User-defined Data Types in Rhapsody

Techletter Nr 5

WILLERT.

Content

Why ?

Using your own datatypes is a requirement that most users will have sooner or later. Rhapsody's standard datatypes are the types that are defined in the underlying language.
Multiple sources, among them MISRA,tell us to use datatypes that are interchangeable between different hardware platforms.
Also the definition of user defined types forces the compiler to check for failures in the usage of variables and functions

This Techletter explains what options there are to define your own data types in Rhapsody and how to use them efficiently in your modeling.
It explains how to define properties so you can select your own types, and what properties to use to get the right code generated.

'C' and also 'C++' have various types and type modifiers/specifiers, this Techletter explains how to use them in a Rhapsody model.

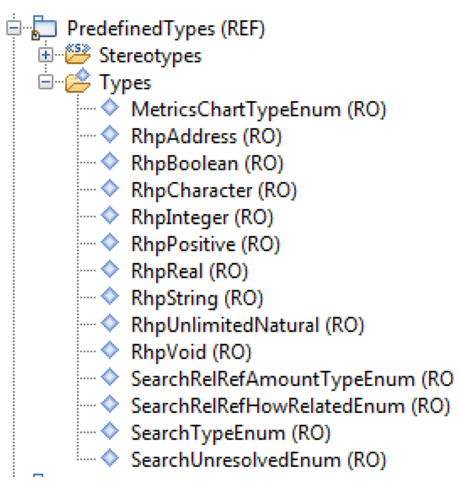
Rhapsody's own types

Rhapsody comes with type definitions "Out-Of-The-Box". They are the standard types that are defined in the Code generation Language.

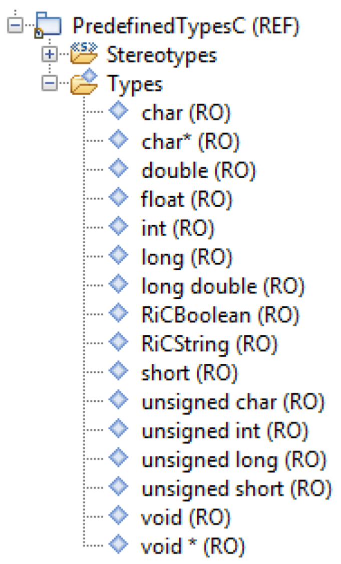
The picture on the far right shows the types for Rhapsody in 'C'.
These are all standard 'C' types except for RiCBoolean and RiCString. Since 'C' does not know a boolean type (like 'C++' does) it is defined in Rhapsody to allow use of it in the model. Also defines are RICTRUE and RICFALSE. RiCString is a 'C'-class that handles strings.

The Types in the example on the immediate right are language independent types used for e.g. SysML.

These can be useful because the underlying Framework should take care of a correct implementation but for embedded environments, other types are needed.



- PredefinedTypes (REF)
  - Stereotypes
  - Types
    - MetricsChartTypeEnum (RO)
    - RhpAddress (RO)
    - RhpBoolean (RO)
    - RhpCharacter (RO)
    - RhpInteger (RO)
    - RhpPositive (RO)
    - RhpReal (RO)
    - RhpString (RO)
    - RhpUnlimitedNatural (RO)
    - RhpVoid (RO)
    - SearchRelRefAmountTypeEnum (RO
    - SearchRelRefHowRelatedEnum (RO)
    - SearchTypeEnum (RO)
    - SearchUnresolvedEnum (RO)



- PredefinedTypesC (REF)
  - Stereotypes
  - Types
    - char (RO)
    - char* (RO)
    - double (RO)
    - float (RO)
    - int (RO)
    - long (RO)
    - long double (RO)
    - RiCBoolean (RO)
    - RiCString (RO)
    - short (RO)
    - unsigned char (RO)
    - unsigned int (RO)
    - unsigned long (RO)
    - unsigned short (RO)
    - void (RO)
    - void * (RO)

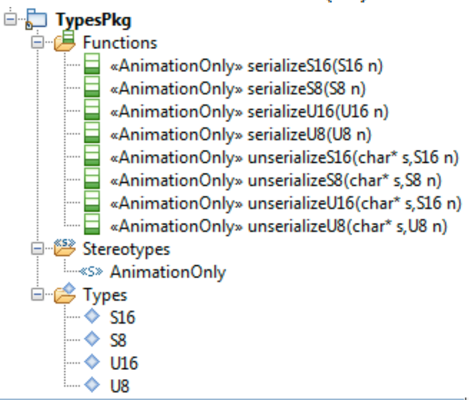
Create your own types

How to create your own types? That is actually quite easy. Rhapsody knows "Type" as a base element. So a right klick on a package, Class or Object give you the possibility to add a type. Types in Classes and/or Objects are local, they contain the name of the Class/Object and are generated in the class.c file. (unless they are private off course, then they are named without Classname and in the class.h file) Defining types in a package makes most sense, they are generated in the package file and they are generated with the correct name.

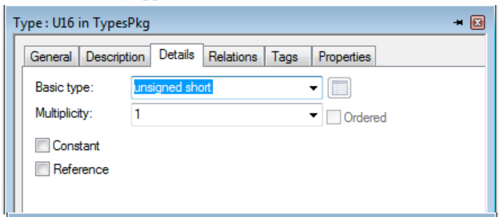
Best is to use a separate package for your user-defined types. Like in this example we use "TypesPkg" in this package we define all types we need. We just used four in the example but it should make clear how the system works.

The Type in Rhapsody allows a lot of different inputs, you can create enums, structs, unions and typedefs, as shown here.

The U16 (Unsigned 16 bit) Type is defined as an unsigned short which, on this system, is an unsigned 16bit type.



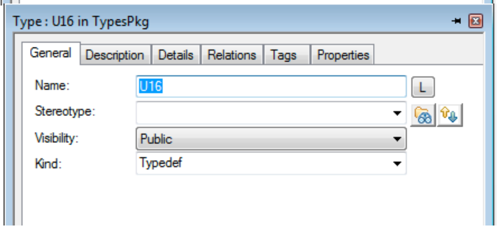
- TypesPkg
  - Functions
    - «AnimationOnly» serializeS16(S16 n)
    - «AnimationOnly» serializeS8(S8 n)
    - «AnimationOnly» serializeU16(U16 n)
    - «AnimationOnly» serializeU8(U8 n)
    - «AnimationOnly» unserializeS16(char* s,S16 n)
    - «AnimationOnly» unserializeS8(char* s,S8 n)
    - «AnimationOnly» unserializeU16(char* s,S16 n)
    - «AnimationOnly» unserializeU8(char* s,U8 n)
  - Stereotypes
    - «S» AnimationOnly
  - Types
    - S16
    - S8
    - U16
    - U8



Type : U16 in TypesPkg

General | Description | Details | Relations | Tags | Properties

Basic type: unsigned short
Multiplicity: 1
☐ Ordered
☐ Constant
☐ Reference



Type : U16 in TypesPkg

General | Description | Details | Relations | Tags | Properties

Name: U16
Stereotype:
Visibility: Public
Kind: Typedef

What about animation?

Rhapsody developer edition has a feature called "Animation". When selected the generated code will be animated. The generated application will then, after start, communicate with Rhapsody via a TCP/IP connection. The application can be controlled by the user and the application will report back information to Rhapsody that can be showed by the user. Sequence diagrams, Statecharts can be animated and the content of instances can be viewed and changed.

Viewing and changing variables that have user defined types is difficult for Rhapsody since it does not have information about the exact meaning of the type's content.

Therefor, functions must be defined that are able to represent the content and to parse a string that contains the content of a variable. This is called serialize<TypeName> and unserialize<TypeName>. Serialize receives an argument from the used type and returns a character string (Use malloc if necessary!), unserialize receives a string and returns a type.

A small trick will prevent these functions to be used when Animation is not selected. This is done with the <<AnimationOnly>> stereotype. It uses a define (OMINSTRUMENTATION) that is set by Rhapsody when Animation is used.

This stereotype defines a prologue and an epilogue, using properties:

C_CG::Operation::ImplementationProlog: `#ifdef OMINSTRUMENTATION`
C_CG::Operation::ImplementationEpilog: `#endif`
This makes sure that the extra functions are only used when animation is switched on.

Integrate your datatypes in Rhapsody

You can tell Rhapsody to use our data types. You can even tell Rhapsody to use _only_ your datatypes. There are 2 properties that take care of that:

```
General::Model::CommonTypes
```

If this property is empty, Rhapsody will use the standard data types in PredefinedTypes(C). Using $ALL will use all datatypes that are defined in your Rhapsody model. using (qualified!) Package names that contain type definitions will use only these.
You can use multiple Types Packages, fill them comma separated in the Property. You can also use the predefined Packages here, just treat them as a normal types package.

```
General::Model::DefaultType
```

This will define the default datatype that is filled in by Rhapsody when you create a new attribute or something else that uses a type. It should be qualified with the Name of the package it is defined in. (e.g. TypesPkg::uint32)

Ditch the standard Types

When the standard Rhapsody Types are no longer needed they can be deleted. Unfortunately, this is impossible, deleting has no effect. There is a property that prevents Rhapsody from displaying them:

```
Browser::Settings::ShowPredefinedPackage
```

When you use the properties that include your own data types and leave out the standard types (and don't use the $All directive) then the standard Rhapsody types will disappear. Only the user defined types will appear in the type selection menu's.

What about structs and unions?

They should not be used for attributes or variables. If you have the need to use a struct you actually need a class. But in the embedded world it is always possible that there are reasons to implement structs or unions, like saving memory space, using bitfields or implementing complex communication protocols. Rhapsody allows the use of structs and or unions as type in attributes (just select "struct" or "union") or you can create your own types that use structs or unions.

What about bitfields?

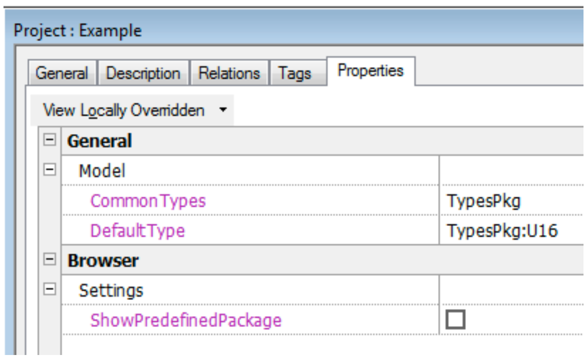
Bitfields can be used, you need to define them using properties. Define an attribute and set the following property:
`C_CG::Attribute::BitField`
to the number of bits you want to use. The attribute must be part of a struct off-course.

What about pointers, volatile and const?

Pointers are easy, there is a "Reference" Field in the Type definition. There is, however, one specific thing to take care of. Rhapsody generates a pointer for every variable used as an argument. Even for enumerators although that does not make much sense.

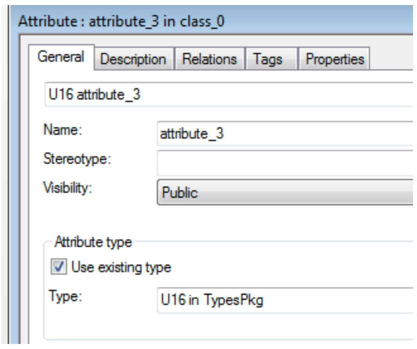
This can be changed in some properties that control the code generation behavior of types: IN, INOUT and OUT. Also the return value can be set.

The properties are: `C_CG::Type::In`, `C_CG::Type::InOut`, `C_CG::Type::Out` and `C_CG::Type::ReturnType`. Use them to define how Rhapsody generates code for parameters that use user defined types.



General	
Model	
CommonTypes	TypesPkg
DefaultType	TypesPkg:U16
Browser	
Settings	
ShowPredefinedPackage	☐

Const is a Field in the attribute features, just tick it and the variable is set to const.
Volatile is handled by a property:
`C_CG::Attribute::IsVolatile`
Should be set to TRUE, then a volatile keyword is generated.



U16 attribute_3

Name: attribute_3
Stereotype:
Visibility: Public

Attribute type
☑ Use existing type
Type: U16 in TypesPkg

That's it!

More releases of the

NEWSLETTER / TECHLETTER

you can find on:

willert.de/Newsletter
willert.de/Techletter

Author:
WALTER VAN DER HEIDEN

Publisher:
WILLERT SOFTWARE TOOLS GMBH

Hannoversche Straße 21
31675 Bückeburg
www.willert.de
info@willert.de
Tel.: +49 5722 9678 - 60