent this.

[4] *Epfl.SwissCube.GroundSegment.Mcs.Distribution.LiveDistributionClient* in *Mcs.Distribution.Client.dll*

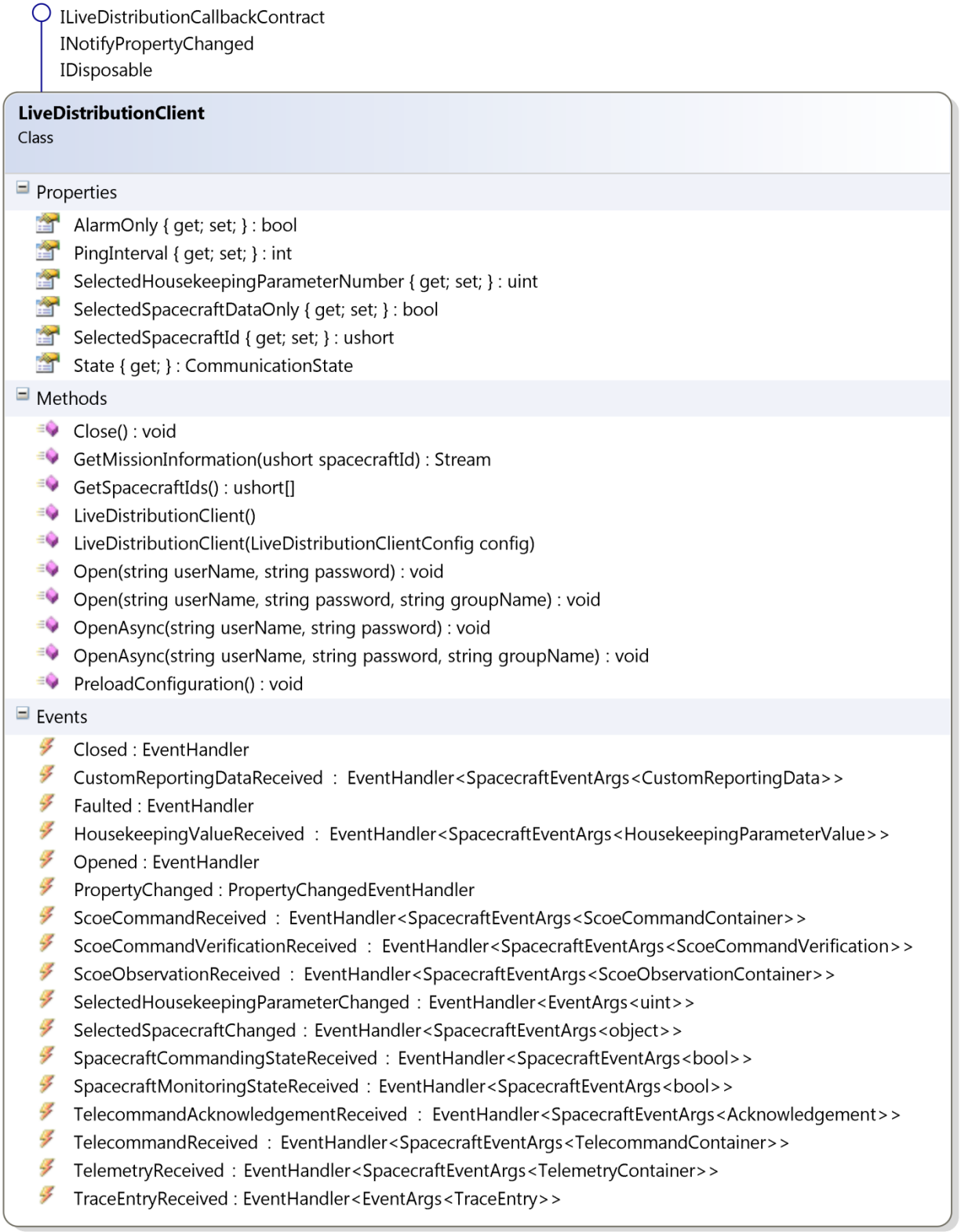

Figure 4 - LiveDistributionClient class

4 ACCESSING ARCHIVE DISTRIBUTION

The modules can access the archive distribution to access all the mission data. To do that, the modules can create an instance of the *ArchiveDistributionClient*[5] class. Figure 5 shows an example.

```
using(ArchiveDistributionClient client = new ArchiveDistributionClient())
{
	var values = client.Channel.GetHousekeepingParametersLastValue(spacecraftId);
	// ...
}
```

Figure 5 - Accessing the archive distribution

Figure 6 shows the definition of the *IArchiveDistribution* and the archive data distribution interfaces implemented by the *ArchiveDistributionClient*'s channel.

[5] *Epfl.SwissCube.GroundSegment.Mcs.Distribution.ArchiveDistributionClient* in *Mcs.Distribution.Client.dll*

Figure 6 - IArchiveDistribution and archived data distribution interfaces

5 CONFIGURING THE MDC TO LOAD MODULES

To instruct the MDC to load a module, its configuration shall be updated. The configuration file is the XML file named *Mission Data Client.exe.config* and located in the installation directory of the MDC. An XML element matching the following format must be added to the *clientModules* section:

```
<add name="unique-name" title="Displayed Title" isLive="true|false"
     type="assembly-qualified-type-name" argument="argument to IActionModule" />
```

Attributes description:

name	Unique name given to the module. Each module must have a unique name. As the name can be used for example to automatically launch a module at MDC's start-up, it is recommended not to use spaces or any special characters in its name.
title	Displayed title for the module in the MDC's main window and in the title of the module's window (if applicable).
isLive	Boolean value (*true* or *false*) that indicate in which category the module must be placed in the MDC's main window.
type	Assembly-qualified name[6] of the module's type (implementing *IActionModule* interface and/or deriving from *FrameworkElement*).
argument	Argument string passed to the *IActionModule.PerformAction* method when the module implement the *IActionModule* interface (see chapter 1 *IActionModule*).

Here is an example of the *clientModules* section configured to load two modules:

```
<clientModules>
  <add name="ImageService" title="Image Service" isLive="true"
       type="SwissCube.ImageService.ImageServiceViewer, ImageService.ClientModule" />
  <add name="launchNotepad" title="Launch Notepad" type="ProgramLauncherModule"
       argument="notepad" />
</clientModules>
```

Figure 7 - Example of *clientModules* configuration section with two client modules

[6] .NET Framework assembly-qualified type names:
http://msdn.microsoft.com/en-us/library/system.type.assemblyqualifiedname.aspx

6 CUSTOM REPORTING DATA

Processing modules can generate data for a non-built onboard service, for mission-specific onboard services or even data specific to the module itself. This data can be stored for archiving and/or sent to clients through the Live Distribution. In complement, distribution modules retrieve this archived data on demand to make it available to clients. This data is called Custom Reporting Data (CRD) in the MCS.

CRD is always exposed and transferred in the form of instances of the *CustomReportingData* class[7]. This class, as shown in Figure 8, is a simple container pairing the data and the data type and subtype.

The data must be binary-serializeable (convertible to a byte array). Two static methods to serialize and deserialize data are present to provide a default binary serialization mechanism based on the .NET Framework's *DataContractSerializer*[8]. Any other reversible [on the client-side] mechanism resulting in a byte array can be used instead.

A data type is similar to a service type of a PUS services in [N1]. For consistency, it is recommended that the data type used by the processing and distribution modules is the same as the PUS service generating the data. For example, if the spacecraft implement a mission-specific service type 128 that generate images, the modules would expose the images and associated data as data type 128.
Subtypes can be freely used for the different subtypes of data generated/stored. For example, in the case the service generating images, there could a subtype for the images themselves, for the metadata, for a summary of the transmitted images, etc.

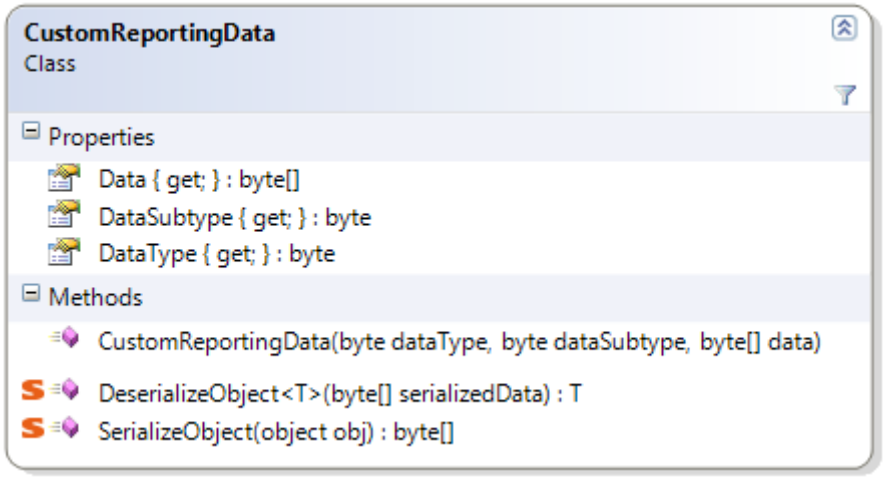

Figure 8 - *CustomReportingData* class

CRD can be received from the live distribution through the *CustomReportingDataReceived* event of the *LiveDistributionClient* instance and through the *GetCustomReportingData* method of the *ArchiveDistributionClient* 's channel.

---

[7] *Epfl.SwissCube.GroundSegment.Mcs.Distribution.CustomReportingData* in *Mcs.Data.Downlink.dll*

[8] DataContractSerializer: http://msdn.microsoft.com/en-us/library/ms405768.aspx

7 FIGURES AND TABLES

Figure 1 - The MCS and MCS extensibility architecture 5
Figure 2 - IActionModule interface 7
Figure 3 - Retrieving the *LiveDistributionClient* instance example 9
Figure 4 - LiveDistributionClient class 10
Figure 5 - Accessing the archive distribution 11
Figure 6 - IArchiveDistribution and archived data distribution interfaces 12
Figure 7 - Example of *clientModules* configuration section with two client modules 13
Figure 8 - *CustomReportingData* class 14

8 REFERENCES

[N1] ECSS-E-70-41A Ground systems and operations - Telemetry and telecommand packet utilization. 30 January 2003.

[N2] SC-ICD-1-0-MCS_Extensibility – Mission Control System Extensibility User Guide