

Slidename	Text
Welcome	Welcome to the Getting Started guide for IBM Rational Rhapsody Developer for C++.
Slide 3	This viewlet shows you how to open the C++ Radio model that was installed with Rhapsody, examine some diagrams to understand the model, and then animate the model to validate its behavior.
Slide 4	The sample models are all in a directory called Samples below your Rhapsody installation directory, which is usually: C:\Program Files\IBM\Rational\Rhapsody\7.5
Slide 5	In this viewlet we open the Radio model directly where it was installed with Rhapsody, and you can do the same as long as you aren't going to modify the model.
Slide 6	BUT if you are going to modify the model at all, or save it, then you should copy the complete Radio directory to somewhere where you can work on it, and open it from there.
Open the Radio model	Let's open the Radio model
Open the Radio model	Click here
Slide 8	Click here
Slide 9	Double click here
Slide 10	Double click here
Slide 11	Click here
Slide 12	Double click here
Slide 13	Click here
Slide 14	Click Open button
Open a Usecase Diagram	The model is now open in Rhapsody. Let's open some diagrams to see what's there. First, let's open the main use case diagram.
Open a Usecase Diagram	Click here
Slide 16	Double click here
Slide 17	This diagram shows that there is one actor who wants either to tune the radio manually by stepping up or down frequencies, or automatically by seeking to a signal, or by recalling a previously-saved frequency.
Open the feature editor	Let's see what additional information is in the model by examining the properties of the select waveband use case.
Slide 19	Double click here
Slide 20	Click here
Slide 21	Here is some additional information that the modeler has put into the Description of the select waveband use case
Slide 21	This button opens a larger text editor window - but we won't use that now because we aren't modifying this model.
Open an Object Model Diagram	Next, let's close the property editor and open an object model diagram (OMD). These are used to show the static structure of the classes and objects in an object-oriented system, and the

	relationships between them.
Slide 23	Click here
Slide 24	Click here
Slide 25	Click here
Slide 26	Click here
Slide 27	Double click here
Slide 28	The Radio class represents the radio. It is a composite class containing instances of the Frequency and Waveband classes. The numbers 1 and 4 in the corner of the nested symbols specify the instantiated number of instances: a Radio has one frequency and four wavebands. The current band is referenced by the itsCurrentWaveband aggregation relationship.
Slide 28	The Waveband class is also a composite class containing five instances of the Frequency class. The current frequency of the waveband is referenced by the itsCurrentFrequency aggregation relationship.
Open a Statechart Diagram	Now let's open the statechart diagram for the Radio class
Open a Statechart Diagram	Right click here
Slide 30	Click here
Slide 31	The diagram isn't all visible, so let's zoom out a bit by clicking the Zoom Out button, and then clicking near the center of the diagram.
Slide 31	Click here
Slide 32	Click here
Slide 33	States are shown as boxes with a label at the top (for example 'on') with specified code to execute on entry and exit. States can be nested in other states, like tuning inside on, and tune inside tuning.
Slide 33	Transitions between states are shown as red lines with an arrow at the destination state. A transition is followed when the specified trigger occurs and an optional guard condition evaluates to true. Transitions can optionally execute code if needed. The trigger, guard, and code are shown in blue on the red transition line.
Start animation	Next we're going to animate this model to to visualise and maybe debug the behaviour of this model of a Radio. First, let's close the diagrams that we have open.
Start animation	Click here three times
Slide 35	Click here
Slide 36	Click here
Slide 37	Next, let's expand the model view so we can see the definition of the Radio class, because when the model is animated an instance of the Radio class will be created below it.

Slide 37	Click here
Slide 38	Click here
Slide 39	Now we're ready to start the Rhapsody animation. This project also contains a small GUI to control the radio during animation, which will be started in a shell window, which we'll minimize. The source of the GUI is provided in the installation below where the radio project was opened at the start of this viewlet.
Slide 39	Click here
Slide 40	Click here
Slide 41	Click here
Slide 42	Click here
Slide 42	Drag to here
Slide 43	The Instances category has appeared because Rhapsody is ready to animate once we click the Go button
Slide 43	Click here
Slide 44	Here's the GUI. You can turn the radio on or off, click + and - to increase and decrease the tuned frequency, and so on. These buttons all create events that the animated model responds to by executing transitions and their associated code.
Slide 45	Now let's examine the instance of class Radio that Rhapsody is animating.
Slide 45	Click here
Slide 46	Double click here
Slide 47	Click here
Slide 47	Drag To Here
Slide 48	Here are the features of the instance of Radio that Rhapsody is animating. You can see the attributes (with their values) and relationships of the instance.
Slide 48	Click here
Animate a Statechart Diagram	Next we'll look at the animated statechart of the Radio class instance. This indicates the current state of the instance with a highlight.
Animate a Statechart Diagram	Right click here
Slide 50	Click here
Slide 51	The instance is in its initial state (denoted by the transition from the red dot) which is the off state, so that state is highlighted. The highlight will move as the instance changes state in response to events, so you can understand and debug the behavior of the model.
Generate an event	We need to turn the radio on. Looking at the transitions from the off state, there's just one that is labelled with the trigger evOnOff and leads to the on state, so this is the event that we need to generate. The transition is visible just above this text symbol. Let's generate this first event manually (not using the

	GUI) by clicking the Event Generator button.
Generate an event	Click here
Slide 53	We must select the Object to send the event to - this will be the instance of the Radio class
Slide 53	Click Select button
Slide 54	Click here
Slide 55	Click OK button
Slide 56	Choose the evOnOff event. In the statechart you can see the transition from the off state to the on state, labelled with the evOnOff trigger.
Slide 56	Click here
Slide 57	Click here
Slide 58	If the event has arguments (for example it carries data), you would put the values in here, or choose ones that you have previously used from the History. The evOnOff event doesn't have any arguments so we don't need to do anything else except click Generate.
Slide 58	Click Generate button
Slide 59	Notice how the highlight has moved to the on state now that the evOnOff event has triggered the transition from the off state. If your PC sound volume is turned up you should also hear the radio station on the default frequency that the Radio starts with. You may want to turn the volume down :-) Note: In this viewlet the sound plays only once, but if you do this animation yourself the sound will play until you turn the radio off or stop the animation.
Slide 59	Click Close button
Animate a Sequence Diagram	An animated sequence diagram (ASD) shows the messages passed between instances as the animation runs. Let's create an ASD, because it is a really easy way to understand the interactions in the radio. There is a prepared empty ASD in the model that already has the lifelines we're interested in watching.
Animate a Sequence Diagram	Click here
Slide 61	Click here
Slide 62	Click here
Slide 63	Click here
Slide 64	Click here
Slide 65	Click Open button
Slide 66	Now let's use the GUI to change the tuning frequency of the radio
Slide 66	Click + button
Slide 67	The ASD shows the evTuneUp event from the GUI. The GUI isn't part of the model, it is outside in the environment of the model, so the event is shown coming from the lifeline labelled

	ENV.
Slide 67	The evTuneUp event triggers a few things to happen, as you can see here. The up operation of the frequency is called, the new frequency is sent to the GUI for display, and a timer is started and expires.If your volume is still turned up, you can now hear a different radio station on the new frequency :-) All this behaviour is specified in the statechart of the Radio class, and the animation in Rhapsody means the diagram isn't just a drawing. You can test it and make sure that it behaves the way you want, or you can learn the behavior of a model you've never seen before.
Slide 67	Click + button
Exit Rhapsody	That's the end of this viewlet, so it's time to stop the animation and exit Rhapsody.Don't forget: do not save the model if you opened it from the Rhapsody installation!
Exit Rhapsody	Click here
Slide 69	Click here
Slide 70	Click here
Slide 71	Click here
Slide 72	Click No button
Slide 73	Click No button
Conclusion	You have completed this viewlet! We hope that it has given you an insight into how to get started with IBM Rational Rhapsody for C++.Rhapsody has allowed you to observe the behavior of the Radio model by animating it, and you have seen how the animation works in real-time in this example.Now you can follow the steps in the viewlet to use Rhapsody to animate the Radio model yourself.The Radio is a small model, but Rhapsody works in just the same way with much larger models where understanding the behavior without animation would be extremely difficult or impossible. With Rhapsody it is simple to animate a model to learn or test its behavior.Please note that this viewlet has not shown you many of the features of Rhapsody which help to make it the leading Model-Driven Development tool for systems and software development.For more information on Rhapsody, Rational software, and IBM please see the resources and links on the next slide.