LAUTERBACH
DEVELOPMENT TOOLS

Integration for Rhapsody in C/C++

MANUAL

Integration for Rhapsody in C/C++

Integration for Rhapsody in C/C++

Version 06-Jun-2024

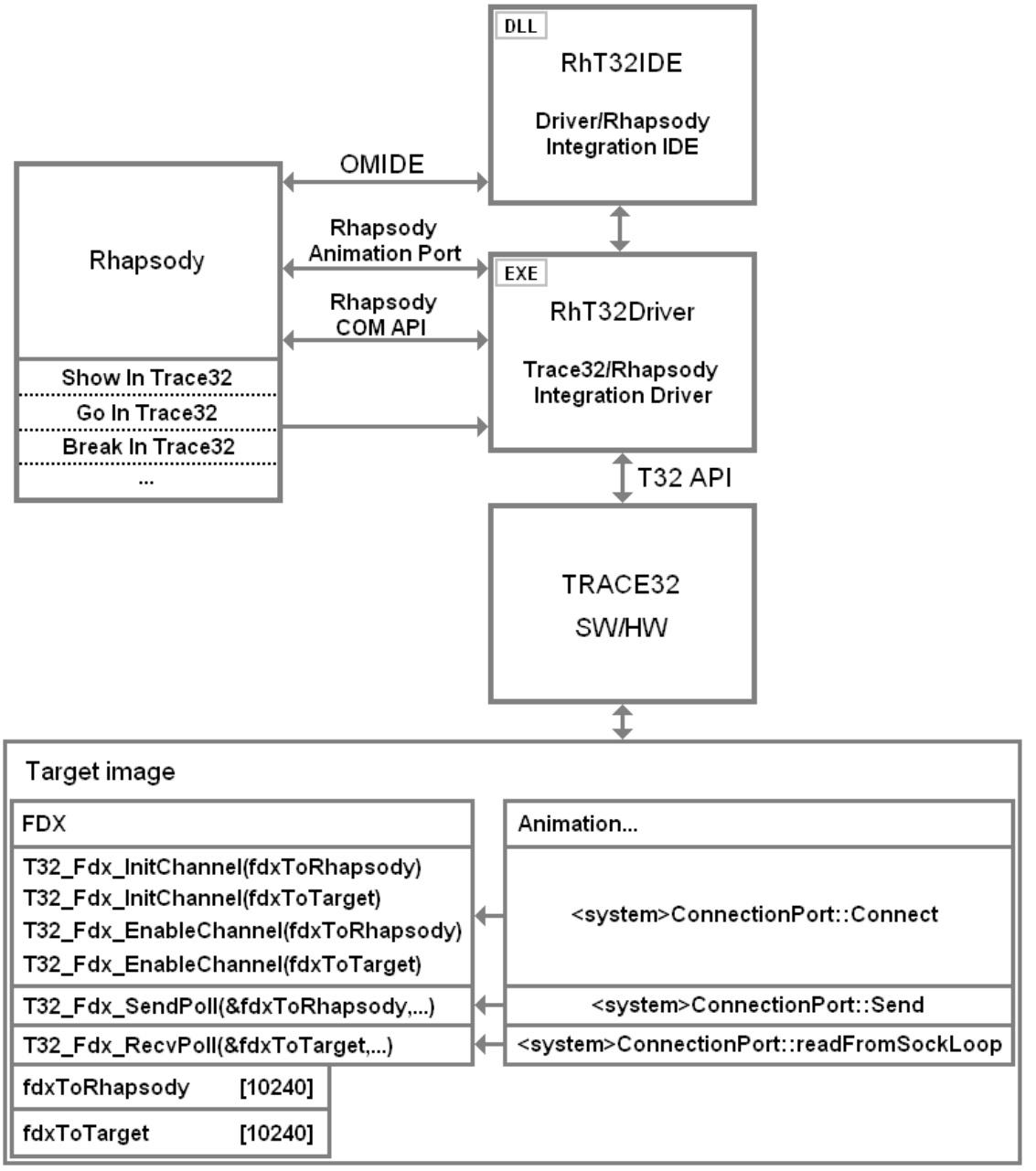
This document describes using the TRACE32 Integration Driver for Rhapsody 7.x or 8.x.

Overview

The TRACE32 Integration Driver for Rhapsody 7.x and 8.x allows you to debug, trace and animate applications created with Rhapsody on a real target using TRACE32-ICD or on a simulated target using the TRACE32 Instruction Set Simulator.

NOTE:	This integration uses internally the **TRACE32 Remote API**. The Remote API has **restrictions** if TRACE32 runs in demo mode. Please see there for further details.

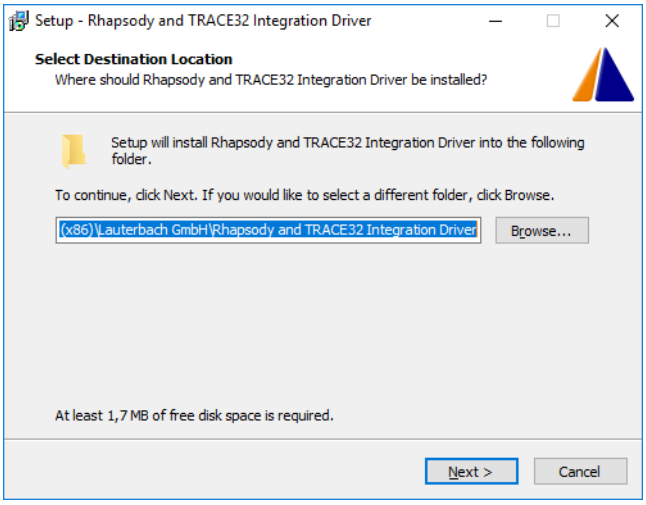
Architecture of Driver



Driver Installation

Before you install the integration driver, make sure Rhapsody is not running.

Locate the installation files in the TRACE32 directory ~~/demo/env/rhapsody/cpp.
Run `setup.exe` and follow the installer instructions:



First Run of Integration Driver

Make sure Rhapsody is not running before starting the driver the first time.

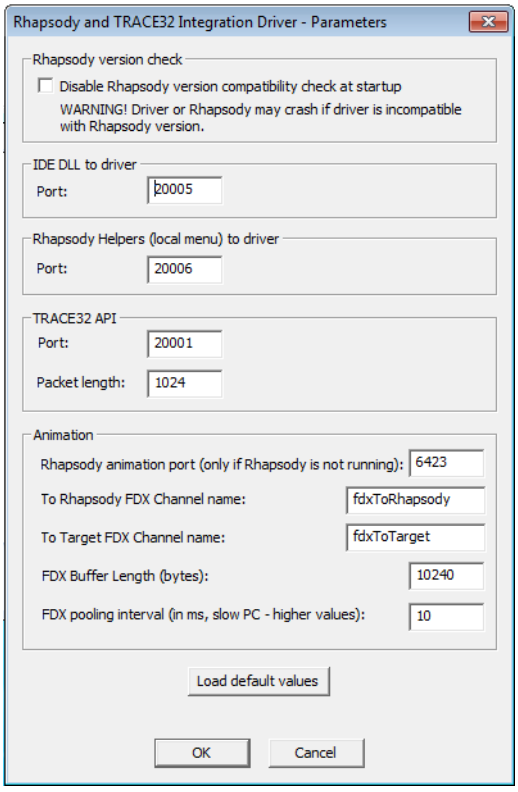
Start the driver from Menu Start -> All Programs -> Rhapsody and TRACE32 Integration -> Start Integration.

The driver needs to access some Rhapsody parameter files an connects to Rhapsody and TRACE32 via ports. For this it needs to run with administrator privileges. Please acknowledge all privilege dialogs opened by Windows.

After the first run you will be asked about several parameters described below, required for the integration working. Provide your own values or select “Load default values“

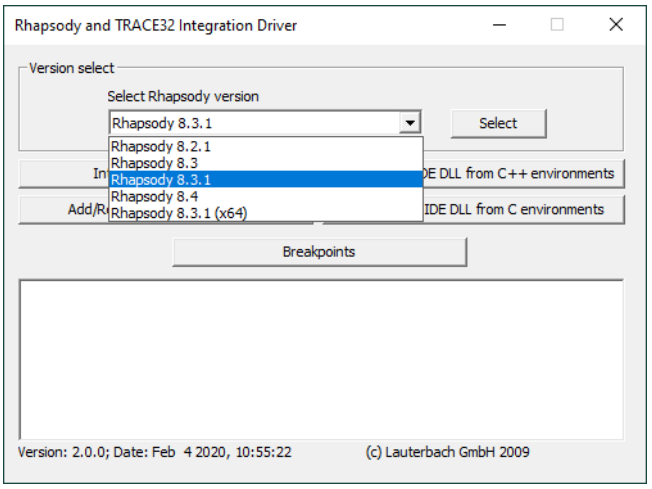
IDE DLL to driver Port	TCP/IP port number for communication between Integration Driver and IDE DLL located at <RHAPSODY_ROOT>\Share\DLLs\RhT32IDE.dll
Rhapsody Helpers (local menu) to driver Port	TCP/IP port number for communication between Driver invoked by Helper commands and Driver which already runs for the integration.
TRACE32 API Port	UDP port number which is used to communicate between Driver and TRACE32. Must be the same as provided in config.t32 file in your TRACE32 installation directory. See **“Preparing TRACE32”** (int_rhapsody_cpp.pdf).
TRACE32 API Packet length	Specifies the maximum packet length in bytes for the socket communication between Driver and TRACE32. Must be the same as provided in config.t32 file in your TRACE32 installation directory. See **“Preparing TRACE32”** (int_rhapsody_cpp.pdf).
Rhapsody animation port	TCP/IP animation port. This port is defined in the rhapsody.ini file. Instrumented applications communicate with Rhapsody on this port during animation session. Default value is 6423.
To Rhapsody FDX Channel name	Name of FDX channel used by the “ConnectionPort” OSAL class. Direction of data in this channel is from target to Rhapsody.
To Target FDX Channel name	Name of FDX channel used by the “ConnectionPort” OSAL class. Direction of data in this channel is from Rhapsody to Target. See **“Rebuilding OXF LangCpp Libraries for OSE, DIAB and PPC”** (int_rhapsody_cpp.pdf).

FDX Buffer Length	Size of FDX channels. This value must be the same as used by the “ConnectionPort” class. See **“Rebuilding OXF LangCpp Libraries for OSE, DIAB and PPC”** (int_rhapsody_cpp.pdf).
FDX Polling Interval	During animation, the driver periodically checks if there is any data pending in FDX channels. This value is interpreted as number of milliseconds between channel checks. For a slow PC or when using the simulator, this value should be bigger (50-100 ms). Default value is 10ms.

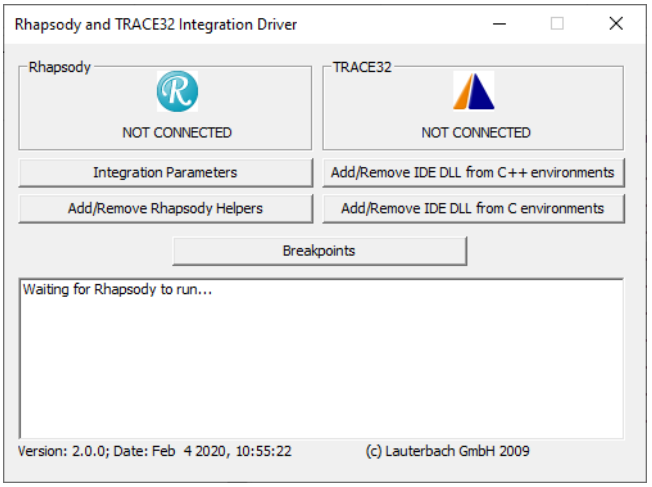


Selecting Rhapsody Version

During startup the integration driver queries the Windows registry to detect all supported Rhapsody versions that are installed in the system. When detection completes successfully ad at least one version is present, the combo box in the upper part of the main dialog is populated with the results.



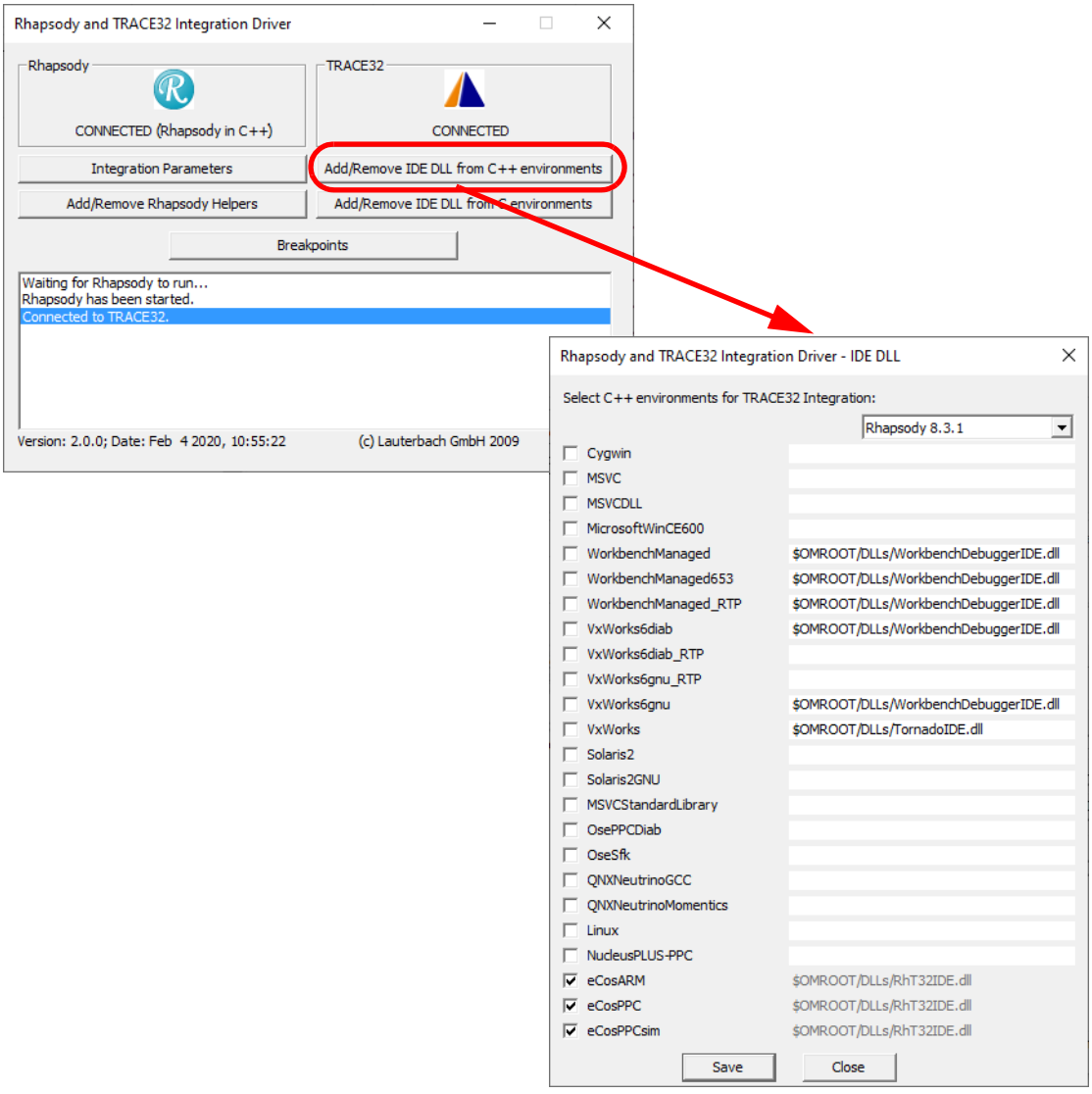
The Integration Driver is started when the “Select” button is pressed. The driver loads the settings of the selected version and attempts to establish a connection using COM API.



Preparing Rhapsody Environment

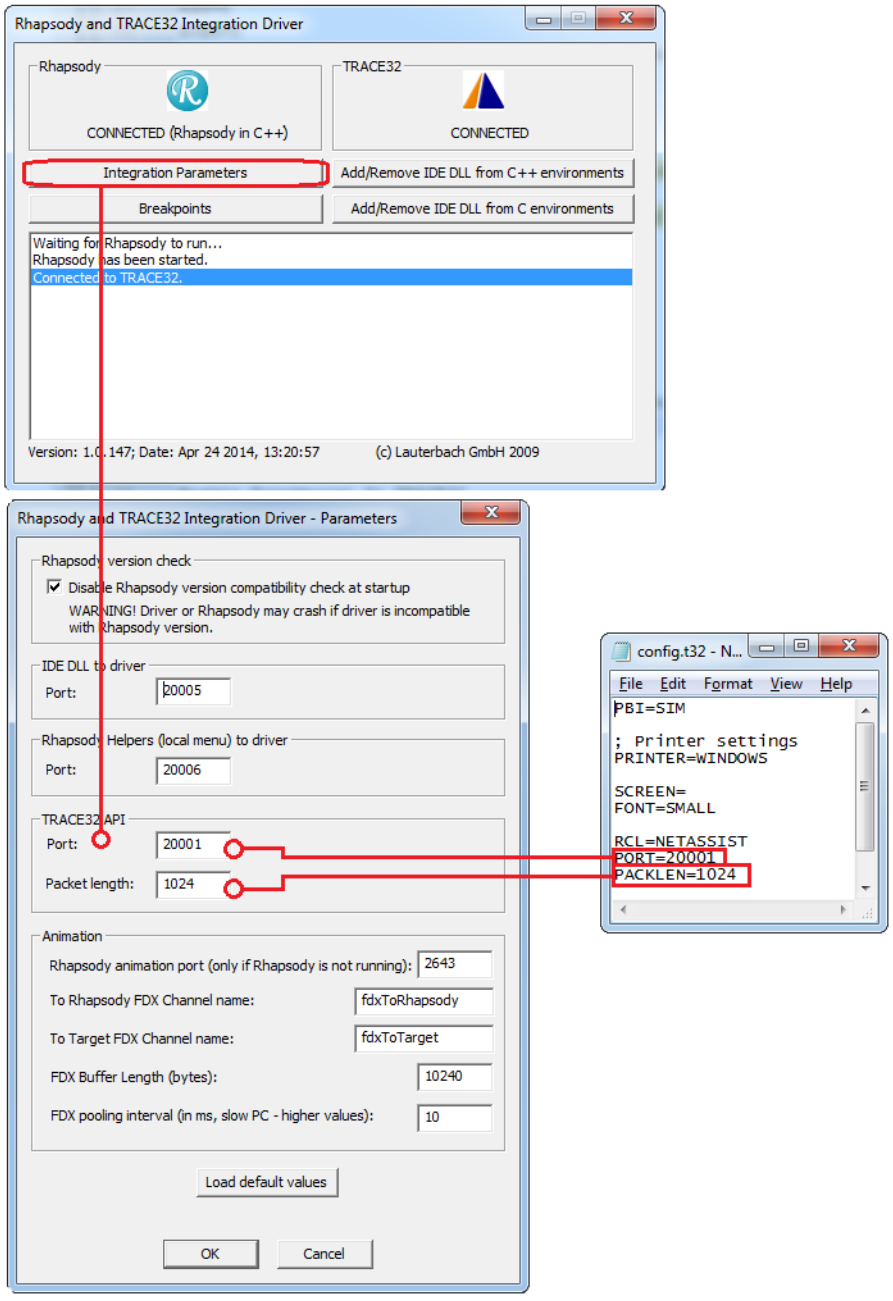
Rhapsody uses one of several Environments depending on the current value of the "CPP_CG::Configuration::Environment" or "C_CG::Configuration::Environment" property. To check the environment of your model in Rhapsody, expand the "Components" and "Configurations" of your model in the "Model View", right-click the configuration, and select **Features** from the pop-up menu. Select the **Properties** tab and "View All".

To enable the driver within Rhapsody for a specific environment, select "Add/Remove IDE DLL from C++ environments" in the driver dialog, then select the appropriate environments in the list. Close any open projects before changing these settings. Settings can be edited separately for each Rhapsody installation: select the version using the combo box near the top of the settings dialog:

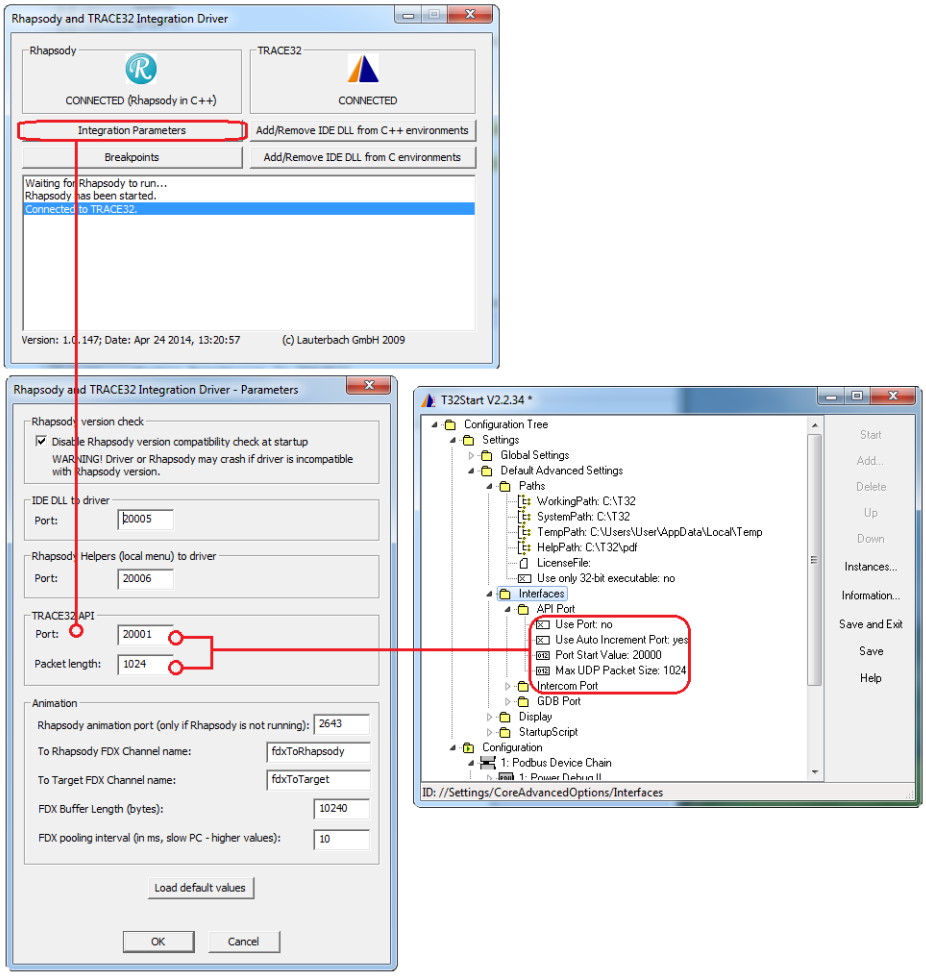


Preparing TRACE32

To enable the communication of the driver to TRACE32, some changes need to be made in **config.t32**. This file can be found in your TRACE32 Installation directory. Please keep one empty line before and after the section with RCL, PORT and PACKLEN.



Changes described above can also be made by using the T32Start utility provided with TRACE32:

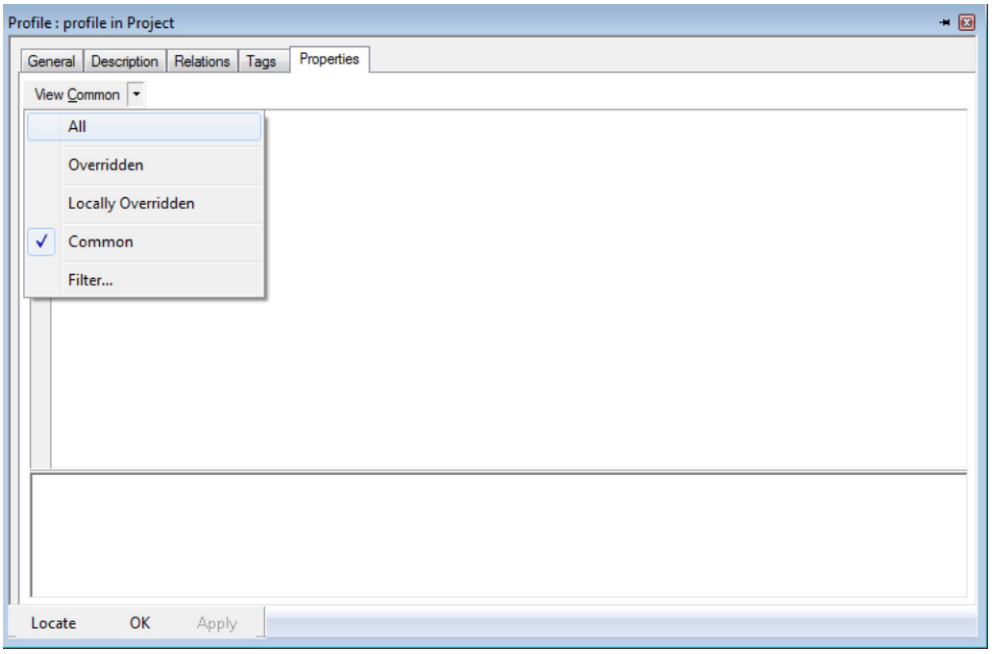


Rhapsody Helpers Configuration

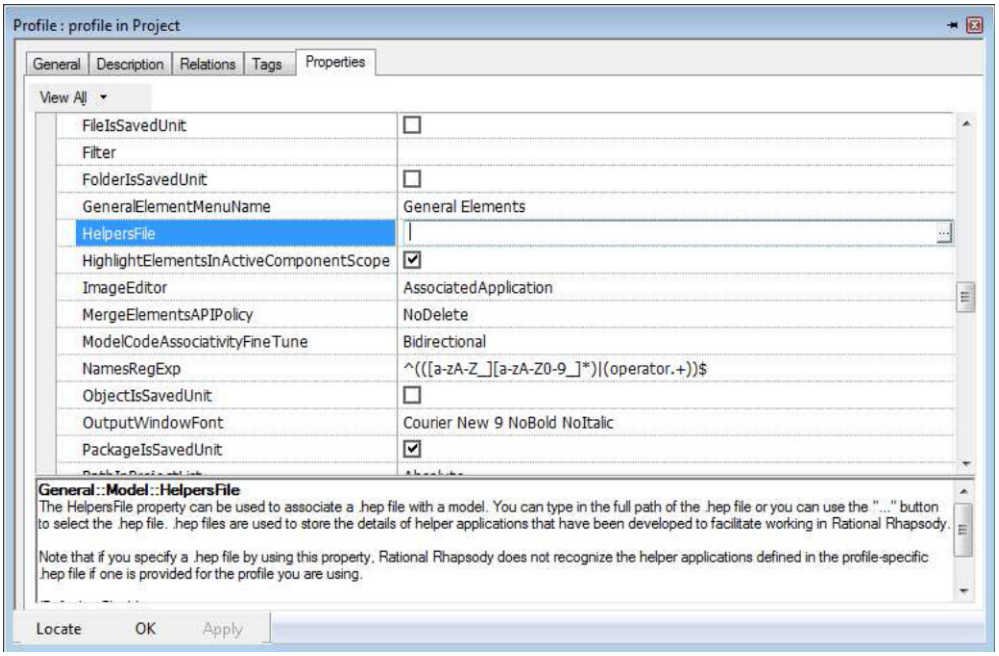
Helpers are the applications that can be added to Rhapsody menus. Integration Driver offers 10 helpers that can be selectively included/excluded from Rhapsody menus to fit your needs. The “helpers.hep” file contains all helpers. To generate new .hep and .sbs files, please run RhT32Driver.exe with the parameter "`-generate_profile`".

Adding helper file to project:

1. Start Rhapsody with “Run as administrator”.
2. If the profile does not yet exist, import the profile:
   File -> Add to Model..., select “%integratio patch%\profile.sbs”
   Optionally you can copy the .sbs and .hep file to your Rhapsody project.
3. Double clock on the profile (profile property windows)
4. Select “Properties”
5. In “View” context menu, select “All”



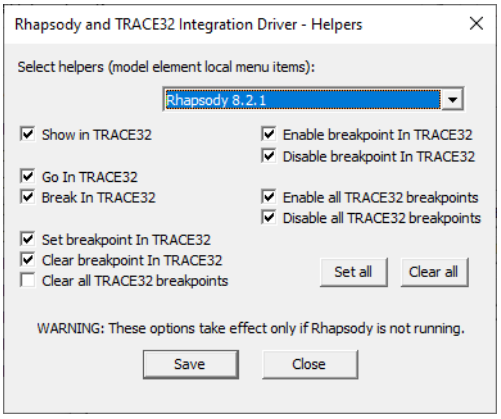
6. Select General -> Model -> HelpersFile and open “trace32_helpers.hep”



Helper settings can be edited separately for each Rhapsody installation detected in the systems.

Editing helper settings:

1. Click “Add/Remove Rhapsody helpers” button in Integration Driver dialog
2. Select Rhapsody version in combo box in Helpers dialog
3. Edit settings
4. Save settings by clicking “Save” button



Rebuilding OXF LangCpp Libraries for OSE, DIAB and PPC

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for OSE operating system, built with DIAB PPC compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf)

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangCpp** that are used during the build of OXF LangCpp libraries for OSE, DIAB and PPC environment.

aom\oseppcdiabaom.mak **omcom\oseppcdiabomcom.mak** **oxf\oseppcdiaboxf.mak** **tom\oseppcdiabtom.mak**	Makefiles that build specific part of OXF libraries
oseppcdiabbuild.mak	Makefile that builds entire OXF.
oseppcdiabpath.bat	Batch file that sets environment paths.
oseppcdiabbuild.bat	Batch file that starts build process by calling dmake
oxf\oseOS.cpp **oxf\oseOS.h**	OSAL classes implementation. oseConnectionPort is changed to use FDX routines instead of TCP/IP sockets. Original declaration and definition of oseConnectionPort is commented with preprocessor commands.
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation. t32fdxarm.c contains additional ARM specific FDX DCC communication routines, however this file is also needed by building process if target is other than ARM.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangCpp** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from the driver directory.

Before building, **oseppcdiabpath.bat** needs to be altered to match your environment (compiler and OSE location).

To rebuild the framework, execute batch files in a command console:

```
oseppcdiabpath.bat
oseppcdiabbuild.bat
```

Rebuilding OXF LangCpp libraries for eCos, GCC and PPC

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for eCos operating system using GCC powerpc-eabi compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf).

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangCpp** that are used during the build of OXF LangCpp libraries for eCos, GCC and PPC environment.

aom\ecosppcgccaom.mak **omcom\ecosppcgccomcom.mak** **oxf\ecosppcgccoxf.mak** **tom\ecosppcgcctom.mak**	Makefiles that build specific part of OXF libraries
ecosppcgccbuild.mak	Makefile that builds entire OXF.
ecosppcgccpath.bat	Batch file that sets environment paths.
ecosppcgccbuild.bat	Batch file that starts build process by calling GNU make
oxf\eCosOS.cpp **oxf\eCosOS.h**	OSAL classes implementation. ecosConnectionPort is changed to use FDX routines instead of TCP/IP sockets.
osconfig\eCos\omosconfig.h	eCos adaptor configuration file
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation. t32fdxarm.c contains additional ARM specific FDX DCC communication routines, however this file is also needed by building process if target is other than ARM.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangCpp** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from driver directory.

Before building, **ecosppcgccpath.bat** need to be altered to match your environment (compiler and eCos location).

To rebuild the framework, execute batch files in a command console:

```
ecosppcgccpath.bat
ecosppcgccbuild.bat
```

Rebuilding OXF LangCpp Libraries for eCos, GCC and ARM

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for eCos operating system with GCC arm-elf compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf). Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangCpp** that are used during build of OXF LangCpp libraries for eCos, GCC and ARM environment.

aom\ecosarmgccaom.mak **omcom\ecosarmgccomcom.mak** **oxf\ecosarmgccoxf.mak** **tom\ecosarmgcctom.mak**	Makefiles that build specific part of OXF libraries
ecosarmgccbuild.mak	Makefile that builds entire OXF.
ecosarmgccpath.bat	Batch file that sets environment paths.
ecosarmgccbuild.bat	Batch file that starts build process by calling GNU make
oxf\eCosOS.cpp **oxf\eCosOS.h**	OSAL classes implementation. ecosConnectionPort is changed to use FDX routines instead of TCP/IP sockets.
osconfig\eCos\omosconfig.h	eCos adaptor configuration file
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangCpp** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from driver directory.

Before building, **ecosarmgccpath.bat** need to be altered to match your environment (compiler and eCos location).

By default, **oxf\ecosarmgccoxf.mak** makefile is configured for ARM9 DCC communication (line 125-126). If the target is not ARM9, compile flags need to changed to one of possible values: **ARM7**, **ARM9**, **ARM11**, **XSCALE**, depending on your target. Simulators and some ARM processors don't support DCC communication. If such a target is used, please remove both compile flags (**-DT32_FDX_DCC** and **-DARM9**).

To rebuild framework, execute batch files in a command console:

```
ecosarmgccpath.bat
ecosarmgccbuild.bat
```

Rebuilding OXF LangC Libraries for OSE, DIAB and PPC

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for OSE operating system with DIAB PPC compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf)

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangC** that are used during build of OXF LangC libraries for OSE, DIAB and PPC environment.

aom\oseppcdiabaom.mak **omcom\oseppcdiabomcom.mak** **oxf\oseppcdiaboxf.mak**	Makefiles that build specific part of OXF libraries
oseppcdiabbuild.mak	Makefile that builds entire OXF.
oseppcdiabpath.bat	Batch file that sets environment paths.
oseppcdiabbuild.bat	Batch file that starts build process by calling dmake
oxf\RiCOSOSE.c **oxf\RiCOSOSE.h**	OSAL classes implementation. Original ConnectionPort is changed to use FDX routines instead of TCP/IP sockets.
osconfig\OSE\ricosconfig.h **osconfig\OSE\RiCOS.h**	OSE adaptor and configuration files
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation. t32fdxarm.c contains additional ARM specific FDX DCC communication routines, however this file is also needed by building process if target is other than ARM.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangC** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from driver directory.

Before building, **oseppcdiabpath.bat** need to be altered to match your environment (compiler and OSE location).

To rebuild framework, execute batch files in a command console:

```
oseppcdiabpath.bat
oseppcdiabbuild.bat
```

Rebuilding OXF LangC Libraries for eCos, GCC and PPC

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for eCos operating system with GCC powerpc-eabi compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf)

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangC** that are used during build of OXF LangC libraries for eCos, GCC and PPC environment.

aom\ecosppcgccaom.mak **omcom\ecosppcgccomcom.mak** **oxf\ecosppcgccoxf.mak**	Makefiles that build specific part of OXF libraries
ecosppcgccbuild.mak	Makefile that builds entire OXF.
ecosppcgccpath.bat	Batch file that sets environment paths.
ecosppcgccbuild.bat	Batch file that starts build process by calling GNU make
oxf\RiCOSeCos.c **oxf\RiCOSeCos.h**	OSAL classes implementation. Original ConnectionPort is changed to use FDX routines instead of TCP/IP sockets.
osconfig\eCos\ricosconfig.h **osconfig\eCos\RiCOS.h**	eCos adaptor and configuration files
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation. t32fdxarm.c contains additional ARM specific FDX DCC communication routines, however this file is also needed by building process if target is other than ARM.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangC** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from driver directory.

Before building, **ecosppcgccpath.bat** need to be altered to match your environment (compiler and eCos location).

To rebuild framework, execute batch files in a command console:

```
ecosppcgccpath.bat
ecosppcgccbuild.bat
```

Rebuilding OXF LangC Libraries for eCos, GCC and ARM

To use Integration Driver, OXF libraries need to be rebuilt.

This chapter shows how to rebuild OXF libraries for eCos operating system with GCC arm-elf compiler. To rebuild OXF for other OS, please refer to **"Rebuilding OXF Libraries for Other Environments"** (int_rhapsody_cpp.pdf). Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains all files with necessary changes needed for proper OXF building.

Following table lists all files from directory **<driver installation directory>\Share\LangC** that are used during build of OXF LangC libraries for eCos, GCC and ARM environment.

aom\ecosarmgccaom.mak **omcom\ecosarmgccomcom.mak** **oxf\ecosarmgccoxf.mak**	Makefiles that build specific part of OXF libraries
ecosarmgccbuild.mak	Makefile that builds entire OXF.
ecosarmgccpath.bat	Batch file that sets environment paths.
ecosarmgccbuild.bat	Batch file that starts build process by calling GNU make
oxf\RiCOSeCos.c **oxf\RiCOSeCos.h**	OSAL classes implementation. ecosConnectionPort is changed to use FDX routines instead of TCP/IP sockets.
osconfig\eCos\ricosconfig.h **osconfig\eCos\RiCOS.h**	eCos adaptor and configuration files
oxf\t32fdx.c **oxf\t32fdxarm.c** **oxf\t32fdx.h**	FDX routines implementation.

Compare above listed files with your existing files in **<RHAPSODY_ROOT>\Share\LangC** directory by using diff tool and alter your files appropriately. If some files don't exist, copy them from driver directory.

Before building, **ecosarmgccpath.bat** need to be altered to match your environment (compiler and eCos location).

By default, **oxf\ecosarmgccoxf.mak** makefile is configured for ARM9 DCC communication (line 125-126). If target is not ARM9, compile flags need to changed to one of possible values: **ARM7**, **ARM9**, **ARM11**, **XSCALE**, depending on your target. Simulators and some ARM processors don't support DCC communication. If such a target is used, please remove both compile flags (**-DT32_FDX_DCC** and **-DARM9**).

To rebuild framework, execute batch files in a command console:

```
ecosarmgccpath.bat
ecosarmgccbuild.bat
```

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains source files (in “Templates” directory) that allows TRACE32 FDX communication. These files are templates that can be used to implement FDX communication for any environment in Rhapsody.

Rhapsody in C++

LangCpp_templateOS.h contains ConnectionPort class definition template for TRACE32 and Rhapsody in C++ Integration.

To implement FDX communication instead of standard TCP/IP, please make following steps for your environment:

<OS_NAME> used below is a name of operating system for which you want to implement FDX communication, for example: 'ose', 'Vx', 'Nu', etc...

<OS_NAME>OS.h and **<OS_NAME>OS.cpp** files referred to in the following steps can be found in:

```
<RHAPSODY_ROOT>\Share\LangCpp\oxf
```

5. Comment **<OS_NAME>ConnectionPort** class definition in file **<OS_NAME>OS.h**

6. Copy content of file **LangCpp_templateOS.h** into **<OS_NAME>OS.h**, below previously commented source code.

7. Comment following method definitions and variable initializations in **<OS_NAME>OS.cpp**:

```
char* <OS_NAME>ConnectionPort::m_Buf = NULL
int <OS_NAME>ConnectionPort::m_BufSize = 0
<OS_NAME>ConnectionPort::<OS_NAME>ConnectionPort()
<OS_NAME>ConnectionPort::~<OS_NAME>ConnectionPort()
void <OS_NAME>ConnectionPort::readFromSockLoop(<OS_NAME>ConnectionPort *me)
int <OS_NAME>ConnectionPort::Connect(const char* SocketAddress = NULL, unsigned int nSocketPort = 0)
int <OS_NAME>ConnectionPort::Send(OMSData *m)
```

   Make sure that there are no other not commented methods of class **<OS_NAME>ConnectionPort**. If there are any, comment them too.

8. Copy content of file **LangCpp_templateOS.cpp** below previously commented source code.

9. Replace all occurences of **T32_INT_OS_NAME** with your OS name in both **<OS_NAME>OS.h** and **<OS_NAME>OS.cpp** files.

10. **ConnectionPort** class can report errors in some situations. Error descriptions are simple strings. To implement handling of these errors please fill method **ErrorNotification** in **<OS_NAME>ConnectionPort** class with code appropriate for your system (for example fprintf(stderr,message)). Filling this function is not obligatory.

11. <OS_NAME>OS.h with new <OS_NAME>ConnectionPort class definition copied in step 2 contains following preprocessor define:

```
#define T32_INT_READFROMSOCKLOOP_THREAD_CREATION_PARAMETERS ...
```

    Replace **...** in this define with **<OS_NAME>Thread()** constructor parameters. These parameters can be found in **<OS_NAME>ConnectionPort::Connect** method that you commented in **step 3**, nearly at the end of that method.

12. **<OS_NAME>OS.h** with new **<OS_NAME>ConnectionPort** class definition copied in **step 2** contains following preprocessor define:

```
#define T32_INT_DELAY_NAME(miliseconds) ...(miliseconds)
```

    Replace **...** in this define with delay function name specific for your OS.

13. Copy **t32fdx.c** and **t32fdx.h** files to **<RHAPSODY_ROOT>\Share\LangCpp\oxf** directory and add them to your OXF compilation/linking makefile. These files can be found in driver installation directory (usually C:\Program Files\...).

14. Add:

```
#include "t32fdx.h"
```

to the beginning of **<OS_NAME>OS.cpp** file. The best place is immediately after:

```
#ifdef _OMINSTRUMENT
```

Rhapsody in C

LangC_templateOS.h contains ConnectionPort struct definition template for TRACE32 and Rhapsody in C Integration.

To implement FDX communication instead of standard TCP/IP please make following steps for your environment:

<OS_NAME> used below is a name of operating system for which you want to implement FDX communication, for example: 'ose', 'Vx', 'Nu', etc...

RiCOS<OS_NAME>OS.h and **RiCOS<OS_NAME>OS.c** files referred to in the following steps can be found in:

<RHAPSODY_ROOT>\Share\LangC\oxf

1. Comment **RiC<OS_NAME>ConnectionPort** struct definition together with typedef below in file **RiCOS<OS_NAME>OS.h**

2. Copy content of file **LangC_templateOS.h** into **RiCOS<OS_NAME>OS.h**, below previously commented source code.

3. Comment following function definitions in **RiCOS<OS_NAME>OS.c**:

RiCBoolean RiCOSConnectionPort_init(RiCOSConnectionPort * const me)
void RiCOSConnectionPort_cleanup(RiCOSConnectionPort * const me)
RiCOSConnectionPort * RiCOSConnectionPort_create()
void RiCOSConnectionPort_destroy(RiCOSConnectionPort * const me)
static void readFromSockLoop(RiCOSConnectionPort *me);
RiCOSResult RiCOSConnectionPort_Connect(RiCOSConnectionPort * const me,const char* const SocketAddress,unsigned int nSocketPort)
RiCOSResult RiCOSConnectionPort_Send(RiCOSConnectionPort * const me, struct RiCSData *m)
RiCBoolean RiCOSConnectionPort_SetDispatcher(RiCOSConnectionPort * const me, RiCOS_dispatchfunc dispfunc)
static void readFromSockLoop(RiCOSConnectionPort *me)

   Make sure that there are no other not commented function **RiCOSConnectionPort_...** If there are any, comment them too.

4. Copy content of file **LangC_templateOS.c** below previously commented source code.

5. Replace all occurences of **T32_INT_OS_NAME** with your OS name in **RiCOS<OS_NAME>OS.h** file.

6. **ConnectionPort** functions can report errors in some situations. Error descriptions are simple strings. To implement handling of these errors please fill function **RiCOSConnectionPort_ErrorNotification(char * message)** with code appropriate for your system (for example fprintf(stderr,message)). Filling this function is not obligatory.

7. **RiCOS<OS_NAME>OS.h** with new **RiC<OS_NAME>ConnectionPort** struct definition copied in **step 2** contains following preprocessor define:

```
#define T32_INT_READFROMSOCKLOOP_TASK_CREATION_PARAMETERS ...
```

   Replace **...** in this define with **RiCOSTask_create()** function parameters. These parameters can be found in **RiCOSConnectionPort_Connect** function that you commented in **step 3**, nearly at the end of that function.

8. **RiCOS<OS_NAME>OS.h** with new **RiC<OS_NAME>ConnectionPort** struct definition copied in **step 2** contains following preprocessor define:

```
#define T32_INT_DELAY_NAME(miliseconds) ...(miliseconds)
```

   Replace **...** in this define with delay function name specific for your OS.

9. Copy **t32dfx.c** and **t32fdx.h** files to **<RHAPSODY_ROOT>\Share\LangC\oxf** directory and add them to your OXF compilation/linking makefile. These files can be found in driver installation directory (usually C:\Program Files\...)

10. Add:

```
#include "t32fdx.h"
```

to the beginning of **RiCOS<OS_NAME>OS.c** file. The best place is immediately after:

```
#ifdef _OMINSTRUMENT
```

Preparing C++ Application for Animation

This section describes how to prepare a C++ application for animation in OSE, DIAB and PPC

Before you can begin animation, you must follow these steps:

1. Set the **CPP_CG::Configuration::Environment** property to environment you are working with. For purposes of this manual environment is **OsePpcDiab**.

2. Set the Instrumentation flag for the configuration to **Animation**, and optionally the animation scope using the **Advanced** button.

   a. In the browser, expand the component, right-click the configuration, and select **Features** from the pop-up menu.

   b. Select the Settings tab.

   c. Set the **Instrumentation** field to **Animation**.

   d. To instrument operations and set a finer scope on the instrumentation, click the **Advanced** button.

3. For OSE, change heap share mode to yes in file:

   <RHAPSODY_ROOT>\Share\MakeTmp\OseDiabPPCconf.mk

```
INCLUDE_OSE_HEAP*= yes

.IF $(INCLUDE_OSE_HEAP) == yes
  HEAP_SHARE_MODE *= yes# [ yes | no ]
  HEAP_SIZE*= SIZE_276K
.END
```

4. Generate code for the configuration. If you don't have a properly configured Embedded File System in OSE, make sure that generated code doesn't have any operations which are using file system (stdout, stdin, cout, ...). It there are any, comment them out.

5. If DIAB PPC compiler is used together with OSE (like in this manual), **-Xinit-section=2** need to be added to **CFLAGS** and **CXXFLAGS** in <OSE_ROOT>\powerpc\makefiles\compilers\diab.mk.

```
CFLAGS+= $(FLAGS)
CFLAGS+= -Xlint
CFLAGS+= -Xinit-section=2

...

CXXFLAGS+= $(FLAGS)
CXXFLAGS+= -Xinit-section=2
```

6. Build application using Code->Build.

Driver installation directory (usually: C:\Program Files\Lauterbach GmbH\Rhapsody 7.x and TRACE32 Integration Driver) contains following files with environments definition:

Properties\siteC++.prp	eCosARM and eCosPPC environments definition for C++ language
Properties\siteC.prp	eCosARM, eCosPPC and OsePPCDiab environments definition for C language

Please refer to documentation provided by Rhapsody: **"RTOS Adapter Guide - chapter: The Deployment Environment - Adapting Rhapsody to a New RTOS - Step 5: Creating Properties for a new RTOS"**. This documentation explains in detail the meaning and use of site<language>.prp files.

In case of using OsePPCDiab environment for C language, all steps listed in **"Preparing C++ Application for Animation"** (int_rhapsody_cpp.pdf) apply, except step 1, where CPP_CG is changed to C_CG.

Environments listed in the table below require additional batch files in **<RHAPSODY_ROOT>\Share\etc** directory. These files are used to invoke Rhapsody application build and can be found in **<Driver Installation Directory>\Share\etc**:

oseppcdiabmake.bat	OSE on PPC
ecosppcgccmake.bat	eCos on PPC
ecosarmgccmake.bat	eCos on ARM

All above batch files need to be altered to match your environment (compiler and eCos/OSE location).

Downloading Application Image to Target

To download application from Rhapsody to target, Integration Driver requires a practice script **download.cmm** to be provided in directory that contains linked application image. For Ping-Pong sample this directory is **<RHAPSODY_ROOT>\Samples\CppSamples\PingPong\AnimComponent\AnimConfig**.

Driver passes image path as parameter &image_path in download script.

Download scripts for LangCpp and LangC can be found at **<Driver Installation Directory>\Download**:

LangCpp\download_OSE_FADS860.cmm
LangCpp\download_ECOS_FADS860.cmm
LangCpp\download_ECOS_ARM9Excalibur.cmm
LangC\download_OSE_FADS860.cmm
LangC\download_ECOS_FADS860.cmm
LangC\download_ECOS_ARM9Excalibur.cmm

Each of download script contains section **"Target initialization"**. Please alter this section appropriately for your target.

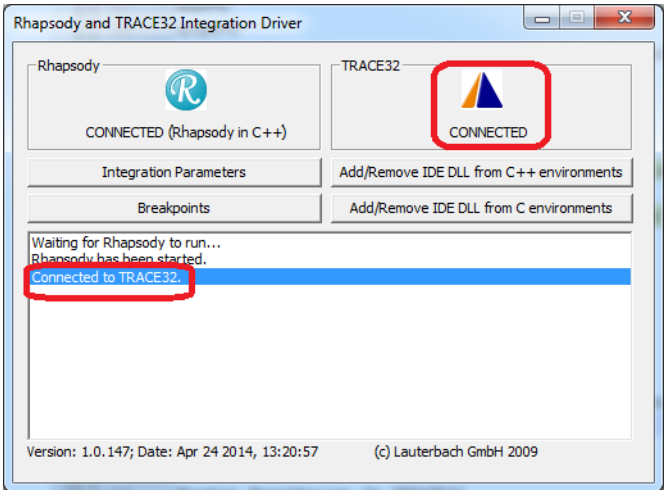
Each download script also contains a **&dcc_fdx_mode** variable, that sets appropriate FDX communication mode: **DCC** (&dcc_fdx_mode=1) or **BUFFERE/BUFFERC** (&dcc_fdx_mode=0). **DCC** communication can be used only on targets that support Debug Communication Channel (most ARMs). **BUFFERC** can be used on other targets (not supporting DCC) and TRACE32 simulators. **BUFFERE** can be used only on simulators. FDX communication mode has to match mode used during OXF building. See **"Rebuilding OXF LangCpp Libraries for eCos, GCC and ARM"** (int_rhapsody_cpp.pdf) and **"Rebuilding OXF LangC Libraries for eCos, GCC and ARM"** (int_rhapsody_cpp.pdf) for details.

After preparing download script, please follow below steps to download application to target:

1. Run Integration Driver.

2. Run TRACE32 Simulator or ICD.

3. From Rhapsody menu select **Code->Target->Connect**. If connection is estabilished, Rhapsody status bar should contain “Connected” message. Driver indicates succesfull connection by message in log window and “CONNECTED” status under TRACE32 icon. If connecting failed try to connect again.



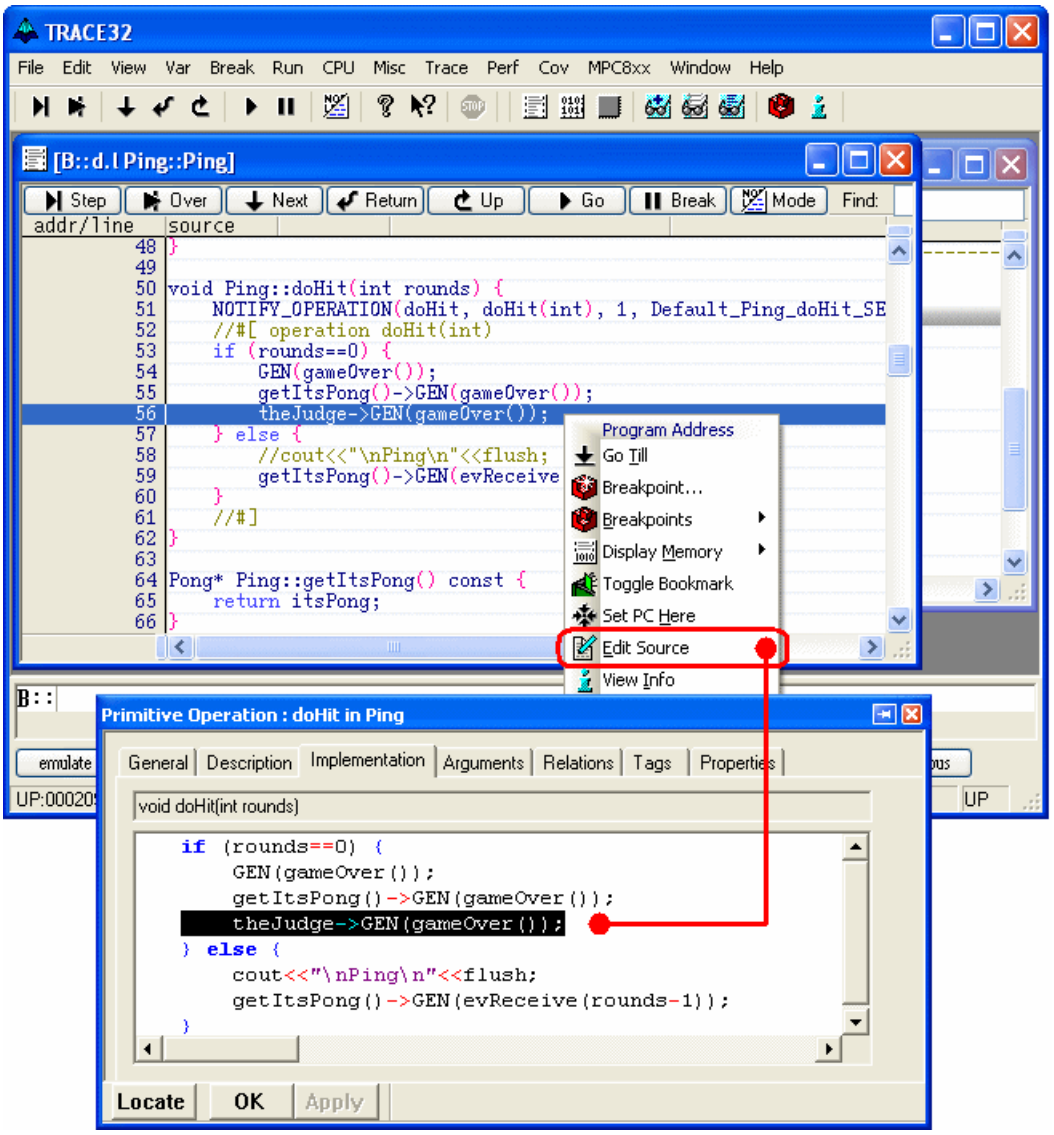
4. In Rhapsody select **Code->Target->Download** to download application image. Successfull downloading is indicated by “Download complete“ message in Rhapsody status bar and by availability of **Code->Run** option in Rhapsody menu.

Integration Features

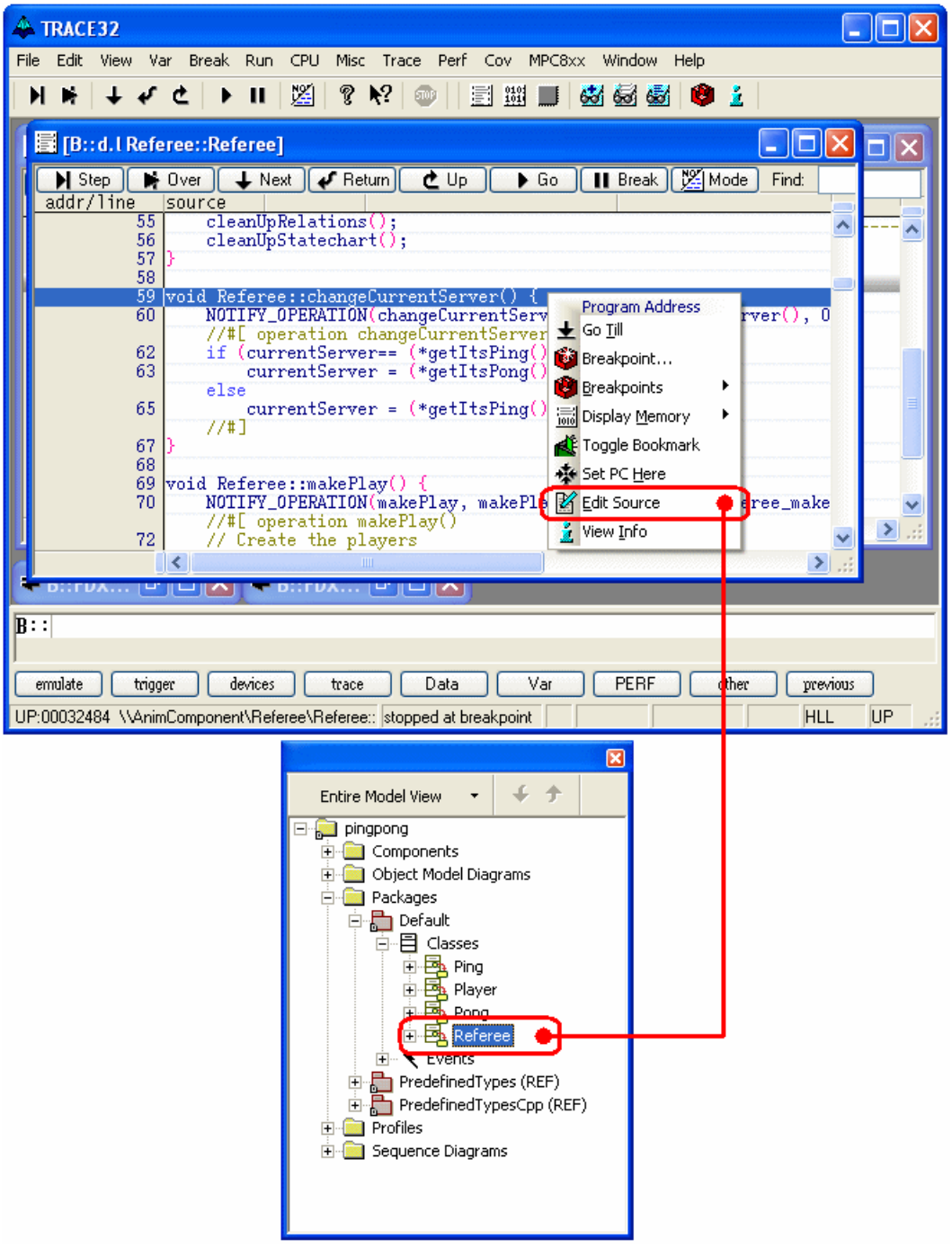
General purpose of Integration Driver is to allow animation of application in target/simulator without performing TCP/IP connection link beetween Rhapsody and instrumented application. Additionally Integration Driver offers some features which can be helpfull during debbuging Rhapsody created applications using TRACE32.

To use this feature, additional PRACTICE command - **SETUP.EDITEXT ON** - need to be executed in download script. Please refer to **"Downloading Application Image to Target"** (int_rhapsody_cpp.pdf).

To edit source in Rhapsody, right-click source line of function (other than constructor and destructor) in TRACE32 and select **Edit Source**.

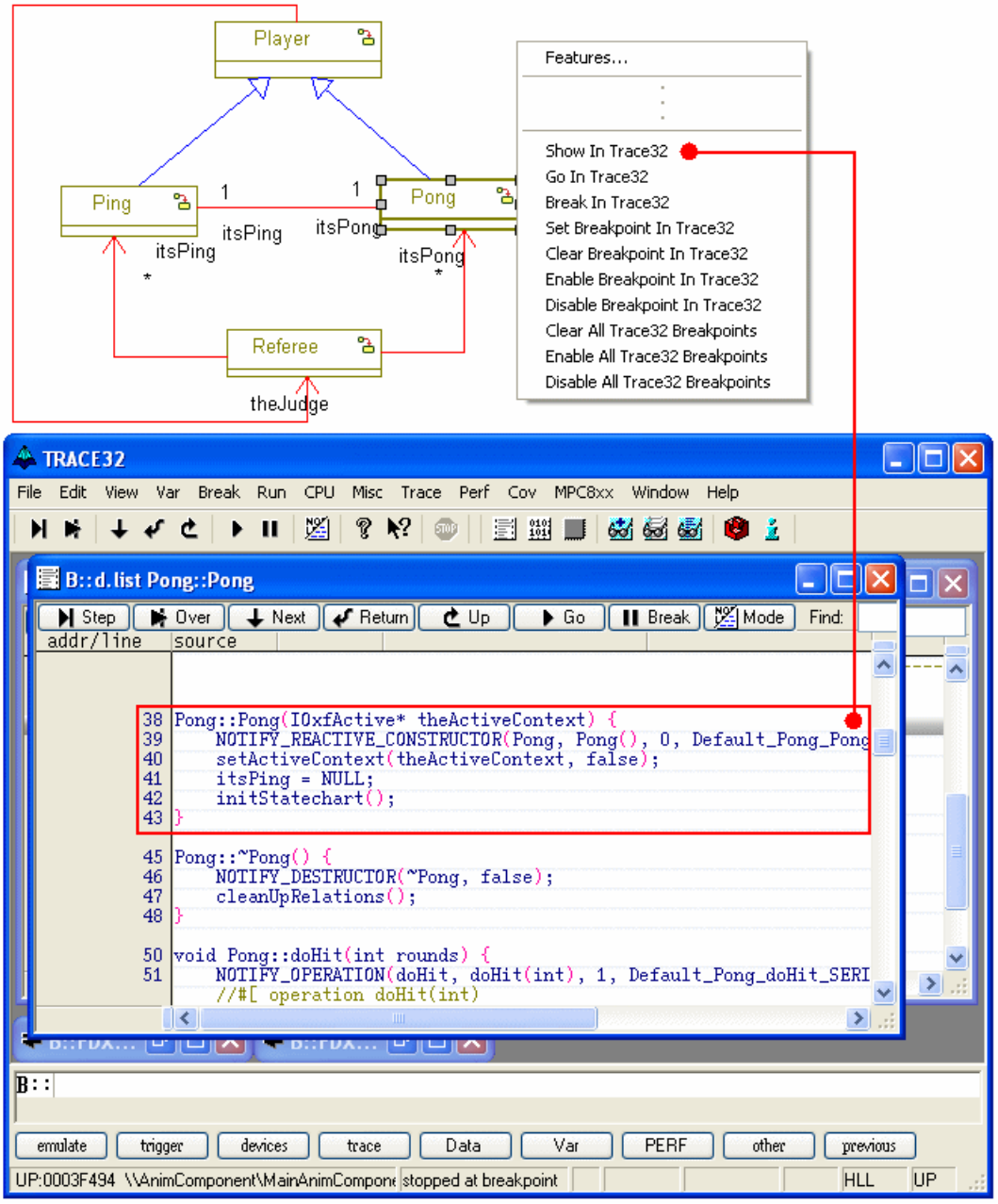


To locate model element in Rhapsody project browser, right-click on function header (or on any line of constructor or destructor) in TRACE32 and select **Edit Source**.



Locating Source in TRACE32

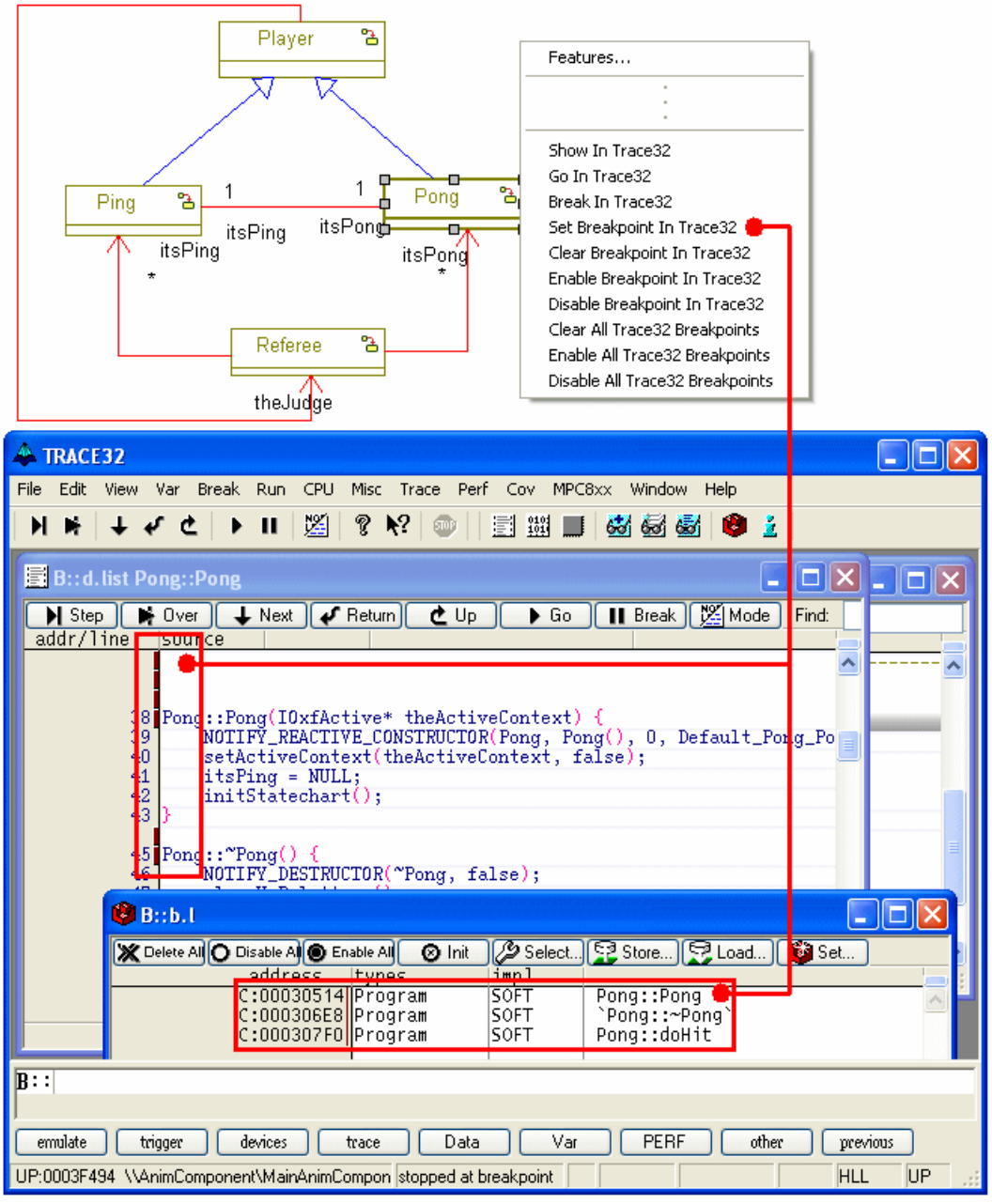
To locate source in TRACE32 right-click on model element in diagram or project browser and select **Show In** TRACE32. The selected method is displayed in the **Data.List** window. For classes TRACE32 lists constructor source.



Breakpoints

Integration Driver allows setting/clearing/disabling/enabling breakpoints directly from Rhapsody by using right-click menu.

To set, clear, enable or disable a breapoint right-click on model element in diagram or project browser and select appropriate option. Refer to **"Supported Model Elements"** (int_rhapsody_cpp.pdf) to see details about model elements support.



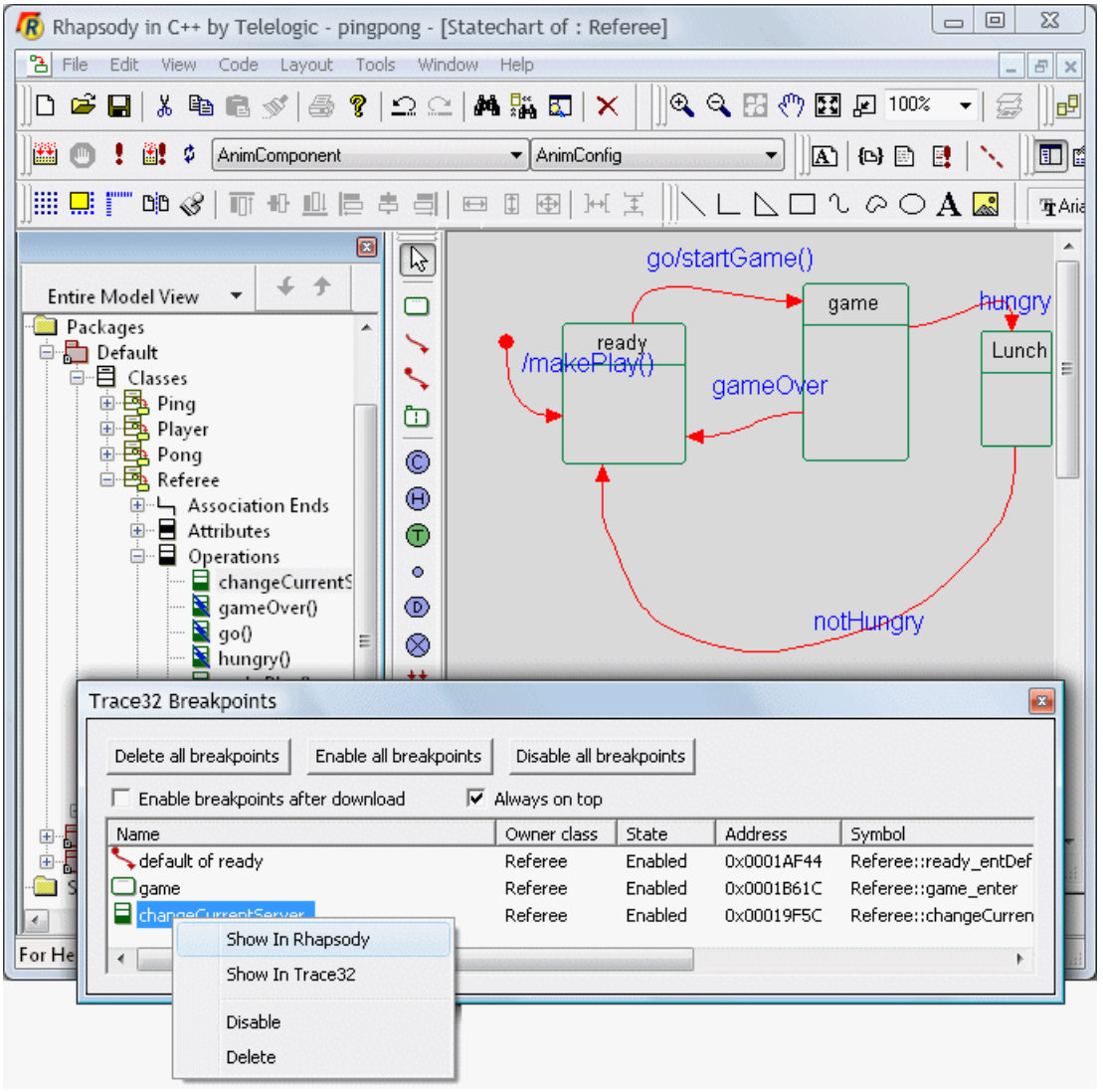
To manage breakpoints, click “Breakpoints“ button in Integration Driver dialog. Tool window “TRACE32 Breakpoints“ allows deleting, enabling and disabling each breakpoint that was set in Rhapsody model.

To locate model element in Rhapsody, right-click on breakpoint in the list and select “Show In Rhapsody”. Additionaly, model element can be located by double-clicking on breakpoint.

To open source code for model element in TRACE32, right-click on breakpoint and select “Show in TRACE32”.

If application is downloaded to target (see **“Downloading Application Image to Target”** (int_rhapsody_cpp.pdf)), driver disables all breakpoints. Please check “Enable breakpoints after download“ to enable breakpoints after download is finished.

NOTE: Symbols used by breakpoints are evaluated only once during set. If statechart implementation is changed, breakpoints need to be deleted and set again to evaluate new symbols.



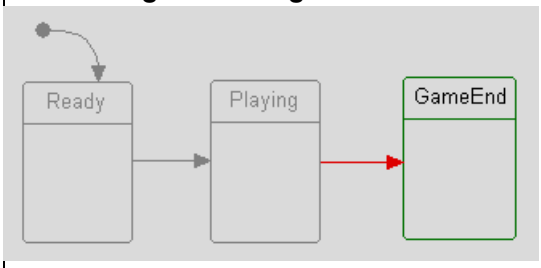
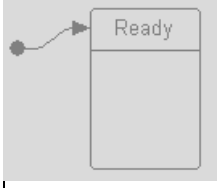
Go and Break in TRACE32

By using **Go In TRACE32** and **Break in TRACE32** you can control the execution of animated application separately from Rhapsody. **Go In TRACE32** is a way to resume execution after stopping at breakpoint

Supported Model Elements

The table below shows supported model elements and their mapping into C++/C source code symbols in TRACE32. These mapping is used by **"Locating Source in TRACE32"** (int_rhapsody_cpp.pdf) and **"Breakpoints"** (int_rhapsody_cpp.pdf) features of this integration.

Class 	Example: **Player** Symbols of following elements are used: constructor, destructor, primitive and triggered operations. See below for examples of each element.
Constructor 	Example: **Player::Player** C++: **Player::Player** C, public: **Player_Init** C, private: **\Player\Init**
Destructor 	Example: **Player::~Player** C++: **Player::~Player** C, public: **Player_Cleanup** C, private: **\Player\Cleanup**
Primitive Operation 	Example: **Player::doHit** C++: **Player::doHit** C, public: **Player_doHit** C, private: **\Player\doHit**
Triggered Operation 	Example: **Player::catch** C++: **Player::catch** C, public: **Player_catch** C, private: **\Player\catch**
Event 	Example: **serve** C++: **serve::serve** C: **serve_Init**

State 	**NOTE:** If statechart implementation is set to “Flat” for current configuration, change value of property “CG::Class::ComplexityForInlining” to “0”, to generate method calls instead of inlining. Example: Class **Ping** inherits from class **Player**. States **Ready** and **Playing** are defined in class **Player**, while state **GameEnd** is defined in class **Ping**. Class **Ping** has below statechart:  Driver will use below symbols for state **Playing**: • C++ (reusable): **Player_Playing::enterState** • C++ (flat): **Ping::Playing_enter** • C: **Ping_Playing_enter** Driver will use below symbols for state **GameEnd**: • C++ (reusable): **Ping_GameEnd::enterState** • C++ (flat): **Ping::GameEnd_enter** • C: **Ping_GameEnd_enter**
Default Transition 	**NOTE:** If statechart implementation is set to “Flat” for current configuration, change value of property “CG::Class::ComplexityForInlining” to “0”, to generate method calls instead of inlining. Example: Class **Ping** inherits from class **Player**. State **Playing** is defined in class **Player**. Class **Ping** has below statechart:  Driver will use below symbols for default transition: • C++ (reusable): **Player_Ready::entDef** • C++ (flat): **Ping::Ready_entDef** • C: **\Ping\Ready_entDef**

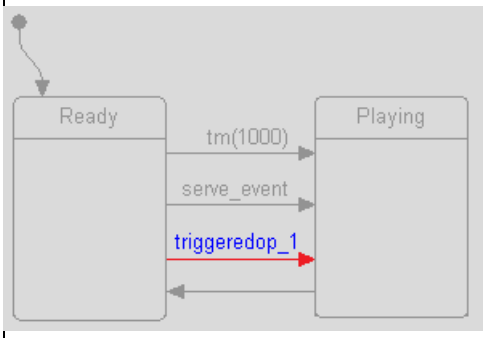
Transition

NOTE: If statechart implementation is set to “Flat” for current configuration, change value of property “CG::Class::ComplexityForInlining” to “0”, to generate method calls instead of inlining.

Driver supports following types of transitions in both Flat and Reusable Statechart implementation:

- transition without trigger
- transition with trigger: timeout, event or triggered operation

Example: Class **Ping** inherits from class **Player**. Class **Ping** has below statechart:



Driver will use following symbols for each transition:

- tm(1000)

  C++ (reusable): **Player::ReadyTakeTimeout**

  C++ (flat): **Ping::ReadyTakeTimeout**

  C: **Ping_ReadyTakeTimeout**

- serve_event

  C++ (reusable): **Player::ReadyTakeserve**

  C++ (flat): **Ping::ReadyTakeserve**

  C: **Ping_ReadyTakeserve**

- triggeredop_1

  C++: **Ping::ReadyTaketriggeredop_1_Ping_Event**

  C: **Ping_ReadyTaketriggeredop_1_Ping_Event**

  NOTE: last “Ping” in symbol is an owner of triggered operation.

- no trigger

  C++ (reusable): **Player::PlayingTakeNull**

  C++ (flat): **Ping::PlayingTakeNull**

  C: **Ping_PlayingTakeNull**