Table of Contents

1 Project Proposal: Real-Time Software Components (RTSC)

1.1 Real-Time Software Components for Embedded Systems

The Real-Time Software Components (RTSC) project is a proposed open source project under the *Device Software Development Platform* project.

This proposal is in the Pre-Proposal Phase as defined in the Eclipse Development Process document and follows the Eclipse Proposal Guidelines. It is written to declare the intent and scope of the project and to solicit additional participation and input from the Eclipse and embedded developer community.

1.2 Background

Component based development has proven itself to have significant benefits in the enterprise IT and web-based environments; complex applications are quickly created and very few, if any, are created from scratch – they all leverage a rich set of third-party open-source or commercial components. Beyond object-oriented language support, component models address all phases of the software lifecycle and standardize software abstractions to the point where – without any ad-hoc conventions or schedule coordination among the participants – interfaces defined by one company can be implemented by a second and used by a third. Moreover, tools that leverage the additional structure imposed by this standardization facilitate the development and (re)use of components, further accelerating the creation of applications assembled from components created by third-parties.

Although real-time embedded systems are increasingly being developed with object-oriented languages and techniques, to enable the same level of cross-company reuse and rapid application development, a component model and supporting tools are needed. However, existing enterprise models (such as JavaBeans, .NET, Corba, etc.) do not address the unique challenges of embedded systems:

- *embedded platforms are extremely cost and power sensitive (Ganssle2006):* to minimize cost and power consumption, a wide variety of CPUs, peripherals, and memories are employed with limited code space and MIPS capacity.
- *embedded software must be "optimal":* to work within the constraints of small memory and relatively slow clock rates necessitated by the cost and power constraints, software must be as small and fast as possible.
- *existing software is predominantly written in C and assembly language (Nass2007):* no standard definition of interfaces nor a common runtime that enables multiple implementations of an interface within a single application exists.
- *no standard C/C++ compiler toolchain exists for all devices:* while GCC supports many CPUs, to achieve the necessary performance from their "portable" ANSI C code bases, developers must leverage C/C++ compilers from the device manufactures that achieve "optimal" performance for their devices.

Several component models have been created to meet these challenges (ECOS Component Model, Koala, Knit, TinyOS/nesC, Real-time Corba, Minimal Corba), but these models and their toolchains are often tied to a specific compiler, embedded operating system, embedded hardware platform, or host development platform. In addition, the more sophisticated models can't be scaled down to support popular but resource constrained devices such as an Intel 8051 or a Texas Instruments MSP430; for example, Corba all but requires a C++ runtime but – because of device memory constraints – no practical C++ support exists for the MSP430. As a result, embedded developers can rarely leverage these models and can't afford to invest the time required to

learn them. Components created for these models can only be used in a limited number of embedded platforms, defeating the opportunity to reuse these components or the skills required to create them in more than just a few closely related projects.

1.2.1 The RTSC Model

> *Sometimes through heroism you can make something work. However, understanding why it worked, abstracting it, making it a primitive is the key to getting to the next order of magnitude of scale.* – Robert Calderbank

The RTSC model and tools enable development of components written in C using *any* compiler toolchain on *any* development host for *any* embedded platform. These components can be configured, assembled, and optimized for use within any embedded real-time system. By focusing on *design-time* rather than on runtime component assembly, the RTSC model and tools enable many of the component-based benefits to scale down to even the most resource constrained embedded system while leveraging existing C/C++ code bases and tool chains.

The RTSC tools, developed over a period of 7 years, are already in use by several Texas Instruments (TI) development groups and have been used to produce "mass market" products such as the DSP/BIOS Real-Time Operating System and the Codec Engine multi-media middleware framework. While these products enjoy the benefits of not having to reinvent the capabilities provided by the RTSC tools, the value of these tools and the motivation to create new tools increases dramatically as adoption of RTSC increases. However, wide-spread adoption is only possible if the model and base tooling are open and freely available.

1.3 How RTSC fits into the Eclipse Ecosystem

The Eclipse DSDP and Tools projects already contain many projects applicable to the development of embedded applications. The goal of the "Real-Time Software Components" project is to complement those existing projects (e.g. CDT), extend Eclipse to provide component based developed environment appropriate for virtually any embedded device with components implemented using C/C++, and encourage extensions of existing component based tools used by Java developers to apply to RTSC components.

The goal is to create a component development platform that can drive the embedded C programmer through the component lifecycle from design (modeling tools), development (C/C++ cross-compiling), testing (unit test frameworks), deployment (package creation tools), and installation (component selection and inter-component compatibility checking).

To be successful, this project needs to foster the development of a rich set of both tools *and* target content. Although TI and some of its third parties currently ship a variety of interesting RTSC components (an RTOS, multi-media middlware, and codecs), a correspondingly rich set of tools built atop the Eclipse platform together with an open RTSC core component model will greatly accelerate the creation of interesting components and component-based applications on a variety of embedded platforms (including non-TI platforms).

1.4 Scope

The goal for the RTSC project is to refine and standardize the core RTSC component model and foundational tools in an effort to bring component-based development advantages to *all* embedded C/C++ developers. The elements included in this project are listed in the Core Architectural Elements section below.

By making this core infrastructure open, extensible, and freely available, we expect to seed additional projects that provide

- more sophisticated tools for component development: unit test frameworks, refactoring tools, etc.
- integration with other popular embedded tools and languages: UML, Doxygen, static checking tools (e.g., Coverity Prevent or Klockwork), etc.
- alternative or domain-specific component composition tools; e.g., a GEF based tool to create an application from existing components or a multi-core component development environment such as Zeligsoft's CE 3.0 product
- rich visualization of component-based applications: graphical representations of the relationships among constituent components, Dependency Structure Matrix tools, etc.
- extensions of existing component-based tools enjoyed by the Java developer to support RTSC components

It is *not* a goal of this project to create the tools described above, rather the goal is to provide the common foundation to enable the creation of these more advanced capabilities by other groups. We want nothing less than a robust component-based development platform suitable for *any* embedded system built atop Eclipse.

Each of these projects benefits from a standard underlying component model and will, if we are successful, bring a complete set of modern component-based tools to all embedded C/C++ developers. In addition, these projects will help shape the roadmap of the core RTSC project by adding new requirements or uncovering new use-cases that need to be supported.

1.4.1 RTSC Overview

The RTSC tools and component model have enjoyed continuous development since 2000 by a small group of senior embedded software developers. Since 2004, the DSP/BIOS 5.x RTOS – created using the RTSC tools – has shipped along with the RTSC tools to ensure that any development system that included DSP/BIOS could consume components (called packages) created by any other development group. Today, internal groups within Texas Instruments regularly (re)build, test, and deploy hundreds of RTSC packages. Many of these packages are used worldwide by thousands of developers both inside and outside Texas Instruments.

- DSP/BIOS 5.x – one of the most popular embedded RTOS's (Turley2006) – is deployed as a bundle of more than 56 packages,
- Codec Engine multi-media middleware runtime (which requires DSP/BIOS) is an independently deployed bundle of more than 21 packages,
- a wide variety of video, imaging, speech, and audio codecs – developed by both Texas Instruments and its third parties – are delivered as a packages, and
- the RTSC toolset itself is delivered as a bundle of over 125 packages

The fact that developers have been using DSP/BIOS 5.x without realizing that it is, in fact, a collection of RTSC components illustrates one of the strengths of the RTSC model: consumers of RTSC components can easily integrate them without converting their entire application into components.

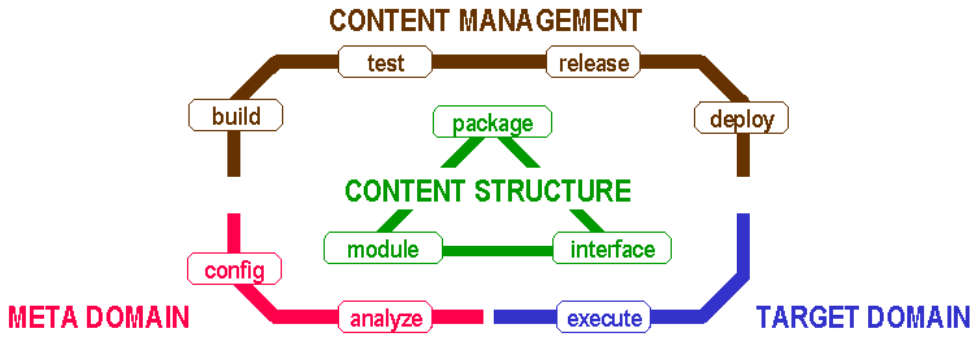
The RTSC tools currently provide basic support for the entire software development cycle: Install, Develop, Debug, and Deploy.

- Install:
    - ♦ package selection, compatibility checks, and side-by-side installation
- Develop:
    - ♦ side-by-side multi-target build with managed toolchains for both cross and native compilers
    - ♦ toolchain-independent package build specifications
    - ♦ component configuration and assembly tool
    - ♦ document generation from component specifications
- Debug:
    - ♦ component-specific views of internal data structures
    - ♦ component compatibility checking

- Deploy:
  - ♦ component packaging tools
  - ♦ on-device real-time logging and diagnostics to monitor system activity

1.4.1.1 Core Concepts

The RTSC component model centers around three top-level concepts: *modules*, *interfaces*, and *packages*. Roughly speaking, modules correspond to Java or C++ classes, interfaces correspond to Java interfaces, and packages correspond to Java jars. Unlike Java, however, RTSC components provide code for two distinct environments: development hosts (with "unlimited" resources) and embedded runtime platforms (with very limited resources). It is this ability for components to operate in and be "configured" on the development host that allows them to scale their runtime requirements to a level appropriate for each embedded system in which they operate.



More specifically:

- All content is logically and physically structured around a trio of programmatic constructs: modules, which encapsulate a related set of types and functions, and have both an external specification and a concrete internal implementation; interfaces, which effectively become abstract modules – a specification without an implementation – that other modules and interfaces can inherit; and packages, which serve as general-purpose containers for modules and interfaces as well as other software artifacts including legacy content.
- All content in the form of modules and interfaces exists in two complementary programming domains: a target domain, where target-content is bound into an application program executing on a particular hardware platform; and a host-based *meta* domain, where associated meta-content plays an active role in the design-time configuration as well as the run-time analysis of target programs.
- All content ultimately resides within individual packages that become the focal point for managing content throughout its life-cycle: all packages are built, tested, released, and deployed as a unit; and while largely self-contained, packages will in general require the presence of other packages that are likewise identified by their globally-unique name and time-varying compatibility key.

1.4.1.2 Core Architectural Elements

The core RTSC tools and runtime support is currently available as a separate product, known as the "XDC Tools" or XDCTOOLS, from TI as a free-of-charge download. After being relicensed under EPL, the entire XDTOOLS product will form the starting point of this project.

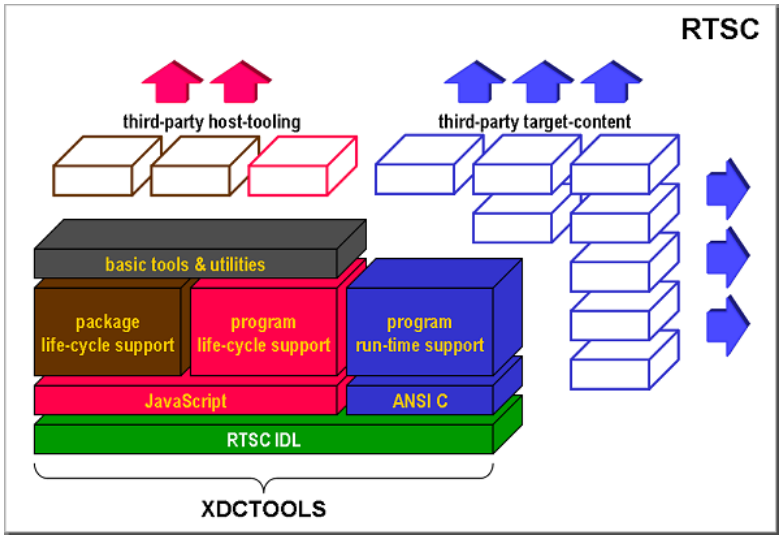
XDCTOOLS is host development system independent and command-line centric; all of the core tools are command-line based and all GUI tools build atop these commands and interface with the user via the Eclipse Standard Widget Toolkit (SWT). With the exception of a few performance sensitive utilities that are written in C, all parts of the toolset are implemented in Java or in JavaScript (executed via Mozilla's Rhino JavaScript engine). Both Windows and *nix development hosts are actively supported.

XDCTOOLS includes the following elements:

- Component Interface Definition Language (IDL) - to specify interfaces all components
- IDL to C/C++ and JavaScript language binding - to generate C/C++ headers and a meta-domain object model used by clients of a component at runtime and design-time, respectively
- Component configuration and assembly tool - to enable rapid creation of applications from components at design-time
- Package build tool - to create a deployable component (i.e., a RTSC package) from pre-built artifacts and support managed multi-target side-by-side builds of C/C++ and Java sources.
- Embedded runtime support package - a scalable platform-independent runtime that includes printf-like diagnostic support, memory allocation, and basic concurrency control
- Package documentation tool - to automatically generate online package reference documentation from IDL specifications
- Package management tool - to select, check compatibility of, and install specified packages (and their prerequisites)
- Component runtime display tool - to support component-specific views of in-the-field runtime objects managed by the component

All of the elements listed above are part of this project and, by virtue of being openly and freely available, will encourage adoption and extensions of the RTSC component model. These elements form the foundation for extensions that span eclipse-based development tools to deeply embedded C-based runtime support and roughly fall into one of three "levels".

1. *Language Support* forms the basis for all RTSC components and includes the IDL compiler and the JavaScript and C/C++ language bindings,
2. *Core Packages* includes over one hundred packages necessary to create, install, configure, and monitor embedded runtime content on a variety of platforms using virtually *any* C/C++ compiler, and
3. *Essential Utilities* includes both command-line and SWT based tools to view package documentation, manage installation and updates of packages, and view the state of a deployed embedded application.



Language Support. Virtually everything in RTSC starts with the RTSC IDL specification language. The XDCTOOLS product itself includes an IDL translator used to first parse spec files and then (among other things) to generate corresponding C headers as well as client programmer documentation.

XDCTOOLS likewise delivers the JavaScript meta-language, integrated more closely with the IDL through an embedded version of Rhino/JavaScript also shipped with the product; besides relying heavily on JavaScript to support the build/release/deploy cycle of RTSC packages as well as the configure/execute/analyze cycle of

RTSC programs, XDCTOOLS encourages use of the JavaScript meta-language as a general-purpose scripting engine that leverages the power and familiarity of JavaScript.

As for C itself, the XDCTOOLS product does not necessarily bundle any particular compiler tool-chain(s); indeed, the XDCTOOLS can interoperate with *any* ANSI C compiler. At the same time, the product does include knowledge of literally dozens of different C compilers from multiple vendors – so-called RTSC targets, which are actually spec'd IDL metaonly modules coupled with a JavaScript implementation that (in principle) anyone could develop and deliver in their own package.

Core Packages. Moving up a level in the block-diagram, the bulk of XDCTOOLS comprises over a hundred packages containing even more modules/interfaces that broadly fall into three major groups:

- packages with metaonly modules/interfaces that themselves support the general build/release/deploy life-cycle of other RTSC packages; just as many contemporary programming environments bootstrap themselves (e.g., all of Java *is* Java classes), RTSC packages are ultimately managed through other well-known RTSC packages that lie at the core of XDCTOOLS.

- packages with metaonly modules/interfaces that support the general configure/execute/analyze life-cycle of RTSC programs; here again, XDCTOOLS builds upon itself through special JavaScript meta-content that in turn drives the synthesis and analysis of target-content elements in executable programs.

- packages with target modules/interfaces that provide a first layer of run-time support for C programs containing other RTSC modules; portable across all targets, these modules augment the standard C runtime library with better embedded support for pluggable memory allocators, event logging plus error handling, entry/exit of critical sections, as well as overall program startup/shutdown.

The latter content – shipped with the XDCTOOLS product in source-code form, to support migration to new RTSC targets – has its origins in some of the more rudimentary elements of DSP/BIOS. Separating out this functionality from the kernel enables RTSC to serve a much broader class of embedded system environments. By making the XDCTOOLS product openly and freely available (not unlike Sun's *Java Runtime Environment* or Microsoft's *.NET Framework*), we anticipate an ever-growing inventory of interoperable third-party target content populating the world of RTSC.

Next-generation kernels such as System/BIOS 6.00 – itself a collection of RTSC modules/interfaces delivered as RTSC packages – simply presume the presence of the XDCTOOLS run-time.

Essential Utilities. Finally, the XDCTOOLS product incorporates a number of basic utilities for:

- building and releasing packages
- invoking other tools implemented in JavaScript
- generating documentation from specs, and
- managing package repositories

These utilities can be used directly from the command-line or else invoked through extension points within your own development environment. In some instances, we will also include a corresponding GUI tool – delivered as a package (of course!) containing metaonly modules implemented in JavaScript in concert with the Java-based (and JavaScript-friendly) Eclipse/SWT graphical environment. In other situations, we leave the provision of higher-level or domain-specific GUI tooling to others.

Here again, we see broad availability of the XDCTOOLS as a catalyst for others contributing compatible yet complimentary tooling – not so much for general-purpose use, but rather to address the needs of more specialized vertical markets through solutions that couple tooling *and* content more aggressively.

1.5 Organization

We propose this project should be undertaken within the top-level Eclipse Device Software Development Platform (DSDP) project.

1.6 Proposed Project Lead and Initial Committers

- Dave Russo, TI (lead)
- Bob Frankel, TI
- Jon Rowlands, TI
- Sasa Slijepcevic, TI

1.7 Interested Parties

The following companies have expressed interest in the project:

- Ericsson
- Nokia
- S2 Technologies
- zeligsoft

1.8 Code Contributions

We will conduct a review of all potential contributions, as several organizations have developed capabilities similar to what Eclipse RTSC proposes. Those contributions which best align with the goals of the project will be refactored and used as the starting point for RTSC.

1.9 Participation

The success of this project is dependent upon the participation of and adoption by embedded developers. We intend to reach out to this community and enlist the support of those interested in making a success of the RTSC project.

1.10 References

Ganssle, Jack; "What processor is in your product?"; Embedded Systems Design, October, 2006

Nass, Richard; "Annual study uncovers the embedded market"; Embedded Systems Design, VOL. 20 NO. 9, September, 2007

Schmidt, Douglas C. and Vinoski, Steve; "Real-time CORBA, Part 1: Motivation and Overview"; C/C++ Users Journal, December, 2001.

Szyperski,C.; “Component Software, Beyond Object-Oriented Programming”, 1998.

Turley, Jim; "Operating systems on the rise"; Embedded Systems Design, June, 2006

van Ommering, Rob; "The Koala Component Model for Consumer Electronics Software"; Computer, March 2000 (Vol. 33, No. 3) pp. 78-85

1.10.1 Web Links

Codec Engine and xDAIS Framework Components:
http://focus.ti.com/docs/toolsw/folders/print/tmdmfp.html

DSP/BIOS Real-Time Operating System: http://focus.ti.com/docs/toolsw/folders/print/dspbios.html

ECOS Component Model: http://ecos.sourceware.org/docs-latest/

Knit: http://www.cs.utah.edu/flux/alchemy/knit.html

Real-Time and Minimal Corba: http://realtime.omg.org/

Embedded Systems Design Surveys: http://www.embedded.com/columns/survey

TinyOS and nesC: http://webs.cs.berkeley.edu/tos/ and http://nescc.sourceforge.net/

XDC Tools Product Download:
https://www-a.ti.com/downloads/sds_support/targetcontent/rtsc/xdc_3_00/index.html (may require free registration)