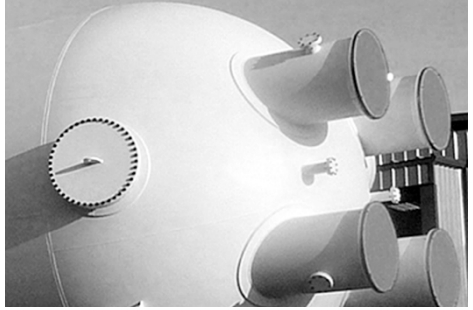


Introducing the New Development Tools

A new Programming Model for R*TIME Applications

What is it?

- A new way to write background and external (display) applications for the R*TIME system (version 13.0 and beyond)
  - C#
  - .NET 4.0 or later
  - All common functions in global assemblies
  - Uses a program to create the initial code for the application, rather than a template
- A new tool for building the R*TIME system
  - Handles C# and Visual Basic applications, as well as legacy C and C++
  - More options on configuring the build system
- A new format for configuration files
  - XML
  - More than one file can be used by an application

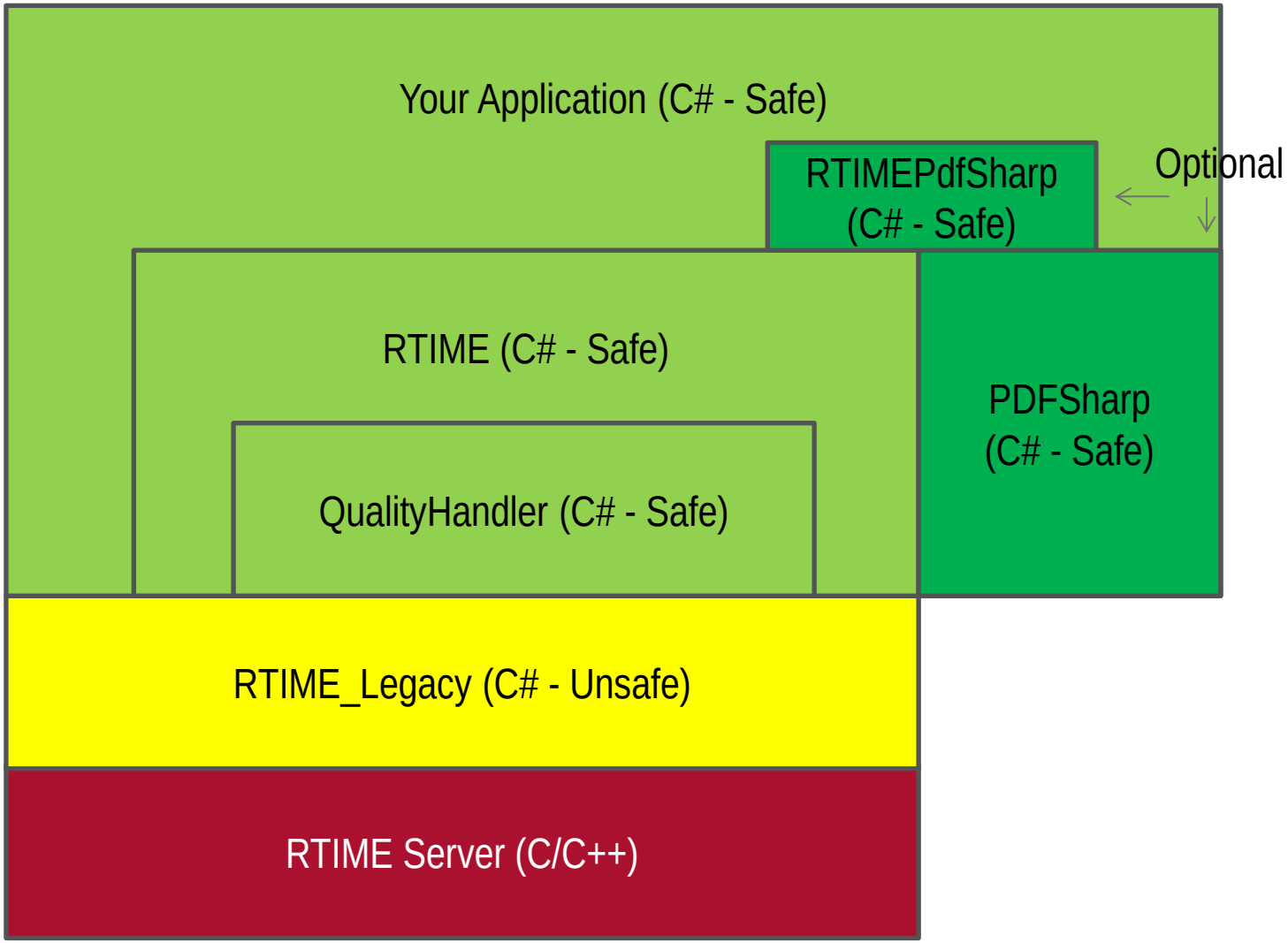
What do we gain?

- A type-safe language
  - Less prone to data corruption from invalid casting
  - Security
    - No pointers
    - All buffers and array access operations are checked for bounds
- Object-Oriented programming
  - Less cluttered code
- Event based applications
- All common code is in an assembly (basically a DLL)
  - System changes can be incorporated without recompiling each application
  - No duplicate coding of common functions, as these are in a library rather than being in each application from the old templates
- Enforcement of programming standards
  - Constants and point names are in configuration files
  - Quality handling is enforced on all operations
  - Error checking built into functions
  - Comments required for objects, variable names checked for Hungarian notation, etc.
  - Common layout of applications
- Faster application development

Can we see it in action?

Demo

Component Layers



RTIME_Legacy

- Wraps many of the legacy RTIME libraries and provides C# interface calls for the functions
  - VSupport (database and CVT access)
  - ExtQueue (sextapps)
  - Archive Retrieval
  - VGroup
  - Messages
  - Steam Library
  - ERFIPC (shared memory library)
- Since the R*TIME server is written in C/C++, it uses pointers. This forces the RTIME_Legacy library to be programmed in ‘unsafe’ mode
  - Type checking and array bounds are still checked
  - Pointer operations are NOT checked for overruns, type conformance, etc.

QualityHandler

- Implements Quality handling rules for the system
  - CVT quality bit mapping
  - Usable/Unusable quality determination
  - Quality priority for operations
  - Error qualities (divide by zero, etc.)
  - Averaging quality handling
- Used by all operations automatically unless explicitly overridden
  - New Project Standards Manual sections to describe base quality handling
  - The QualityHandler object can be overridden and replaced on a per-application basis if that application needs different rules
  - Quality handling can be manually performed and stored for those cases where a small subset of points need unique quality handling

RTIME Assembly

- Provides object definitions for common application data manipulation functions
  - The objects are used to encapsulate data and the operations that can be performed on that data
  - The object definitions can be sub-classed to provide custom functionality when needed
- Provides a standard run-loop object that monitors objects and the R*TIME system for events
  - When an event occurs, the run-loop object will signal the application about the event
    - Applications are written as a series of ‘event-handler’ methods

What events are available?

- Program Initialization
- Periodic Calculation Timer
- Custom Timers
  - Interval Timer
  - Calendar Timer
- Display Activation (external applications)
- Button/Action-Area Click (external applications)
- Server Status Changes
- Value/Quality Changes
- Simulator Freeze/Reset
- Configuration Source Changes (mostly handled internally by objects)

What objects are available?

- Data
  - Has both a value and a quality
  - Math operations are overridden to perform quality handling
  - Knows what type of data has been stored in it (floating point, integer, point VID, time, etc.)
  - Three subclasses
    - DBData – data that is associated with a point in the WCVT/CVT
    - ConfigData – data that comes from the configuration source
    - ArchiveData – data that comes from an archive data fetch
  - Arrays are available for 1-dimensional data tables (including table lookup functions)
- String
  - Has both value and quality
    - Common operations are overridden to perform quality handling
      - Concatenation, etc.
  - Two subclasses
    - DBString – string from the R*TIME database
    - ConfigString – string from the configuration source
- 2D and 3D Data tables
  - Excel templates with macros to create the configuration file representations of the data from a spreadsheet
- Steam Table
  - Static functions for steam table functions
  - Supports both ‘67 and ‘97 steam tables

More available objects…

- AppAlarm
  - Configuration defined application alarm information
- Message
  - Configuration defined message for system message files or display message dialogs
- Sockets
  - Configuration defined listening or initiating socket
  - Interface definitions for receiving packet and sending packet objects
- Fonts, Rectangles, Points
  - Configuration defined objects for report layout and creation
- Timers
  - Create and add to your application to have custom time-based events
- Archive Request Set
  - Gets and manages a data set from the server archive files

Application Objects

- Your application is a sub-class of one of two classes that are derived from the base Application object type
  - BackgroundApp
    - Automatically sets up a watchdog timer from configuration data (if defined)
  - DisplayApp
    - Gets display function-code to handler method mapping from the configuration file
    - Has an array of objects representing the $LOCAL variables, another for the display fields, and one for the drawing areas
- The Application object will automatically handle housekeeping tasks
  - Keeps a list of all configuration sources used and checks for changes – sending a notification event to all objects who used those sources
  - Keeps a list of all timers created – checking for timer 'firings' and calling handler callbacks as needed
  - Checks for active/standby server changes – sends notifications and maintains a global property that can be checked by application methods
  - Monitors threads (applications are multi-threaded) for stall/failure, and responds as needed (failover, notification, system message, etc.)

PDFSharp

- The system now includes the PDFSharp assembly
  - A public-domain library for creating PDF files
  - Very similar in usage to the cpdf library the legacy system uses
  - Full documentation and examples for using the library are abundant on the Internet
- The RTIMEPdfSharp assembly contains “wrapper” object definitions for many of the PDFSharp objects, such as Fonts, Points, Rectangles, etc.
  - The wrapper objects allow the information for these objects to defined in a configuration file, and to automatically update themselves when the configuration file changes (registering for a configuration-file change event)

ProgramCreator

- The ProgramCreator application is used to create the initial project files
  - Visual Studio solution, project, appconfig files
  - Initial source code
  - Initial configuration file
- Header used to specify properties of the application and specify which system events will be handled by the custom code
  - Database specific or generic
  - Watchdog timer setup
  - Initial calculation event rate
  - Events – server status change, initialization, simulator actions, etc.
- 5 or 6 tab pages to specify objects that will be put into the configuration file, with instantiation code and reference variables added to the source file
- ProgramCreator can add additional objects to configuration and source files of existing projects as well

ProgramCreator Objects

- Objects that can be specified through ProgramCreator
  - DBData objects
    - Puts point names in configuration file (per coding standards)
    - Arrays of points or dictionaries of points (with reference tags) can be created
    - Automatic data transfer direction for calculation events can be specified (Input, Output, InOut, Manual)
  - ConfigData objects
    - Allows for easy constant storage in configuration file (per coding standards)
    - Arrays of numbers can be created
  - ConfigString objects
    - Allows for easy string storage in configuration file (per coding standards)
    - Arrays of strings can be created
  - Message objects
    - Allows user messages to be customized without program modification
    - Can specify repeat restrictions (maximum posts without reset, minimum time between postings)
  - Miscellaneous objects
    - Fonts, Points, Rectangles, etc. for PDF reports
    - Socket definitions for communications
    - AppAlarm definitions
  - DisplayEvent objects (Display external applications only)
    - Maps display function codes to methods in program, with security restrictions
    - Stub methods will be created in source file

Excel Templates

- Excel templates have been created for 2D and 3D data tables
  - Templates have instructions and labels for help in using correctly
  - Data cells can reference calculations or other data sources
  - Button tied to a macro that will convert the spreadsheet data into XML and a referencing code snippet
  - Ability to save the XML to a file
  - The XML generated is fully tagged, but it is easier to edit multi-dimensional data in Excel then a configuration file, so it is recommended that the Excel spreadsheet be considered ‘source code’ for the data file, and all data maintenance operations performed on it, then regenerate the XML when changes occur
- 2D and 3D tables can be configured to clip to the table edges or to allow extrapolation, or to change the quality of the returned data (i.e., set to ‘Suspect’) if extrapolation occurs.

CodeManager

- The previous tool used for building the system (CMTool) did not handle C# applications, so a new one has been created – CodeManager
  - Handles C# and Visual Basic program packages as well as the C/C++ (FORTRAN has been deprecated)
  - New interface allows individual packages to fetched and/or built
  - Individual packages can be temporarily removed from the build without losing the configuration entry
  - Can put a C# application into PVCS in a 'flattened' state
  - Understands the different product families (RTIME, Viewer, PMAX, etc.), and will use the global versioning information to label all packages within that family.
  - Has the ability to 'override' individual source files
    - Custom code files can be fetched into base system programs and the program rebuilt without requiring all code for the package to be in the client PVCS
  - Files can be individually configured for inclusion in the package, which PVCS package it should be fetched from, whether it should be deleted with a clean build or not, etc.
  - Three sets of custom commands that can be run with each package
    - Pre-Fetch, before the fetch command runs
    - Pre-Build, after the fetch but before the build phase
    - Post-Build, after the program has been built
  - Updates copyright information with every build
  - Fetches can be done with a named PVCS license, instead of requiring looser security or a default account name

Configuration Files

- Configuration Files are now in XML
  - Fully tagged data (not dependent on order)
  - Data can be nested to more than the 2 layers that .INI files allowed
  - Fully UNICODE compliant
  - Many tools exist to view/edit in more human-readable fashion
    - (Don't try using a straight text editor. The correct tool makes all the difference…)
- Files are now organized consistently
  - Configuration section at the beginning
    - Update rates
    - Watchdog timer settings
    - Console activation
  - Object sections defining each RTIME object used by the application
    - DBData
    - ConfigData
    - Messages
    - Etc.
  - Each object section is sub-divided into a set of 'sections', similar to the old INI files. Within each section are individual 'parameters' – object definitions for each referenced object

Documentation

- Full documentation exists for the public API, ProgramCreator, and CodeManager portions of the system
  - The documentation is updated with every change, as it goes into the Curtiss-Wright PVCS system, so it is always up-to-date
  - The API manual is a hypertext document (in Word). When an object, method, or enumeration is referenced, that reference can be clicked to take you to the definition of the object, method, or enumeration.
- A “C# Application Program Internals” document has been created
  - Describes what “non-public” sections of the RTIME assembly (in particular the RunLoop and Application objects) are doing “behind-the-scenes”

Miscellaneous Additions

- All common messages (“point not found”, “program doesn’t run on PSS”, etc.) are in a single configuration file – StandardMessages.XML
  - Allows easier customization or internationalization of these messages for an installation
- Geographic security has been centralized into a single file – GeographicSecurity.XML
  - Set up as series of zones that a workstation can belong to. i.e. “ControlRoom”, “ShiftSuperviser”, or “ComputerRoom”. Each workstation can be configured to be associated with any number of zones
  - Individual function codes can be configured to only be allowed on workstations associated with a named zone
  - NetSrvr will also use this file to configure Viewer workstation (4.10 or later) with display geographic security restrictions (a display can be configured to only be allowed on a workstation associated with a zone)
- An Excel/VB application has been created to directly pull the point information from appropriately formatted tables in the SDD and insert it into code and configuration files appropriate for the RTIME library

Kevin Coble

Software Engineer