Startup Manager Specification

OpenEdge Framework Technical Design Document

OpenEdge Application Architecture Specification (OEAA)

Version 0.3

Version	1.0		
Spec Team Members	Sunil Belgaonkar	sbelgaon@progress.com	Progress
	Shelley B. Chase	schase@progress.com	Progress
	Marian Edu	marian.edu@acorn.ro	Acorn IT
	Rom Elwell	rome@issol.com	Innovative Software Solutions
	Mike Fechner	mike.fechner@consultingwerk.de	Consultingwerk
	Ganesh Iyer	gaiyer@progress.com	Progress
	Peter Judge *	pjudge@progress.com	Progress
	Tom Kincaid	tkincaid@progress.com	Progress
	Christopher Longo	chlongo@progress.com	Progress
	Paul Moberg	paulmoberg@quickenloans.com	Quicken Loans
	Mark Opfer	mopfer@dmsi.com	DMSi
	Simon L. Prinsloo*	simon@vidisolve.com	Vidisolve
	Phani Sajja	psajja@progress.com	Progress
	Kevin Schantz	kws@qad.com	QAD
	Robin Smith*	rosmith@progress.com	Progress
	* *Denotes contributing author*		

TABLE OF CONTENTS

1 Introduction

The Common Component Specification (CCS) project is designed to simplify the development of modern business applications by defining the architecture and components of a modern application development framework for the OpenEdge platform. The CCS project identifies a set of specifications for the common components needed in developing modern business applications. When these components are built as part of a modernization framework, application developers can concentrate more on the business logic of the application rather than on infrastructure and integration.

The *Startup Manager Specification OpenEdge Application Architecture Specification (OEAA) version 0.3* builds on the *CCS Specification: OpenEdge Application Architecture* and defines the Startup Manager API as Object-Oriented ABL.

The class and interfaces described in the document will form part of the OpenEdge Application Architecture (OEAA) and provides a framework against which any consumer of a CCS compliant framework or component can code, regardless of the choice of the framework implementation. Implementers of a CCS compliant Startup Manager must implement and use these components as described in this document.

Classes that form building blocks of a framework deliver standard behaviour, features and functionality independent of any specific application requirements. Such a framework should supply a stable and reliable set of services for the entire lifetime of an ABL session. The *CCS Specification* recognizes the fact that a session may need a bootstrap process in order to get a session to this stable state by initializing some or all of the framework services. This process needs to be flexible enough to allow the creation of custom architectures by combining services from various frameworks and it also need to be robust enough to cope with the inherent unstable state of the framework environment during this process. Booting a framework to a stable state is the responsibility of the **Startup Manager**. (Chase, et al., 2016, p. 9)

1.1 Purpose

The purpose of this technical specification is to describe the API definitions, the expected behaviour and other collateral needed to use (or implement) a CCS compliant Startup Manager.

It will enable consumers of any CCS compliant Startup Manager to understand the services, behaviour and requirements of a Startup Manager. Implementers of a CCS compliant Startup Manager must use this as document as minimum requirements to implement for the expected components.

1.2 Scope

The *CCS Specification: OpenEdge Application Architecture Version 1* defines the components of an OEAA-compliant architecture, the responsibilities of each component, and the communication protocol between components.

This specification defines the API, expected behaviour and other collateral needed to successfully use (or implement) a CCS compliant Startup Manager.

Any references to and examples of common components as defined in *CCS Specification: OpenEdge Application Architecture Version 1*, other than the Startup Manager itself, will be for illustrative purposes only and will not form part of this specification beyond the minimum requirements imposed on those components in order to accommodate their safe initialization by the Startup Manager.

As the managers are specialized services, the *Service Manager Specification: Technical design document* (Judge, 2016) will take precedence, should this specification ever be in conflict with it.

1.3 Definitions, Acronyms and Abbreviations

Business Service – Services specific to the purpose of the application.

CCS – Common Component Specification. *(Chase, et al., 2016)*

OEAA – OpenEdge Application Architecture. *(Chase, et al., 2016)*

OERA – OpenEdge Reference Architecture. *(Ormerod, 2006)*

Manager – Service specific to the Common Infrastructure. *(Chase, et al., 2016)*

Service – Self-contained unit of functionality

SOA – Service oriented architecture

1.4 Contents Overview

Section 1 is the introduction and includes a description of the project, applicable and reference documents.

Section 2 provides the component's overview.

Section 3 contains the context in which the component will be applied.

Section 4 describes the component's design method, standards and conventions.

Section 5 contains the component's component descriptions.

Section 6 includes the component's revision history, outstanding issues, and action items

2 Component Overview

2.1 Component Description

The Startup Manager is classified as a Required Component of the OEAA. *(Chase, et al., 2016, p. 8)*

The Startup Manager is a Common Infrastructure Service of an application. As such, it can potentially be used in all layers of the OpenEdge Reference Architecture (OERA).

“Common Infrastructure services are non-domain specific related functions that provide the common infrastructure support for a modern application. They comprise of standard behaviour, features and functionality independent of any specific application requirements.” (Ormerod, 2006, p. 3)

According to Ormerod *(2006, p. 3)*, Common Infrastructure services, or the so-called Managers in CCS terminology, are expected to be started with the ABL session and will therefore always be available.

The *CCS Specification: OpenEdge Application Architecture Version 1* requires the following functionality of the Startup Manager: *(Chase, et al., 2016, p. 9)*

The Startup Manager MUST

- Prepare a set (zero or more) of Manager components, that are in a predefined order. The names and order of these Manager components will be provided by a set of configuration data. The choice of the configuration data location is the responsibility of the Startup Manager implementer and will be coded by an implementer. These data may come from one or more of the following.
    - a CCS component responsible for configuration (to be spec’d);
    - a hard-coded list;
    - a known file on disk;
    - any other source of configuration data (an HTTP connection, a database etc)
- Start and initialise that set of Manager components in order. Initialisation is usually done via a call to an Initialize() method.
- Provide a reference to the started Managers via the getManager() method
- Set the values of relevant Managers on the Ccs.Common.Application class.
- The Startup Manager MAY use the Service Manager to start Manager components but this is NOT required.
- The Startup Manager SHOULD recognise that there may be dependencies that must be manually resolved in the Startup Manager.
  During the boot process, a system is not in a well-known, pre-defined state and there is no guarantee that any individual Manager component is available at any given point in time. Managers from different implementations may have different dependencies.

2.1.1 Functional overlap with Service Manager

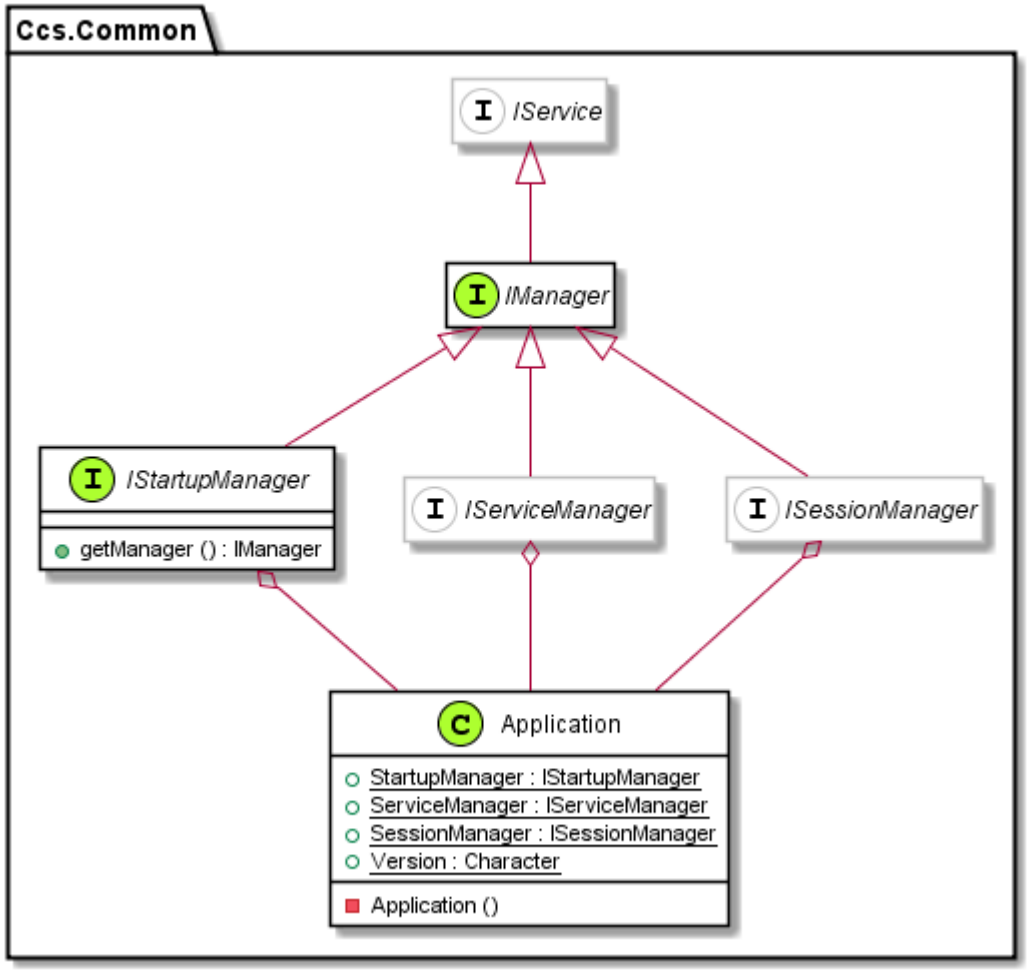
The Architecture spec states that

> *The Service Manager is responsible for instantiating all Business Services and managing their life cycle. The Service Manager is used to instantiate these objects or procedures and shut them down as appropriate based on a life cycle configuration of the service. The Service Manager is the central controller or factory that ensures that Business Services are initialized consistently and not left consuming resources unnecessarily or started multiple times. (Chase, et al., 2016, p. 13)*

There is some overlap between this and the Startup Manager's responsibility in running Manager components, since Manager components are specialised Services. It is the choice of the implementer whether the Startup Manager uses the Service Manager to instantiate a Manager if it chooses to.

A Service Manager implementation MUST take into account the fact that some components are Managers and were started by the Startup Manager. It MUST NOT attempt to instantiate/run these without reference to the Startup Manager, via the getManager() method and/or the Ccs.Common.Application class. The authors recognise that this requirement cannot be enforced in an interface and that the onus for this falls on the Service Manager implementers.

2.2 Component Architecture



2.2.1 Ccs.Common.IService

StartupManager implements the Ccs.Common.IService interface, as specified in *Service Manager Specification: Technical design document,* (Judge, 2016).

2.2.2 Ccs.Common.IManager

This interface inherits from *Ccs.Common.IService*, but does not currently contribute any new members.

It serves as a marker to indicate that a class is a common infrastructure service that can be started and managed by the Startup Manager. Classes implementing this interface are started by the Startup manager and, once session initialization has completed, are always available, as described by the OERA.

The Startup Manager is the sole authoritative factory and repository of services implementing this interface.

Both the framework builder and the application programmer can add custom extensions to the Common Infrastructure by implementing this interface and configuring the Startup Manager to make it available.

2.2.3 Ccs.Common.IStartupManager

The CCS requires a pluggable model that allows for components from different vendors to co-exist. *(Chase, et al., 2016, p. 9)*

This requires that the Startup Manager be implemented as a factory for all common components.

It needs to use configuration data to identify and start the managers in a pre-defined sequence in order to accommodate dependencies where possible.

As managers from different frameworks can be plugged into the same session, dependencies between managers may vary. During the start-up phase, framework implementers that rely on such dependencies may risk incompatibility with certain configurations involving a mix of managers from different vendors.

Since the Startup Manager itself must be a pluggable component, the bootstrap procedure will need to inject the Startup Manager of choice into the framework. The implementation detail of the selected Startup Manager will determine if the procedure should also invoke an initialization method and if it must assist with obtaining the Startup Manager's configuration or not, as deciding how and where to find the configuration data is an implementation detail.

This interface defines the following method:

- *getManager ()* return an IManager interface of the requested manager, if it is configured.

2.2.4 Ccs.Common.Application

The injection requirement of the Startup Manager means that it cannot follow any traditional singleton pattern. An alternative, well-known method is needed to locate it, which in turn will grant access to all other managers.

This will be achieved by the CCS itself providing the *Ccs.Common.Application* class.

The class has a private constructor and cannot be instantiated.

The class provides the following static properties that need to be populated during the bootstrap process:

- StartupManager: Ccs.Common.IStartupManager
- ServiceManager : Ccs.Common.IServiceManager
- SessionManager : Ccs.Common.ISessionManager

2.3 Component Package Definition

The Startup Manager class and interface will be a subset of the *Ccs.Common* namespace, which will also contain all other interfaces referenced by the Startup manager's properties.

2.4 Component Property Data and Organization

The requirements of the *CCS Specification: OpenEdge Application Architecture Version 1* in terms of the naming conventions, storage organization and interaction between components applies to the components defined in this specification.

2.5 Component Run-time Characteristics

The session Startup procedure will inject the Startup Manager of choice by creating it and calling the *initialize()* method and, if required by the particular implementation, supply the configuration data or assist with obtaining it.

The constructor of the Startup Manager:

1. Should not perform any other work, but rather defer it to the *initialize()* method.

The i*nitialize()* method of the Startup Manager:

1. May locate the Startup Manager configuration data or may require that it be supplied as a parameter to an implementation specific overload.
2. Must either ensure that its configuration data provides proper implementations of IServiceManager and ISessionManager or alternatively supply default values for these when they are not configured.
3. Use the configuration data to locate and instantiate all the configured managers in the correct sequence.
4. Once the manager has been instantiated, call the initialize() method on the manager.

In general, managers should be singletons, loaded during bootstrap, and be available at all time. (Chase, et al., 2016, p. 8) (Ormerod, 2006, p. 3)

Although the practice is discouraged, the Startup Manager must provide for situations where an implementer chose to implement more than one manager's interface in the same object and not waist resource by launching an instance for each interface. For example, if a particular object implements both IMessagingManager and ILoggingManager, the reference of first one started during initialization must be re-used by the second one.

The *getManager()* method will take a Progress.Lang.Class instance as input parameter. The class must represent a type that is:

- An interface – IsInterface() = TRUE

- Derived from Ccs.Common.IManager – IsA(Ccs.Common.IManager) = TRUE

The method will then locate the definition of this type in its configuration. If it is defined and launched, it will return the instance of the implementing type. In all other cases, an error must be thrown.

2.6 Component Error Handling

It is preferred that a fail fast philosophy be followed during initialization and that errors are left to bubble up through the call stack to the session startup routine rather than being intercepted.

Errors must have messages. Thus when the default constructor or the constructor taking only a string is used to instantiate an AppError, it must be followed with a call to the AddMessage() method before the error is thrown.

Component Interaction with External Sub-systems

The Startup Manager act as the factory of the following common component services and serves as the authoritative provider of references to any object implementing the IManager interface, which includes, but is not limited to, the following objects:

- Ccs.Common.IServiceManager
- Ccs.Common.ISessionManager
- Ccs.Common.IContextManager
- Ccs.Common.IAnalyticsManager
- Ccs.Common.IAuthenticationManager
- Ccs.Common.IAuthorizationManager
- Ccs.Common.IConnectionManager
- Ccs.Common.ILoggingManager
- Ccs.Common.IMessagingManager
- Ccs.Common.IPropertyManager
- Ccs.Common.ITranslationManager

3 Component Interfaces and Classes

3.1 Ccs.Common.IManager

Inherits Ccs.Common.IService.

Flags a class as an OERA Common Infrastructure service, to be managed by the StartupManager.

```
USING Ccs.Common.IService FROM PROPATH.

INTERFACE Ccs.Common.IManager INHERITS IService:

END INTERFACE.
```

3.2 Ccs.Common.IStartupManager

Session bootstrap and factory of common components.

```
USING Ccs.Common.IManager FROM PROPATH.
USING Ccs.Common.IServiceManager FROM PROPATH.
USING Ccs.Common.ISessionManager FROM PROPATH.

INTERFACE Ccs.Common.IStartupManager INHERITS IManager:

  /*------------------------------------------------------------------------------
  Purpose: Retrieve an instance of the specified IManager object.
  Notes: If a manager is not configured, no error should be raised, but if it is
         configured and fails to load, an exception must be raised.
  @param pServiceType The Progress.Lang.Class repersenting the required service.
  @return IManager implementation of the requested type, or ? if its not configured.
  ------------------------------------------------------------------------------*/
  METHOD PUBLIC IManager getManager ( pServiceType AS Progress.Lang.Class ).

END INTERFACE.
```

3.2.1 Public Instance methods

Name	initialize() *Inherited from Ccs.Common.IService*
Description	Initialize the StartupManager
Return Type	Void
Parameters	None
Exceptions	Progress.Lang.SysError Progress.Lang.AppError

Name	getManager()

Description	Locate the instance of the injected type representing a given common infrastructure service.		
Return Type	Ccs.Common.IManager or Unknown		
Parameters	Input	Progress.Lang.Class	Interface type for which the implementation is desired.
Exceptions	Progress.Lang.AppError		

3.3 Ccs.Common.Application

Provides a well-known point to find references to specific CCS Manager components.

```
BLOCK-LEVEL ON ERROR UNDO, THROW.

USING Ccs.Common.IServiceManager FROM PROPATH.
USING Ccs.Common.ISessionManager FROM PROPATH.
USING Ccs.Common.IStartupManager FROM PROPATH.

CLASS Ccs.Common.Application FINAL:

  /*------------------------------------------------------------------------------
   Purpose: Provides access to the injected IStartupManager.
   Notes:
  ------------------------------------------------------------------------------*/
  DEFINE STATIC PUBLIC PROPERTY StartupManager AS IStartupManager NO-UNDO GET. SET.

  /*------------------------------------------------------------------------------
   Purpose: Provides access to the injected IServiceManager.
   Notes:
  ------------------------------------------------------------------------------*/
  DEFINE STATIC PUBLIC PROPERTY ServiceManager AS IServiceManager NO-UNDO GET. SET.

  /*------------------------------------------------------------------------------
   Purpose: Provides access to the injected ISessionManager.
   Notes:
  ------------------------------------------------------------------------------*/
  DEFINE STATIC PUBLIC PROPERTY SessionManager AS ISessionManager NO-UNDO GET. SET.

  /*------------------------------------------------------------------------------
   Purpose: Version of the Common Component Specification implementation.
   Notes:
  ------------------------------------------------------------------------------*/
  DEFINE STATIC PUBLIC PROPERTY Version AS CHARACTER NO-UNDO
    INITIAL '1.0.0':u
    GET.

  /*------------------------------------------------------------------------------
   Purpose: Prevent creation of instances.
   Notes:
  ------------------------------------------------------------------------------*/
  CONSTRUCTOR PRIVATE Application ():
    SUPER ().
  END CONSTRUCTOR.

END CLASS.
```

3.3.1 Public Constructors

	Default constructor
Description	This constructor is private in order to prevent the creation of an instance.

3.3.2 Public static properties

Name	StartupManager
Description	This property is the session wide gateway that is needed to get hold of the injected Ccs.Common.IServiceManager instance, which in turn, gives access to the instances of all other injected Mangers.
Type	Ccs.Common.IStartupManager
Getter	Public
Setter	Public

Name	ServiceManager
Description	This property is the session wide gateway that is needed to get hold of the injected Ccs.Common.IServiceManager instance, which in turn, gives access to the instances of all other injected Mangers.
Type	Ccs.Common.IServiceManager
Getter	Public
Setter	Public

Name	SessionManager
Description	This property is the session wide gateway that is needed to get hold of the injected Ccs.Common.ISessionManager instance, which in turn, gives access to the instances of all other injected Mangers.
Type	Ccs.Common.ISessionManager
Getter	Public
Setter	Public

4 Guidelines for Framework Creators

It is the purpose of the StartupManager to manage the boot process by injecting the desired managers in a given sequence. During instantiation, manager should not rely on any other manager, but if it does, it must access it via the Startup Manager, as the Service Manager will not be guaranteed to exist.

Keep in mind that the system is unstable during boot up, meaning that the exact time that a manager becomes available is not known. Managers should be sensitive for such conditions.

Constructors can only succeed or throw fatal errors without returning a reference. It is however possible to encounter non-fatal errors during initialization that needs to be communicated. For instance, a logging manager may not have sufficient access to write to the specified log file or directory and consequently write to a file with an arbitrary name in the session's temp directory instead.

As such, it is advised that Constructors should do as little work as possible, if any, and defer initialization to the *initialize()* method. This way the constructor can return a valid reference while initialization can still throw a non-fatal error.

It is preferred that a fail fast philosophy be followed during initialization and that errors are left to bubble up through the call stack to the session startup routine rather than being intercepted.

However, if the implementer chooses to catch errors during the construction or initialization of a configured Manager, care must be taken to preserve the original error detail. It must not blindly rely on the availability of other services like a message, logging or translation manager. If it tries to use such a service and encounter a problem, the original error must be thrown or it must be preserved as an inner error on any newly thrown error.

5 Guidelines for Application Developers

In the case of optional managers, the application or implementation's rules, rather the infrastructure's rules, determine if the problem is fatal or not. For example, in one case an application may simply continue using the default language if no translation manager is available, while another may consider this a fatal problem.

The startup manager will always return a valid reference to the requested common infrastructure manager or throw an exception if the manager is not available. This enables developers to use standard error handling techniques to handle non-fatal cases while relieving the developer from the burden to test for valid references everywhere.

Sample code of how the StartupManger Instance may be used to retrieve a common infrastructure manager:

```
DEFINE VARIABLE oLogManager AS ILoggingManager NO-UNDO.

oLogManager = CAST(Ccs.Common.Application:StartupManager:getManager(
                                        GET-CLASS(ILoggingManager)), ILoggingManager).
oLogManager:logMessage("My log message").
```

Sample code of how to instantiate an application service via the Service Manager:

```
DEFINE VARIABLE oSample AS MyService NO-UNDO.

oSample = CAST(Ccs.Common.Application:ServiceManager:getService(
                                            GET-CLASS(MyService)), MyService).
IF oSample:getSomething() THEN ...
```

6 References

Chase, S. B., Smith, R. & Elwell, R., 2016. *CCS Specification: OpenEdge Application Architecture Version 1,* s.l.: Progress Software.
Judge, P., 2016. *Service Manager Specification: Technical design document,* s.l.: s.n.
Ormerod, M., 2006. *Defining The Openedge® Reference Architecture - Common Infrastructure,* s.l.: Progress Software.

Document Control

Title: Startup Manager Startup Manager SpecificationOpenEdge Application Architecture Specification (OEAA)

Version: 0.3

Document History

Date	Version	Author	Change Details
2016-05-04	**0.1**	**Simon L Prinsloo**	**Initial draft**
2016-05-18	**0.1.1**	**Simon L. Prinsloo**	**Incorporate comments from team discussion.** **Add class diagram.** **Add draft source code.** **Expand introduction and scope**
2016-05-26	**0.1.2**	**Simon L. Prinsloo**	**Incorporate comments from team discussion.**
2016-06-13	**0.2**	**Simon L. Prinsloo**	**Accept all changes and incorporate feedback.**
2016-06-13	**0.2**	**Peter Judge**	
2016-06-014	**0.2**	**Robin Smith**	
2016-06-14	**0.2**	**Simon L. Prinsloo**	**Update all fields, as some fields and the Bibliography reflected old values.** **Change references to Ccs.Common.Framework to Ccs.Common.Application.** **Minor editing, e.g. breaking apart long sentences, inserting missing citations and re-ordering paragraphs in section 4 to improve the flow of the discourse.**
2016-06-15	**0.3**	**Simon L. Prinsloo**	**Incorporate spec team feedback and accept all changes.**
2016-06-16	**0.3**	**Simon L. Prinsloo**	**Remove italics formatting in section 1.4.** **Change second example in section 5 to be consistent with the first one.**

			As noted by Mike Fechner, v. 1.0 should be the final version accepted after community input. Thus change the version from 1.0 to 0.3.