RXF Migration Guide

Version 5 to Version 6

for Rhapsody in C Users

Version	1.1
Date	2012-10-26
Created by	Eike Römer
Modified by	Eike Römer
Created for	All RXF and Rhapsody in C users migrating from V5 to V6
Document Status	Released

Table of Contents

Document Change History

Date	Modified by	Notes
2012-10-22	E. Römer	Initial Version
2012-10-23	E. Römer	Improvements and added Screenshots
2012-10-26	E. Römer	Minor changes in structure and corrected typos

Changes from V5 to V6

The RXF is currently available in major version 6. It allows Willert Software Tools to create more customized product releases based on an internally used, highly modular component structure. But also on the customer side migration from a V5 product to V6 involves several changes and improvements.

Product Names and Modeling Tool Support

Until V5 all products have been adapted for the UML tool Rhapsody by IBM Rational. There were only two V5 releases ("EA_Bizet" and "EA_ez430") which were manually created for Enterprise Architect by Sparx Systems. V6 allows to encapsulate modeling tool specifics in a modeling specific component and makes it possible to support multiple UML tools with code generators just as we support a wide range of RTOSes, compilers and targets.

While V5 products have always been names after classical composers like Beethoven, Händel, Strauss or Monteverdi, in V6 we now use product codes which are more technical and not as easy to vocalize. They consist of all important component names (in short form) a product has been built for. This is the big benefit of the new product codes, you can always see what is part of a product and what will be the difference between two installed environments.

Product codes are put together following this scheme:

Modelling Tool	Programming Language	RTOS	Compiler	Target	Evaluation Board*	Extension**

* *The Evaluation Board component may not be part of a product.*
It is usually responsible for allowing LEDs to blink in a sample project on the actual board or even to specify which CPU of a CPU family the product can be used with. E.g. for "Keil_ARM" there are preconfigured products available for the ARM7 board "Keil MCB2130" and the ARM Cortex-M3 board "Keil MCB1700".

** *Extensions are optional and multiple extensions may be combined in on product.*
Possible extensions are "TD" for Embedded UML Target Debugger or "Eval" for an evaluation version of the RXF.

An RXF Product is adapted to each of these components. The components have short names (like EA for Sparx Systems Enterprise Architect or Rpy for IBM Rational Rhapsody). For the product name the components are put together in the order shown above, each separated by an underscore.

Version Numbers

In version 6 releases always have a version number like V6.xx, no minor revision separated by an "r" like in V5 is used anymore. But The new release mechanism now also allows to create custom releases which might have a postfix appended to the version name, e.g. "_beta" for beta versions or "p1" for a patch which has been applied to the latest release.

RTOS / OO-RTX Based RXF Similarity and Performance

The RXF with support for a preemptive third party RTOS is now very similar to the OO-RTX non-preemptive variant. The RXF used in combination with an RTOS is now completely switched over to the RXF which already has been used for a long time together with the OO-RTX by Willert Software Tools. This gives us a much better performance in RTOS based products. But the biggest difference is the footprint. Our V6 Multitasking RXF with Keil RTX reduces the code size by about 60%!

Sample comparison based on the product Rpy_C_KeilRTX_Keil_ARM_MCB1700 versus Strauss, based on a Blinky model:

	V5 with RTOS	V6 with RTOS	OO-RTX*
Code Size (ROM)	18 kB	7.7 kB	4.8 kB
Read-Only Data (ROM)	356 B	324 B	284 B
kPEPS (1000 Processed Events per Second)	68**	103	134

* The grey OO-RTX column is just for comparison to an OO-RTX based solution, which is off course smaller but has the limitation of non-preemptive multitasking.

** Value may be inaccurate, because it was measured with different optimization level and with an older version of the test model.

> The values specified here may be different depending on the versions of the RXF and compiler/linker and depend on the configuration and optimization level. The values inside the table were measured with disabled Highwatermarks in release buildset with the highest compiler optimization for time.

This also shows the Willert Multitasking RXF (including the resource needs of the Keil RTX) requires less than the double footprint than the OO-RTX solution.

Structure and Installation

V5 RXF products were installed completely into the Rhapsody directory structure:

<Rhapsody>\Share\WST_RXF_V5\*<RXF release name>*

Now, with our modeling tool independent approach, RXF files are installed in a "Willert" directory at a location of your choice. On Windows XP it is no problem to put this folder under "C:\Program Files". For Windows 7 it is recommended to use a directory inside the user path, as write permissions to several files under this path are required. To hook into Rhapsody, only Rhapsody's Site<language>.prp file is adapted, tool menu entries might be added and the product's profile is copied to the Rhapsody Profiles folder.

Sending Events

The RXF V6 allows to send asynchronous messages / events via the macros FIRE, FIRE_S, FIRE_ISR, FIRE_ISR_S and includes macros for easy handling of static event arrays to be used in interrupt service routines.

The following table gives an overview of event sending macros available in RXF V6 for Rhapsody in C:

Use Case	Macro	Alternative Macros
Sending events from a singleton object (where no me pointer is available).	FIRE_S(<sender address>, <receiver address>, <event>)	FIRE_SINGLETON
Sending an event from an interrupt service routine realized in a singleton object, where events may not be allocated dynamically and no me pointer is available.	FIRE_ISR_S(<sender address>, <receiver address>, <static event>)	FIRE_ISR_ SINGLETON
Sending an event from an interrupt service routine, where events may not be allocated dynamically, but where a me pointer of the sender is available. This is a rarely used way, as an ISR usually does not have a me pointer available.	FIRE_ISR(<receiver address>, <static event>)	CGEN_ISR
Sending events from anywhere else in the application (from a normal object).	**FIRE**(<receiver address>, <event>)	**CGEN, RiCGEN**
Sending static events from an interrupt service routine which may call the fire function multiple times before events are consumed. Using these macros requires an array of the static events to exist and an index attribute. Initialization has to be done via the macro WST_EVT_initStaticEventArray_S (<source>, <event>, <event array>, <size of array>).	FIRE_ISR_ARRAY(<receiver address>, <event array>, <index attribute>, <size of array>).	-

Sample Usage:

- FIRE(me->myLED, evToggle());
- FIRE_S(&SingletonController, SingletonController.myLED, evToggle());
- FIRE_ISR_S(&IRQ, IRQ.myLED, IRQ.staticEvToggle);
- Sending static events from an interrupt service routine via the static event array:
  - Type with the name "MAX_STATIC_EV_INDEX" of kind Language must exist: "#define %s (4)".
  - Attribute "evToggle* %s[IRQ_MAX_STATIC_EV_INDEX];" with the name "staticEvToggle" must exist.
  - Attribute of type int with the name "staticEvToggleIndex" must exist.
  - Initialization: WST_EVT_initStaticEventArray_S(&IRQ, evToggle(), IRQ.staticEvToggle, IRQ_MAX_STATIC_EV_INDEX);
  - Sending the event: FIRE_ISR_ARRAY(IRQ.myLED, IRQ.staticEvToggle, IRQ.staticEvToggleIndex, IRQ_MAX_STATIC_EV_INDEX);

WSTDeployer

The WSTDeployer is now more independent of the modeling tool. In V5 it used to access tags set inside the component to know e.g. the name of the IDE project file. The WSTDeployer configuration was then used to just select the path where the file can be found. V6 does not access tags inside the model anymore and asks to browse to the IDE project file instead of its directory.

The latest V5 releases also modified the Rhapsody Tools menu by adding an entry to configure the WSTDeployer. This is now always done in V6 and an additional entry to launch the IDE project currently associated with the active component directly from Rhapsody has been added for most environments.

RXF Libraries

Evaluation releases of the RXF do not contain source code of most of the RXF files. They come with precompiles libraries (usually located under Source\LIB and named RXFD.<lib> and RXFR.<lib>).
All full product versions do not come with a library and most of them are not prepared to be used with the RXF in a library. In V5 the first steps for a user of non-evaluation product were always to build the IDE projects "CreateRXFLibrary" and eventually "CreateAnimationLibrary" for both, Debug and Release, build-sets. This is not necessary anymore, the WSTDeployer option, if a library should be used here, is now disabled by default. Source files of the RXF will be deployed into your IDE project and built along with your generated application code. This has several advantages:

First steps are easier.

The problem of an updated compiler without recompiled libraries is gone. If the user first build the RXF libraries, then later updates the compiler and tries to link strange errors, sometimes even during runtime may occur. We experienced such problems via related support requests.

Defines such as WST_CFG_HIGHWATERMARKS or WST_PORTS_DISABLED are now more comfortable, because the libraries do not need to be recompiled with the defines which have been set in the current UML model configuration.

With todays compilers and linkers there are mostly no differences in code size, because unused functions are deleted by the optimizer no matter it it comes from a library or a source code/object file.

Embedded UML Target Debugger

The Embedded UML Target Debugger which was available in the V5 products Liszt, Bizet, Satie and Mozart, was depending on information parsed from the MAP file after building the target application. This made it very specific for the supported MAP file formats and depended on specific linker settings. In V6 we have a new mechanism for the Embedded UML Target debugger which uses class IDs which are send during class construction and destruction along with the instance address. This can actually even reduce the tiny overhead and makes the XMLData.xml file generation independent of the MAP file.

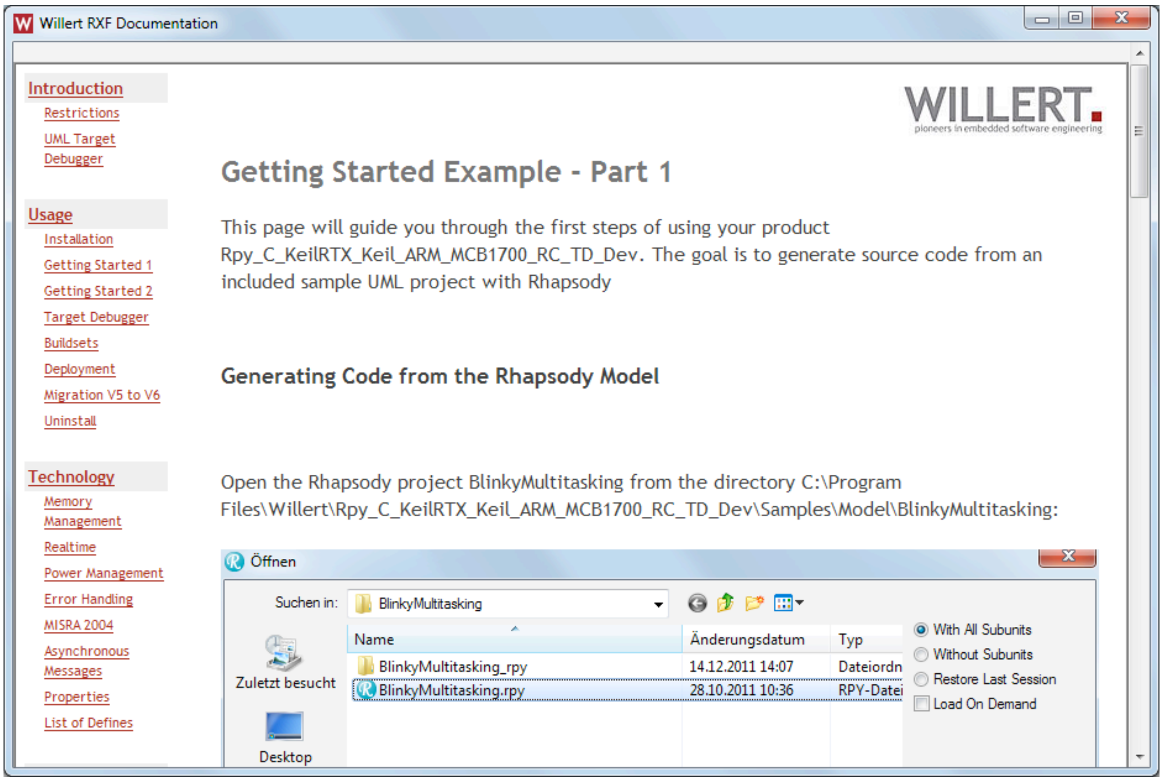
Container Classes

UML models may contain relations between classes which are not 1-to-1 relations, but which have a multiplicity. If the multiplicity is a fixed value, the relation can be realized via a simple array by the code generator. But it it is a "1-to-*" or "*-to-*" relation the relation must be managed dynamically via an implementation of container classes. Depending on other settings of the relation, like it it is ordered and it it has a key parameter for accessing it, container classes like a collection, list or map are required.

RXF V6 products may be delivered with different sets of container classes and different feature support. Please check your product‘s RXF documentation for details about the container classes contained in the release.

RXF Documentation

The HTML documentation had been rewritten and restructured in large part. In addition to the menu structured version it is now also available as one huge HTML file called „WST_RXF_CompleteProductDocumentation.html“ to allow complete printing more easily. The documentation is automatically generated for a specific release and adapted to the relevant components of a product.



Getting Started Example - Part 1

This page will guide you through the first steps of using your product Rpy_C_KeilRTX_Keil_ARM_MCB1700_RC_TD_Dev. The goal is to generate source code from an included sample UML project with Rhapsody

Generating Code from the Rhapsody Model

Open the Rhapsody project BlinkyMultitasking from the directory C:\Program Files\Willert\Rpy_C_KeilRTX_Keil_ARM_MCB1700_RC_TD_Dev\Samples\Model\BlinkyMultitasking:

Name	Änderungsdatum	Typ
BlinkyMultitasking_rpy	14.12.2011 14:07	Dateiordn
BlinkyMultitasking.rpy	28.10.2011 10:36	RPY-Datei

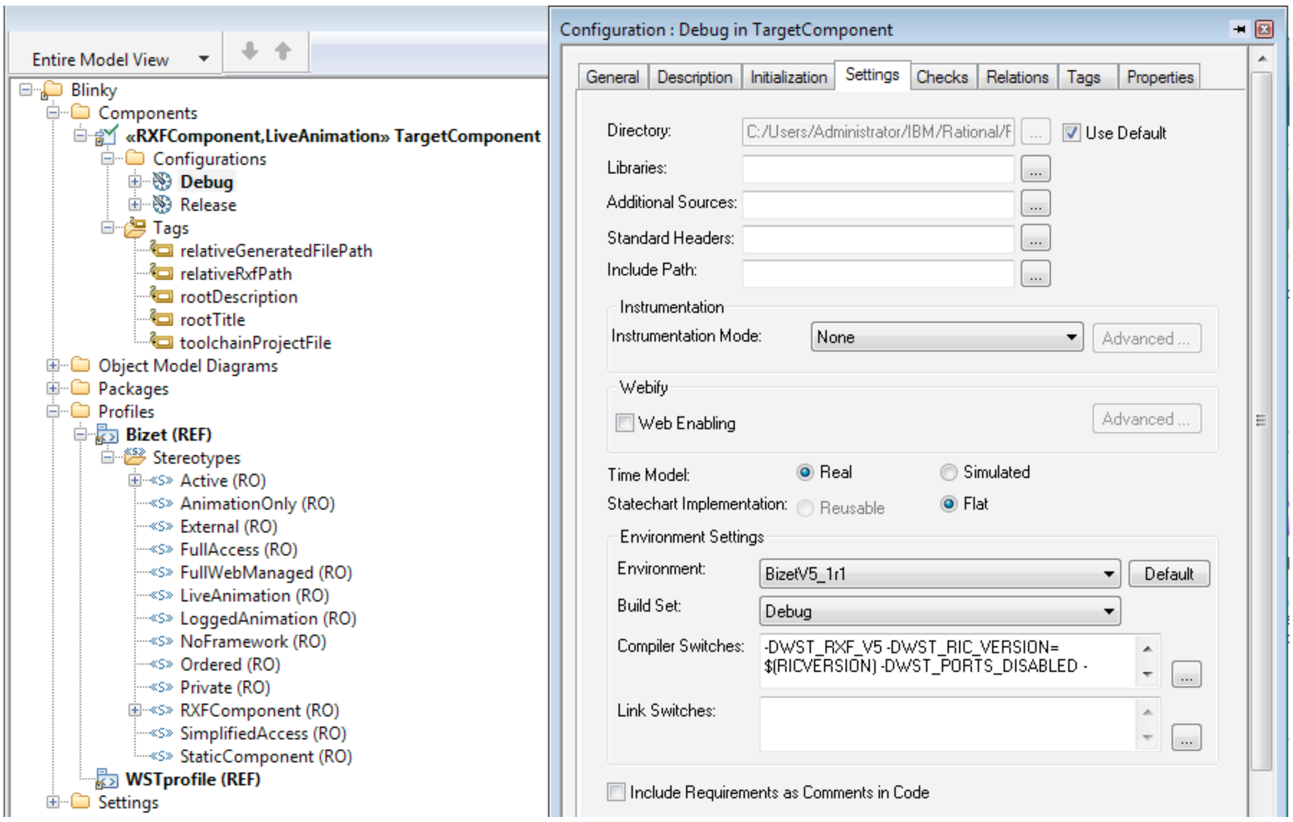
[✓ With All Subunits] Without Subunits Restore Last Session Load On Demand

Modifying an Existing Rhapsody Model

This chapter will guide you through the first steps of adapting an existing Rhapsody UML model which works with a V5 RXF to the new V6 RXF. Both versions can co-exist in the model, so we are not modifying existing components or configurations to use a new environment, but we want to be able to switch between RXF versions.

For our sample we will go through the necessary steps to allow using the sample Blinky model (in this case from a V5 RXF Bizet installation) also with the V6 RXF called „Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05“.

In the Rhapsody Browser you can see the „old“ V5 Bizet profile and the component using it‘s stereotype „RXFComponent“ which sets the environment. Also it is visible that the WSTDeployer is still configured via tags in the V5 component.



Entire Model View

```
Blinky
  Components
    «RXFComponent,LiveAnimation» TargetComponent
      Configurations
        Debug
        Release
      Tags
        relativeGeneratedFilePath
        relativeRxfPath
        rootDescription
        rootTitle
        toolchainProjectFile
  Object Model Diagrams
  Packages
  Profiles
    Bizet (REF)
      Stereotypes
        «S» Active (RO)
        «S» AnimationOnly (RO)
        «S» External (RO)
        «S» FullAccess (RO)
        «S» FullWebManaged (RO)
        «S» LiveAnimation (RO)
        «S» LoggedAnimation (RO)
        «S» NoFramework (RO)
        «S» Ordered (RO)
        «S» Private (RO)
        «S» RXFComponent (RO)
        «S» SimplifiedAccess (RO)
        «S» StaticComponent (RO)
    WSTprofile (REF)
  Settings
```

Configuration : Debug in TargetComponent

General | Description | Initialization | Settings | Checks | Relations | Tags | Properties

Directory: C:/Users/Administrator/IBM/Rational/F [✓] Use Default
Libraries:
Additional Sources:
Standard Headers:
Include Path:

Instrumentation
Instrumentation Mode: None

Webify
[ ] Web Enabling

Time Model: [● Real] Simulated
Statechart Implementation: Reusable [● Flat]

Environment Settings
Environment: BizetV5_1r1
Build Set: Debug
Compiler Switches: -DWST_RXF_V5 -DWST_RIC_VERSION=$(RICVERSION) -DWST_PORTS_DISABLED -
Link Switches:

[ ] Include Requirements as Comments in Code

Install RXF V6 Product

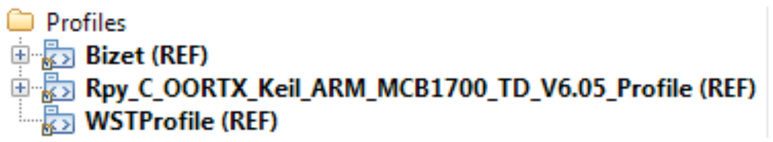
As a first step you need to install the new RXF product. It should be pretty straight forward. For details please see HTML help section „Installation“. The Willert path you will be asked for must not be in a subfolder of Rhapsody and you and our tools need to have write access in that directory. This means it is recommended to use a subfolder in the user‘s home path at least under Windows 7.

Adding RXF V6 Profile

In addition to the old RXF profile, you can now select „File -> Add Profile to Model“ in the Rhapsody model and browse to the new profile to be added:

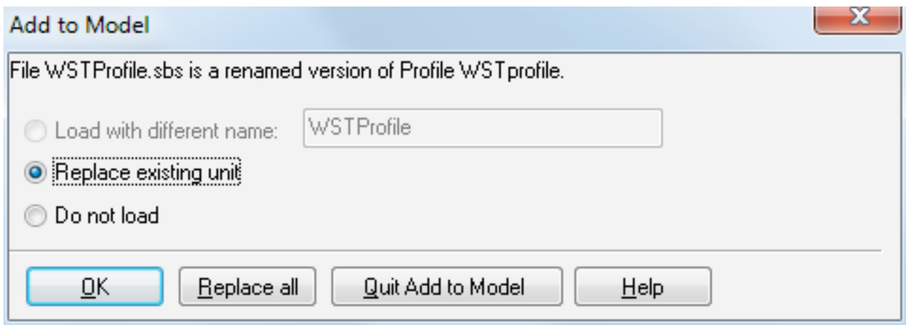
<your Willert directory>\<your V6 product>\
  Config\UserSettings\<your V6 product>_Profile\<your V6 product>_Profile.sbs

For example:
C:\Users\Administrator\Willert\Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05\
  Config\UserSettings\Rpy_C_OORTX_Keil_ARM_MCB1700_TD_Profile_rpy



Profiles
- Bizet (REF)
- Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile (REF)
- WSTProfile (REF)

Optionally you may also use the WSTProfile from the same directory. But different from the product profile it is recommended to only have one WSTProfile in the model, meaning the old WSTProfile should be replaced by the new one.



Add to Model

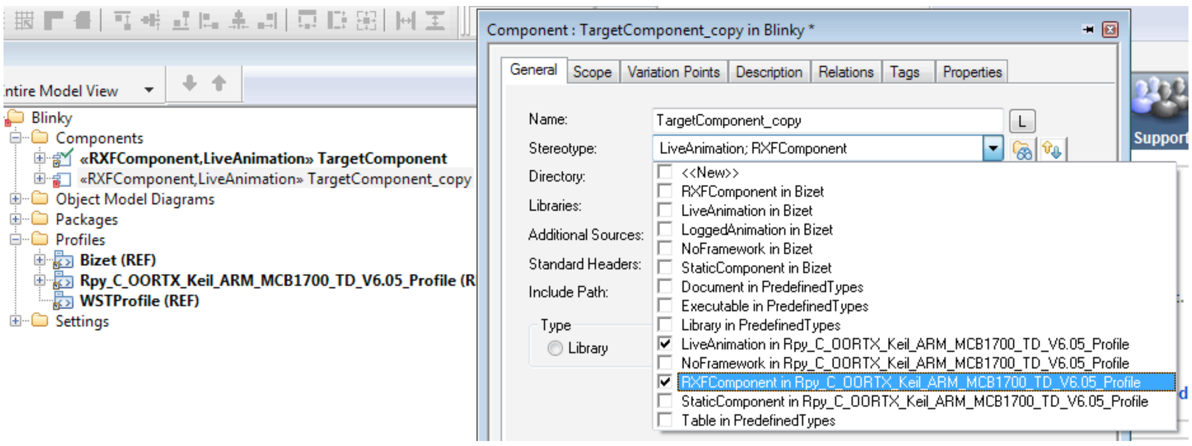
File WSTProfile.sbs is a renamed version of Profile WSTprofile.

- Load with different name: WSTProfile
- [✓ Replace existing unit]
- Do not load

Adding a New Component and Setting Stereotypes

Depending on how complex your configured Scope, initial Instances etc. is, it can be helpful not to create a new Component, but make a copy of the V5 Component.

Open the copied component‘s Features dialog and deselect the V5 Stereotypes and select the appropriate V6 Profile ones.



ntire Model View

Blinky
- Components
  - «RXFComponent,LiveAnimation» TargetComponent
  - «RXFComponent,LiveAnimation» TargetComponent_copy
- Object Model Diagrams
- Packages
- Profiles
  - Bizet (REF)
  - Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile (R
  - WSTProfile (REF)
- Settings

Component : TargetComponent_copy in Blinky *

General | Scope | Variation Points | Description | Relations | Tags | Properties

Name: TargetComponent_copy

Stereotype: LiveAnimation; RXFComponent

Directory:

Libraries:

Additional Sources:

Standard Headers:

Include Path:

Type: Library

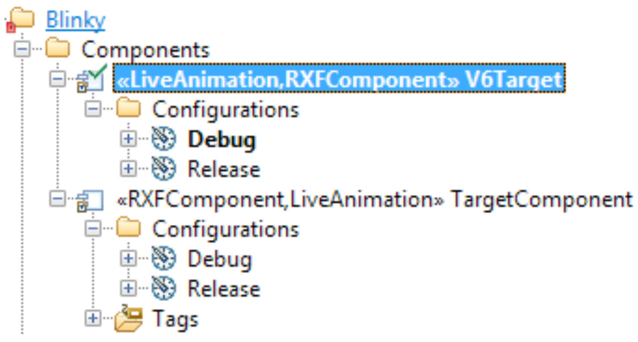
- [ ] <<New>>
- [ ] RXFComponent in Bizet
- [ ] LiveAnimation in Bizet
- [ ] LoggedAnimation in Bizet
- [ ] NoFramework in Bizet
- [ ] StaticComponent in Bizet
- [ ] Document in PredefinedTypes
- [ ] Executable in PredefinedTypes
- [ ] Library in PredefinedTypes
- [x] LiveAnimation in Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile
- [ ] NoFramework in Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile
- [x] RXFComponent in Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile
- [ ] StaticComponent in Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6.05_Profile
- [ ] Table in PredefinedTypes

If your product does not include the Embedded UML Target Debugger you do not need to select the „LiveAnimation“ Stereotype but just the new „RXFComponent“ stereotype.

Give your copied component a more meaningful name, in our sample „V6Target“.

Please carefully go through the Features dialogs of the copied **component** (V6Target) **and** it‘s **configuration(s):**

- The tags can completely be removed, because the WSTDeployer does not need them anymore.
- Any overridden properties on both levels which are inside the subject of the V5 RXF (e.g. „**C_CG::BizetV5_1r1**“) should be un-overridden (by right clicking) and if they still seem to be relevant for V6 they need to be set again under the new property subject (e.g. „**C_CG::Rpy_C_OORTX_Keil_ARM_MCB1700_TD_V6_05**“).
  Filtering for „Locally Overridden“ properties on both levels may be useful.
  If you can not find directly matching properties please consult the RXF properties documentation which is part of the HTML help files.
- If you selected fixed / overridden paths etc. in the features, you need to correct them for the new environment and component.



Blinky
- Components
  - «LiveAnimation,RXFComponent» V6Target
    - Configurations
      - Debug
      - Release
  - «RXFComponent,LiveAnimation» TargetComponent
    - Configurations
      - Debug
      - Release
    - Tags

Make sure the „V6Target“ component is selected as the active component.

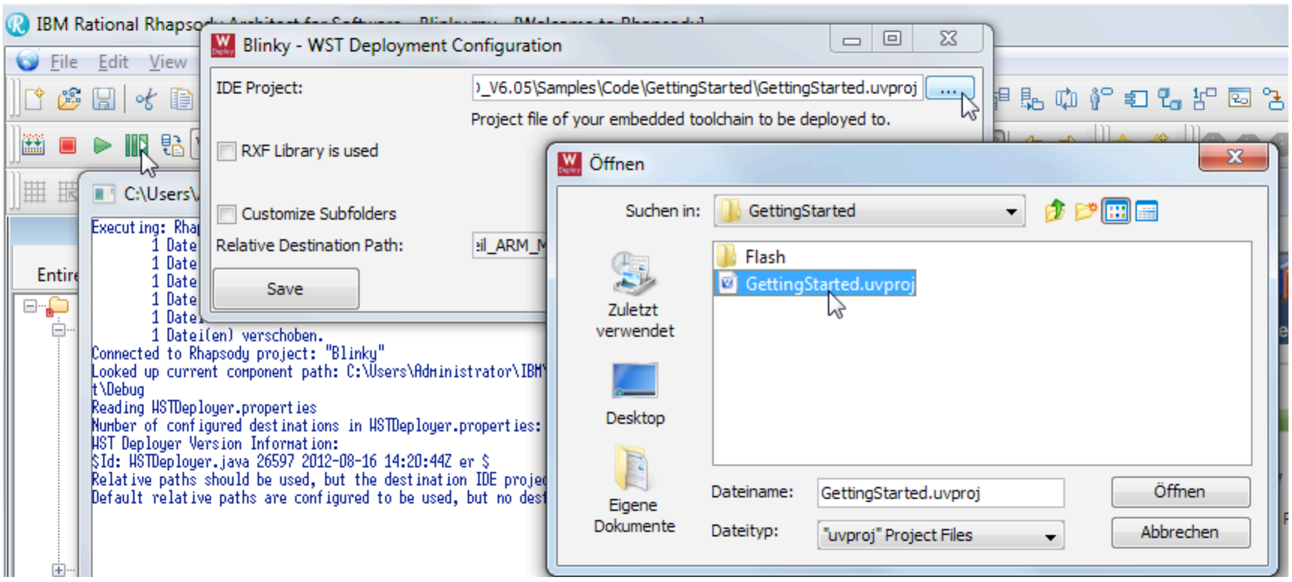
Modifying user code for Event Generation

As described in the previous chapter of this document, the V6 RXF knows new event generation macros like FIRE(). But it is still compatible with CGEN() and CGEN_ISR() macros. You do not need to change them.

The only exception are Singleton classes (which don‘t have a me-pointer) and which send an event when the Embedded UML Target Debugger is part of the product. As it needs to know the event source and the me-pointer is not available in that situation, you need to use a new V6 macro like FIRE_S() instead. For details how these macros are used please see the previous chapter or RXF HTML help.

Configuring the WSTDeployer

The Deployer is still launched via the Rhapsody Run command, and if no configuration for the current component is known, it will ask for the destination IDE Project on the first run. You can now try a Generate/Make/Run command and select the „Samples\Code \GettingStarted\GettingStarted.<IDE project extension>“ file in this sample:



Blinky - WST Deployment Configuration

IDE Project:)_V6.05\Samples\Code\GettingStarted\GettingStarted.uvproj

Project file of your embedded toolchain to be deployed to.

☐ RXF Library is used

☐ Customize Subfolders

Relative Destination Path: :il_ARM_M

Save

```
Executing: Rha
        1 Date
        1 Date
        1 Date
        1 Date
        1 Date
        1 Datei(en) verschoben.
Connected to Rhapsody project: "Blinky"
Looked up current component path: C:\Users\Administrator\IBM
t\Debug
Reading WSTDeployer.properties
Number of configured destinations in WSTDeployer.properties:
WST Deployer Version Information:
$Id: WSTDeployer.java 26597 2012-08-16 14:20:44Z er $
Relative paths should be used, but the destination IDE projec
Default relative paths are configured to be used, but no dest
```

Öffnen

Suchen in: GettingStarted

Dateiname: GettingStarted.uvproj

Dateityp: "uvproj" Project Files

Note that per default the „RXF Library is used“ checkbox is not marked and not all V6 products come with a preconfigured IDE project to build the RXF libraries. So per default all RXF files are also added to the IDE project and it‘s all built together.

Your project should now also work with V6 in parallel to V5!

If there are any questions or problems please contact us via support@willert.de.