Constellation™

A Real-Time Software Framework

Overview, Capabilities and Benefits

Constellation Version 1.0
Welcome to Constellation

Constellation™ is a complete Software Platform, providing end-to-end development support for real-time, intelligent control systems. It offers a comprehensive combination of facilities, services, and tools to speed system design and implementation, ease integration, and manage process complexity.

This overview guides you through the main features of Constellation and discusses how they address the challenges facing many large-scale systems development projects.

1 What is Constellation?

Constellation is:

- An Integrated Tool-Chain that offers the right tool for all tasks, at all stages of project development.

Its main features include:

- Hierarchical Composer that provides an intuitive, top-down understanding of the system architecture.
- Component-based Construction that supports and encourages modular design and software reuse.
- Robust Software Suite that jump-starts development by providing many core services for the application.
2 Why Constellation?

2.1 Application Complexity

(Constellation is the Software Platform for developing Intelligent Control Systems. These systems are extremely challenging to design, build and maintain, mainly because:

- They possess complex, modal-driven behaviors that are awkward to describe and implement.
- They have stringent scheduling and resource requirements.
- Run-time bugs are difficult to catch and resolve.
- Their distributed deployment requires robust networking support.
- They need to adapt as hardware technologies evolve.

2.2 Process Complexity

In addition to the technical barriers, the sheer complexity of the development effort brings an additional set of challenges that can threaten process efficiency. Do you ever find yourself struggling with:

- Coordinating a multidisciplinary team, with each member having different skill sets, using different vocabularies, and preferring different tools?
- Maintaining a large code base spanning multiple product lines?
- Wanting to reuse software from legacy systems?
- Communicating the high-level design to management or other stakeholders?
- Keeping the documentation current?
2.3 Constellation Provides Proven Solutions

*Constellation* lets you confront these problems head-on by providing a combination of core capabilities and an open-architecture designed to permit easy and seamless integration with 3rd-party “Best-of-Class” tools, as depicted in Figure 1. This focus gives *Constellation* the optimal trade-off between utility and flexibility:

- **As a framework,** *Constellation* reduces development time, effort, and risk by resolving the critical infrastructure-related needs of your application. Moreover, its focus on components naturally leads to implementations that maximize maintainability while ensuring forward compatibility.

- **As a development environment** integrated with a wide range of solutions, *Constellation* ensures that the best tool is available at each stage of development.

- **Because it spans the entire development cycle,** as shown in Figure 1, *Constellation* allows you to efficiently manage your process. As the common development medium, *Constellation* breaks down communication barriers between system architects, algorithm designers, and software developers.

*Constellation* has been successfully used by teams consisting of as few as one person to as many as hundreds of developers, separated by geography and time. Applications developed using *Constellation* span a wide domain. These include:

- Radar Tracking Systems.
- Hardware-In-The-Loop Testing and Simulation Systems.
- Intelligent Transportation Systems.
- Semiconductor Handling Systems.
- Surgical Robotics.
- CNC Manufacturing Equipment.
- Space Shuttle Launch Control System.
- Unmanned Ground, Sea, and Aerial Vehicles.
Whatever your application or however complex your requirements, *Constellation* can scale to meet your needs.

**Figure 1** The Application Development Cycle

![Application Development Cycle Diagram]

Constellation spans the entire Application Development Cycle through a combination of core services and integrated 3rd-party tools.

### 3 The Constellation Design Philosophy

The *Constellation* design philosophy augments the power of object modeling ([top-down](#)) design with the leverage of component-based synthesis ([bottom-up](#)) implementation.

Top-down design is an intuitive process that consists of dividing a complex problem into successively smaller parts. It is a powerful, general approach that provides the 10,000 foot perspective, critical for communication among managers, engineers, and programmers.
In contrast, bottom-up implementation attempts to piece together the solution from preexisting components. Bottom-up leverages development effort by sharing its cost through multiple component reuse. 

*Constellation* exploits the respective advantages of both approaches so that you can rapidly create well-designed applications from interchangeable parts.

### 3.1 Top-down Software Organization

*Constellation* application development is naturally “top-down” because a *Constellation* diagram both initiates design and becomes the implementation. For example, consider the *Constellation* diagram of an Intelligent Cruise Control (ICC) application, as shown in Figure 2. For the software architect, the diagram represents the system as it ought to be. That is, the diagram logically groups the major subsystems and describes their relationships and dependencies; it provides a roadmap for how development should proceed.

To *Constellation*, however, this block diagram represents the implementation. In other words, the diagram defines the actual implementation mechanics, including what objects need to be linked into the application, how data/function pointers should be passed, and when to schedule execution.

Of course, the subsystems in Figure 2 are, themselves, extremely intricate. With *Constellation*, you can hierarchically decompose each subsystem into its own set of diagrams. *Constellation* lets you continually build the hierarchy from the top down (depicted in Figure 3), until you are satisfied with the appropriate level of granularity.

### 3.2 Bottom-up Programming

Just as *Constellation* allows multiple layers of top-down design, it also supports multiple levels of bottom-up componentization. With *Constellation*, you can create highly reusable, primitive components that have general utility. You can also combine primitive components into composite groups to create powerful, custom solutions.
At the most primitive level, Constellation components are implemented in C++ code. This viewable source-code contains a combination of user-supplied and auto-generated code. You supply the code that implements the object’s desired functionality. Constellation automatically generates the necessary definitions and services required to transform your object into a Constellation “component.” This process is visualized in Figure 4.

Constellation ships with many standard components. You can incorporate these directly into your application, or you can use them as examples or guides to create new components. And because Constellation components are true C++ objects, you can derive new components from existing ones. This allows you to leverage existing bodies of work with minimal overhead, thereby reducing implementation effort and avoiding extra validation and verification.
3.2 Bottom-up Programming

3.2.2 Composite Object Groups

At the next level, Constellation allows you to form composite components from primitive components. These Composite Object Groups (COGs) may, in turn, be combined into other COGs. Indeed, a COG is nothing more than the block diagram shown in Figure 2.

The advantage of COGs is that they are graphical. As a consequence, they are much easier to understand, maintain, and/or modify. And once created, new COG components become reusable objects archived within repositories that may benefit future projects or other development teams.
3.3 Interfaces are the Key

Interfaces play a central role in Constellation's design philosophy because they enforce component reusability. Constellation makes it easy to create, modify, and manage interfaces so that they become the focus of the development process (Figure 5).

The result is that the design becomes interface-driven. This approach naturally facilitates parallel development because an interface formalizes a component's contract. Thus, as long as the terms of the contract are enforced, a component remains valid regardless of when it was implemented.

Interfaces also allow you to assemble systems from preliminary component implementations at a much earlier stage. Then, as each component matures in development, the
system can be upgraded in piecewise fashion. This “spiral” mode of development is especially effective in rapidly developing deployable systems that are easily upgradeable in the field.

Figure 5  Managing Interface Objects

Constellation provides facilities to help simplify the management of interface objects. You can use the Form View editor to create or modify the interface objects. Constellation will automatically convert the specifications into C++ code.
The Constellation Application Framework

The Constellation Application Framework enables component-based programming. It gives structure to the application, and it provides the infrastructure to support component insertion, such as:

- A mechanism to specify and maintain components.
- Facilities to bind components.
- Services to allocate and manage computing resources.
- A scheduler to regulate program execution.
- A means to process external asynchronous events and sequence the response.

However, the potential payoff for component reuse can be huge. By increasing reliance on existing modular software, you can reduce unit development and testing, ensure more uniform quality, and shorten development cycle time. Additionally, larger, more complex applications can be developed with minimal verification and validation effort.

With Constellation’s Application Framework, you can enjoy these advantages without having to develop, debug, and test your own infrastructure. And since the framework guides design, component-based development flows naturally without excessive effort during integration.

Taken together, these cost-savings can become significant, resulting in your being able to focus more resources on innovation rather than reinvention.

4.1 Framework Provides Program Structure

Real-time, intelligent control systems require both asynchronous, event-driven processing (i.e., behavior) and periodic, sampled-data processing (i.e., performance). As a consequence of this duality, text-based implementations often result in a tangle of logic statements intertwined with control loops. These implementations are both error-prone and difficult to maintain.

Constellation’s framework provides explicit constructs to handle the hybrid nature of intelligent control systems at a graphical level. The result is clean, self-documenting solutions that are much easier to understand and debug.
4.1 Framework Provides Program Structure

4.1.1 Asynchronous Event-Driven Behavior

*Constellation* lets you model the **event-driven behavior** using Finite State Machine (FSM) diagrams (an example is shown in Figure 6). *Constellation* simplifies the process: you create the FSM diagrams using *Constellation*’s graphical editing utilities, then *Constellation* automatically implements these diagrams during run-time execution. There is no need for you to hand-edit any source code to obtain the benefits of the implementation.

The syntax and semantics of the *Constellation* FSMs are consistent with those of the Unified Modeling Language (UML™) State Chart diagrams. This lets you directly implement the system’s behavioral requirements exactly as they were specified in UML. Moreover, the resulting implementation will be universally understood because they are based on well-recognized standards.

![Finite State Machine Diagram](image)

*Constellation* FSM diagrams have the same semantics as the UML State Chart diagrams. Concepts such as states, transitions, events, levels, and history are all supported.
4.1.2 Periodic Data Flow

*Constellation* automatically schedules your periodic data flow components for execution so you can avoid having to maintain multiple nested loops that must be updated as components are introduced into or removed from the system (see Figure 7).

Figure 7  Automatic Execution Ordering

Based on the data dependency, Constellation will schedule execution in the order specified by: ComponentA -> ComponentB -> ComponentC. You can override the ordering using pop-up menus.

4.2 Framework Provides Services

A central challenge of component-based development is the problem of integrating components into an application with crosscutting constraints, such as deadlines and resource requirements. A second obstacle stems from the need to bind components for execution. *Constellation* addresses these concerns at the application level through the concept of service mappings. These mappings include habitats, systems, and modes.

4.2.1 Habitats

*Constellation* habitats provide the application’s execution environment. They are essentially managers that create and control system (i.e., OS-specific) resources, such as:

- Threads and thread priorities.
- Sample rates.
4.2 Framework Provides Services

- Message queue lengths.
- Stack sizes.

With habitats, the crosscutting constraints are satisfied at the application-level, rather than at the component-level. This significantly increases a component's reuse potential because its implementation is no longer tied to a particular system, or limited by available resources. Thus, you can develop a component and expect to use it on any hardware or OS supported by Constellation.

Best of all, Constellation gives you full access to a habitat's parameters through a GUI interface. This gives you full control of resources, without having to sacrifice modular implementation.

4.2.2 Systems

**Constellation systems** allow you to specify different configurations of the same application. This useful concept lends continuity to the development cycle because the same set of diagrams may be used across all stages.

As an example, consider a typical controller shown in the diagram in Figure 8. Ideally, this controller should be tested at increasing levels of operational fidelity, i.e., numerical simulation and hardware-in-the-loop.

You can specify that the HARDWARE system should include components that communicate with hardware. In contrast, the SIMULATION system should include those components that simulate hardware behavior. This lets you use the same diagram to conduct both the numerical simulation and the hardware-in-the-loop tests.

4.2.3 Modes

**Constellation modes** provide the structure that links the application's event-driven and periodic behaviors. They contain the components that provide the periodic behavior. At the same time, they can be activated by the event-driven part of the application.

To illustrate, suppose you wanted to design an Intelligent Cruise Control that maximizes comfort during normal operating conditions. Ideally, this controller should also include a design that minimizes error—at the expense of comfort—during emergency conditions. With Constellation, you can create components that satisfy both criteria and group them into **NORMAL** and **EMERGENCY** modes, as shown in Figure 9, so the appropriate design will execute only when desired.
Figure 8  Systems

Constellation’s Systems allow you to easily configure your deployed application. You can graphically specify which components to link in for each particular System.
4.2 Framework Provides Services

Figure 9  Modes

With Constellation’s Modes, you can assemble periodic components into different execution groups. In the Intelligent Cruise Control example, you can create a NORMAL and an EMERGENCY mode to provide different behavior for different operating conditions.
4.3 Framework Provides Seamless Networking Support for Distributed Execution

Do you need to deploy a control or simulation system that distributed over several computers? Do you have a remote sensing application?

The Constellation framework provides seamless networking support so that you can have the needed connectivity without added effort.

You can deploy your components in a single application, or you can deploy the same components in multiple applications distributed over a network. The choice between “local” or “distributed” can be made, and remade, at any time during the development process. This imparts maximum flexibility to the design of your distributed application, while maintaining a component’s reusability potential.

To meet the full range of communication needs, Constellation provides support for both a data distribution service and a client-server broker by using:

- Network Data Delivery Services (NDDS®).
- Common Object Request Broker Architecture (CORBA®).

4.3.1 NDDS

NDDS® is an anonymous data distribution service based on the publish-subscribe transaction model. This means that as an application developer, you do not need to know who is producing the data your application requires. Likewise, you do not need to establish who is consuming the data your application creates.

NDDS takes care of all necessary tasks to route the data. You just have to specify in the application diagram which component requires NDDS support (see Figure 10). There are also a number of “Quality of Service” parameters you can adjust to fine-tune the network performance.

Concerned about interoperability with other applications?

NDDS supports an emerging standard, called the Data Distribution Service (DDS), being developed by the Object Management Group (OMG). This ensures that data delivered by NDDS can be read by other middleware conforming to the DDS standard. More information about NDDS can be found at our Website: http://www.rti.com.
4.3 Framework Provides Seamless Networking Support for Distributed Execution

4.3.2 CORBA

CORBA® is the distributed client-server broker architecture based on a set of standards defined by the OMG. With Constellation’s CORBA integration, networked components can interact as if they are residing on the same host machine.

The CORBA Object Request Broker (ORB) automatically routes remote method invocations so that your component does not need to specify any network settings or addresses.

And as with the NDDS integration, CORBA integration is completely seamless. Consequently, you only need to specify in the diagram which component needs CORBA support.

Constellation eliminates the need for network programming. You can use popup menus in the diagram to indicate when a component will need network support.
5 The Constellation Development Environment

Constellation is effective as an end-to-end Development Environment because it fits into your process. Constellation allows you (or your developers) to choose the solution that best fits the particular task at hand.

To see how Constellation works within your process, let us explore the stages of an example development project as seen through the individual contributions of a fictional engineering team. The team may be composed of several members (or it may consist of a single person playing several roles):

- Requirements engineer; has a deep understanding of the needs of customers, but limited software development experience.
- System architect; also doubles as a software developer; most comfortable working in a Microsoft Visual Studio® development environment.
- Modeling specialist; has a background in dynamics and kinematics; experience using the MathWork’s Simulink®; little programming expertise.
- Algorithm designer; with a background in controls; prefers working in the MathWork’s MATLAB® environment.
- System integrator; embedded system specialist.

These members have clearly different tasks, come from different backgrounds, and probably speak in different terminologies. How then can we coordinate their efforts in a way that avoids communication bottlenecks, smooths workflow, and minimizes duplication of effort?

5.1 Stage 1: System Specification

As with most projects, the effort is usually initiated by identifying the requirements. The problem, however, is that it is often difficult to map the requirements to evolving software.

In this regard, a powerful tool from Telelogic, called the Dynamic Object Oriented Requirements System (DOORS®), is popularly used to manage and track the system requirements.
5.1 Stage 1: System Specification

5.1.1 DOORS Integration

*Constellation* provides a DOORS integration plug-in that tightly couples the requirements captured in DOORS to the application being developed in *Constellation*, as shown in Figure 11. With this integration, you can:

- Link your requirements to specific software elements in *Constellation*.
- Analyze the impact of changing requirements on the application’s design.
- Verify that the design satisfies all requirements.
- Understand how changes to the design will affect the fulfillment of the requirements.
- Identify potential redundancies, inconsistencies, or extraneous elements in the application design.

![Figure 11 DOORS Integration](image)

*Constellation’s integration with DOORS ensures that requirements are directly mapped to specific software elements.*
Constellation’s tight integration ensures that the implementation will not drift from the requirements. In turn, this substantially reduces the risk of project creep, and eliminates unnecessary design iterations.

5.2 Stage 2: Architecture Design

A natural segmentation occurs between the requirements and the architecture design stages. This problem is further compounded by the fact that requirements engineers (domain experts) and system architects (software experts) usually hail from different backgrounds. These factors lead to a heightened risk of miscommunication, which leads, in turn, to poorly conceived designs and inefficient execution.

Constellation reduces this risk by providing graphical tools to model, document, and create the architecture. The results are graphical, so they can be easily communicated across the team. They are also synchronized to the DOORS requirements, so that changes to the architecture can be automatically tracked and propagated to the requirements.

Constellation provides a total of five “views” to describe a system’s architecture. Three of the views are used for modeling and documentation, while the remaining two views are used to specify execution.

5.2.1 UML Support

Constellation’s three modeling views conform to standard UML semantics (for more information on UML, please visit the OMG Website at: http://www.omg.org). Specifically, these views are:

- Use Case Diagram.
- Class Diagram.
- Sequence Diagram.

A fourth view, State Chart Diagram, is similar to Constellation's FSM execution diagram. Thus, models created in Constellation are rooted in standards, and can communicate with other standards-based tools.
5.3 Stage 3: Implementation

The implementation stage represents the greatest challenge in terms of coordinating the efforts of various developers working along parallel paths. Constellation mitigates the coordination issues by providing a common medium that links your developers together.

5.3.1 Simulink Integration

For example, you can insert your Simulink model directly into a Constellation application (see Figure 12). Simply use the Mathworks’ Real-Time Workshop® to generate code from the Simulink diagram. Constellation provides facilities to import this code into a full software development environment.

Alternatively, you can link your Simulink and Constellation applications together during execution. Constellation supports the Mathworks’ External Mode integration, so the changing value of signals in one running application is reflected in the other.

These two seamless modes of interaction allow your modeling specialists (i.e., Simulink users) and your software developers to work in parallel, instead of in sequence.

5.3.2 MATLAB Integration

Similarly, Constellation also provides transparent MATLAB integration. This feature is especially valuable to algorithm designers who typically prefer to do core development in MATLAB, where they can leverage the power of its many built-in functions to quickly create and tune their algorithms.

Constellation’s MATLAB integration allows algorithm designers to quickly verify their MATLAB-based solutions. For example, the embedded Constellation application that is controlling the hardware can invoke the MATLAB functions that are providing the sophisticated control. As a result, hardware-in-the-loop tests can be conducted at a much earlier stage of the design because MATLAB does not need to be translated to C/C++.

5.3.3 Microsoft Visual Studio Integration

Constellation automatically initializes a Microsoft Visual Studio project for each component and application. Constellation also synchronizes the associated settings, so that your Visual Studio expert can take advantage of Constellation’s effort-saving code-generation, without giving up Visual Studio’s powerful editing and debugging capabilities.
5.3.4 Automatic Documentation

As a general rule, documentation receives perhaps the least amount of attention during the course of a project. Unfortunately, insufficient documentation also presents one of the biggest and most persistent problems during both development and long-term maintenance.

With Constellation, these issues are minimized because documentation is much easier to create, manage and maintain. For one thing, Constellation’s graphical diagrams are self-documenting. Additionally, Constellation allows you to insert notations and comments directly into a diagram. Constellation also generates hyperlinked Web pages for each component you create.

Integration of Simulink into Constellation lets you insert your Simulink model directly into a Constellation application. You can also link the execution of the Simulink and Constellation applications so that changing signal values in one application are reflected in the other.
5.3 Stage 3: Implementation

5.3.5 Build Support for Multiple Platforms

*Constellation* also provides build support for multiple platforms (see Figure 13). As a consequence, different team members can work concurrently on different platforms without added overhead.

This means that your embedded systems specialist (on VxWorks® systems) can work alongside your software developer (on Windows® systems) off the same set of diagrams, at the same time.

Figure 13 Integrated Build Support

![Integrated Build Support](image)

*Constellation* also provides build support for multiple platforms, and thus, different team members can work concurrently on different platforms without added overhead.

5.3.6 Software Management

As a team-wide concern, the management of large amounts of software is a major recurring issue. *Constellation* helps relieve management complexity by providing graphical tools to display, search, and control software.
5.3.6.1 Repository Explorer

Constellation’s Repository Explorer (Figure 14) provides a graphical way to store and retrieve software components, either locally or on a central server. The explorer promotes peer-to-peer sharing and ensures greater visibility into the entire code base.

Figure 14 Repository Explorer

![Repository Explorer]

The Repository Explorer promotes peer-to-peer sharing and ensures greater visibility into the entire code base by providing a graphical way to store and retrieve software components.

5.3.6.2 Component Find

Constellation’s Component Find tool improves the likelihood of reusing a particular piece of software by making it easily accessible.
5.3.6.3 Version Control

*Constellation* allows you to conveniently commit and track edits with integrated software version control. Pop-up menus simplify versioning commands, which encourages tighter conformance to process safeguards. Figure 15 shows integration with Concur- rent Version Control (CVS). However, *Constellation*'s “opentool” paradigm allows you to integrate with any tool you choose.

Figure 15 Integrated Software Version Control

Constellation allows you to conveniently commit and track edits with CVS, or any other tool of your choice, which encourages tighter conformance to process safeguards.

5.4 Stage 4: Debug

The problem of debugging real-time, multi-threaded applications has an added dimension of complexity that renders text-based debugging techniques ineffective. For instance, setting a breakpoint in one thread of execution does not prevent the other threads from continuing. Or for that matter, it may not be desirable to pause execution of code that is controlling critical hardware.
**Constellation**'s integrated debugging tools are designed specifically to analyze real-time applications. **Constellation** lets you plot performance, trace state transitions, and interact with the running application.

### 5.4.1 StethoScope

Unlike graphic plotting utilities that display data only at the end of a run, StethoScope® (shown in Figure 16) allows you to unobtrusively plot selected variables from a running **Constellation** application. This feature significantly reduces debugging time because you no longer have to pause execution to view output data.

You can also dynamically modify variable values during program execution and observe their effects on system performance.

![Figure 16 StethoScope](image-url)

**StethoScope** plots data while the application is running. It has many features useful for monitoring run-time performance, including triggering, buffering, and playback.
5.4.2 LiveLook

*LiveLook* can be used to monitor and/or playback asynchronous events in a running *Constellation* application. *LiveLook* is extremely useful for debugging the behavioral or event-driven aspects of a real-time system from an intuitive, high-level view.

5.4.3 Interactive Shell

Finally, *Constellation* provides a terminal (or shell) into the running application. Among its many features, you can stop or start execution threads, view variables, and navigate through the application.

The shell also incorporates a full TCL interpreter, which allows you to create scripts to execute batch commands. This powerful scripting feature allows you to automate tests to exercise individual units or entire subsystems.

Together, *StethoScope*, *LiveLook*, and the interactive shell offer unmatched capabilities that allow you to quickly catch and resolve run-time bugs that may otherwise remain undetected using conventional debugging techniques.

6 The Constellation Solution

In summary, *Constellation* helps you to resolve the application and development complexities of embedded, real-time control and simulation systems. Its effectiveness derives from its nature as a design philosophy, application framework, and integrated development environment.

As a design philosophy, *Constellation* improves design clarity while achieving economies through software reuse. Its hierarchical "top-down" design, coupled with "bottom-up" construction, allows you to craft clear, self-documenting applications in minimal time.

As an application framework, *Constellation* reduces development time by eliminating the need for application developers to implement many core infrastructure-related services. This reduces both development risk and testing time. Additionally, its seamless integration with the NDDS and CORBA networking middleware reduces the complexity of creating distributed applications.

Finally, as a development environment, *Constellation* increases efficiencies and promotes communication across a team. Its integration with popular products, such as Simulink, DOORS, Visual Studio and *StethoScope* allows engineers to fully utilize the best products available for the many tasks at hand.
The sum effect is a powerful Software Platform tailored for complex system development. Make Constellation the solution of choice for your next control system project. For more information about Constellation, please visit Real Time Innovation's website at http://www.rti.com.