0:01

[Music] in this video we will create a simple

0:07

to-do list application at the end of the tutorial you will have learned how to display and modify C++ models in key

0:14

quick if you're not familiar with cue quick or kegerator we'd recommend watching the getting started with cute

0:20

hello quick world video first as this for you will assume some familiarity with these topics the first step is to

0:28

launch cue creator will be using version 4 for one along with cute 5 9 2

0:36

let's create a new project click the new project button

0:43

select cute quick controls to application and then click on the choose button

0:51

give the project a name we'll call ours to-do lists separated by hyphens

1:03

select the suitable location for the project we will use the default location

1:10

click Next when you're done

1:16

next choose a build system we'll stick with the default which is cumec

1:24

we're now given the option of cute quick controls to style choose material from the common box and then

1:31

click Next

1:37

select the kid that you'd like to use and then click Next

1:43

click finish to create the project

1:49

our project has been created and now we're looking at main.qml in the editor

1:54

queue creator generated some code for us that we don't need so let's remove it

2:00

first remove the code in main.qml leaving only application window and its

2:05

properties then remove page one form dot UI

2:11

qml by right-clicking on it in the product tree and selecting remove file

2:19

check the delete file permanent on the check box and then click ok we'll be

2:25

doing most of our work in the editor so we won't use UI qml files next rename

2:32

page one thermal two to do list doc qml by right-clicking on it in the project

2:37

tree and selecting rename

2:46

you

2:52

add the following code to application window

3:00

you

3:18

copy the imports at the top of main dock UML and replace the imports and to-do-list Takayama with them

3:31

you

3:36

then replace the top level item with a frame

3:43

the frame type comes from cute quick controls - and provides a convenient way to illustrate the boundary about a

3:49

duelist let's run the application click the Run button or press ctrl R

4:02

you we see a small empty frame in the middle

4:08

of the window close the application and return to creator

4:16

within the frame at all this view

4:24

you this view is the most common view type

4:30

in key quick there are three important things that ListView needs in order to display data asides a model and a

4:38

delegate the size determines the area within which the user can interact with the list items add the following code

4:52

you

4:58

notice that we're using implicit with and implicit height instead of width and height implicit sizes are a natural ah

5:06

default size for an item for example the text type has an implicit size which is

5:12

the width and height of the text it contains in our case frame will automatically resize itself to the

5:18

implicit size of its child item so long as there is only one item we take

5:24

advantage of that here so that we don't have to explicitly specify a size for the frame

5:29

the listviews model is the source of the data that it uses to create list items type of the following code

5:43

this tells of this feud that the model is composed of 100 numbers we use an

5:48

integer based model for now because it will allow us to see how the ListView behaves

5:53

the model could also be a cure model this model and javascript array or a C++

5:59

model amongst other things the delegate is a component a type of

6:05

visual template that describes how list items will look for our to do list we

6:10

want to check box to mark items as done and a text field to enter description we

6:16

also want these items to be laid out side by side so add a row layout

6:27

you rollout comes from the cute quick

6:33

layouts import and behaves similarly to the row from Q quick within the row layout add a checkbox and a tax fields

6:48

you

6:56

these two types are controls from the q-quit controls to import run the

7:01

application to see how it behaves so far you might notice that the contents for

7:07

this view are visible outside of the frame which is not what we want we'll fix that shortly the checkbox and text

7:15

field can be interacted with but as they are not hooked up to the model any changes will be lost when they go far

7:21

enough outside the ListView this is because this view destroys delegates that are not visible in order to save

7:26

memory allowing it to display large amounts of data in an efficient manner to try this out

7:33

check the checkbox that is a few delegates down in the ListView

7:41

scroll down to this view by flicking or scrolling with the mouseville

7:49

then scroll back up to the top the checkbox is no longer checked as it

7:56

was destroyed and recreated close the application and go back to

8:02

creator set clip - drew on a ListView

8:09

this will ensure that any less items outside of you will not be visible in

8:14

the delegate set width to parent width

8:23

now the rollout will be as wide I saw this view whereas before it was only as wide as combined widths of its children

8:29

the chat box and text field we want the sex field to be as wide as possible so

8:35

that the user has plenty of space to write in add the following in touch property binding to text field

8:48

you an attached property provides a way to

8:54

annotate an object with extra properties or signal handles that are otherwise unavailable to it in this case we use

9:01

the via attach type to make sure the text field takes up as much horizontal space as possible

9:08

run the application to see the changes

9:14

notice that the ListView items are no longer visible outside of the view and the text field is wider

9:21

now we'll take a look at the model close the application and return to creator

9:27

replace the integer model with a list model

9:36

within it add a list element object with the following code

9:45

you

9:57

each property in a list element is called a role if you're familiar with C++ models and queued it's the same

10:03

concept the done role solves boolean values and the description role stall strings copy

10:10

and paste the list element and set as properties to the following values

10:20

you

10:35

if we were to run the application now we'd have to list items but they won't display the data from our model because

10:41

they're not connected to it yet in the checkbox control set the checked property to model done

10:55

you Rahl's from tomorrow are available to

11:02

delegates through the model object we could also refer to the role directly without going through the model object

11:08

however doing it this way will result in the code being easier for others to read and understand as they can instantly see

11:15

where the properties come from in text field set the text property to model

11:20

description

11:28

you

11:35

now I'm going to run the application the checkbox and text field are displaying the data from the model

11:42

however any changes to the controls values are not stored in the model for that to happen we need to respond to

11:49

user interaction and set the values of the model rules accordingly close the application and return to creator

11:57

in the checkbox add the following code

12:06

you

12:11

now whenever the checkbox is clicked for your mouse touch or keyboard the new value will be stored in the model in

12:18

text fields at this code

12:26

you

12:33

this does something similar with the only different Spain that the editing finished signal is emitted

12:38

whenever the Enter key is pressed all the text field loses focus our to-do

12:43

list can now be edited but are still using the qml based list model for some

12:48

use cases simple models like this are sufficient especially if the data cannot be editors or does not need to be saved

12:55

to disk in order to be persistent in most cases however the logic of the

13:00

application is in C++ and hence the C++ model is needed to provide a startup to

13:05

qml let's add a C++ model open the file

13:10

menu and select new file or project

13:18

under the files and classes section select cute

13:25

then select cute item model from the list click the choose button

13:33

now we're looking at a wizard that will create the skeleton of our model for the class name type to do model

13:47

to the base class select key abstract this model

13:54

this is a convenience class most models and it's derived from cue abstract item model which is the base

14:01

class of all other models in cubed uncheck customize header row and check

14:07

items are editable the header and source filenames are

14:12

automatically suggested for us based on the class name we entered click Next

14:20

and then finish we're now looking at to do model dot cpp

14:26

to start off we're going to implement just enough functionality so that we can see our c++ model in our key quick list

14:33

view the first thing that we need to do is define the roll names that were using qml done and description switch to the

14:42

header file by pressing f4 or right-clicking anywhere in the file and selecting switch-hitter source from the

14:48

context menu at the following a nun after the

14:54

constructor

15:01

you

15:19

at the following code after function

15:28

you

15:50

right-click on the function signature and select refactor add definition in to do model plus EBP

16:03

we've now been taken to the definition of the function in the source file add the following code

16:15

you

16:55

in the Ruhr account function add a return statement that simply returns 100

17:07

this will give us 100 rows in the model in the data function add this code after

17:14

the fix me comment

17:21

you

18:05

now the model will return placeholder data but we still need to make her more aware of it and use it open main dot cpp

18:17

at an include for to do model th after the cute includes

18:27

you then after the line where qqe

18:33

application is constructed add this code

18:42

you

19:09

this function registers as C++ model with a key ml type system welcome to do this now

19:17

at an import for us to do library after the cute imports

19:28

you replace the list model with to do model

19:39

you open the build menu and select run queue

19:47

make

19:53

this ensures that mock the meta object compiler is run any class with the Q

19:59

object macro that is added to a project requires queue mate to be run now run the application which will also call it

20:06

to be built

20:11

a model data from C++ is displayed notice though it's not possible to

20:16

change the values of the controls

20:21

to make the controls editable we have to implement the set data function which also requires us to store the to do list

20:28

start of somewhere we could store the data in the model but in a real-world scenario the data is stored in the

20:35

backend separate from the model open the file menu and select a new file or

20:42

project under the files and class the section select safe house plus then C++ class

20:52

we're now looking at the C++ class wizard type to-do list into the class

20:57

name field in the base class combo box select queue

21:05

object click Next

21:12

and then click finish we're now looking at to do this dot cpp switch to the

21:18

header file there are three actions to do this class

21:25

should support in terms of to-do items adding new items modifying existing items and removing items each to-do item

21:34

has two properties that we defined as roles earlier done and description let's

21:40

make it to do item struct to make it easier to manage multiple items add the

21:45

following code above to do this class decoration

21:54

you

22:13

include p-vector after the key object include

22:22

you at a private key vector to do items as a

22:28

member variable off to do list

22:36

you

22:46

next we're going to add three functions to support two actions we mentioned earlier after the constructor and before

22:54

the signal's keyword add the following code

23:03

you

23:13

this will allow the model to modify the to-do lists data next add two functions

23:20

below the public slot section

23:28

you

23:44

this will be called by the Comal code under the signal section add the

23:50

following code

23:56

you

24:27

using pre and post signals matches up with what the model expects it needs to

24:32

be notified before something is about to happen to the data that it is exposing and also afterwards now we need to

24:39

provide a way for the model to access the data in the list add this function after the constructor

24:51

you

25:02

let's implement these functions for each function except the signals which don't

25:08

need definitions we're going to be using Creators refactoring option to add a definition in the source file

25:14

right-click on the signature of the items function and select a refactor add

25:20

definition into dualists EVP in the body of the function add the

25:27

following code

25:35

do the same for set automat

25:42

you

25:48

in its body at this code

25:54

you

26:09

first we check if the index is valid returning false if it's not the set data function to do model

26:17

returns a ball to indicate whether or not the data was successfully modified so that's why we also returnable next

26:24

add the following

26:32

you

27:04

this stores the current or old item in a local constable it then checks if the

27:11

new item has identical values to the old one if so it returns false to indicate that nothing has changed finally add

27:18

this code

27:26

you

27:33

if the execution of the code has gotten this far it means that the autumn has changed so restore it and out lists and

27:39

return true now let's implement the two slots

27:47

add a definition for pens autumn via the refactor menu

27:57

at the following code to its body

28:04

you

28:25

we omit the pre and post signals before and after the arm eventually we will connect the model up

28:32

to their signals go back to the header file

28:38

and add a definition for remove completed items

28:45

add the following four loop to its body

28:52

you

29:02

then within the follow at this code

29:10

you

29:50

we loop over each item in the list and check if it's done if so we omit the pre

29:56

signal remove the item and then omit the post signal if the item is not done we

30:01

move on to the next one finally we're going to pretend that our list already had some data in it by

30:07

appending a few items in the constructor

30:16

you

30:44

I have to do this class is now done typically C++ classes like this created

30:50

as part of the larger application back-end and C++ so that's what we're going to do we'll create an instance of

30:57

the past in main dot CPP

31:02

after the last acute include add this line

31:12

you this is necessary in order to be able to

31:18

expose the object to cure Mel then include to-do list on edge

31:30

you after the qml Ridge the top coal at this

31:35

line

31:42

you and before the engine load call add this

31:49

line

31:55

you

32:19

this is where the to-do list is actually exposed to QML we set it as a context

32:24

property which means that it is available to any Q more files loaded by the application there are other ways to

32:31

expose other X to qml as properties but this is the most convenient one

32:36

ronke make

32:43

and then build a project rejected no errors

32:51

now we're going to make to do model aware of to-do lists open to do model 2h

32:57

and add a forward declaration for to-do list

33:07

you we'll only be referring to to-do list

33:13

through pointers in the header so it's not necessary to include it yet after the key object macro begin typing Q

33:20

underscore property and then hit enter once the suggestion with the red icon on the left is visible this is a convenient

33:28

snippet that saves us from typing without changing the position of the cursor type to do list

33:37

then press tab on top list

33:43

the key object attributes are automatically updated press ENTER to confirm it changes

33:49

and then add an asterisk to the left of the first instance of list to market as

33:55

a pointer remove and notify this change part as it's optional and not required

34:00

in our case under the private section of to do model ad this mine

34:13

you raaah click on that member variable and

34:19

select refactor create getter and set our member functions

34:29

switch to the South file an admin include to-do list or edge

34:40

you

34:49

in the constructors initializer list initialize em list with a null pointer

34:59

at the top of set lists add the following code

35:08

you

35:33

before the source of model started changes we must call begin reset model as I'll be connecting to the signals

35:40

other to-do lists it's also a good idea to disconnect from the oldest when setting a new one in our case this isn't

35:47

necessary as a property will only be set once however it's a good habit to get

35:52

into after the list has been assigned and the following if statement

36:04

you if a valid to do this was set on the

36:10

model we will connect to its various signals in order to call the beginning and end functions add the following code

36:16

within the if statement

36:24

you

37:11

this connects a lambda to the pre item a pendant signal of to-do lists the lambda

37:17

calls begin insert rows with an invalid model index which is fine as we want to add the items to the Bert note of the

37:23

model the next two arguments are the first and last indices within which the new items will be inserted since we want

37:31

to append items to the end of the model we use an index as one passed the last valid index in the list

37:37

next connect post item appended to a lambda that calls end insert rows

37:49

you

38:07

connect priyada removed to a lambda that calls begin remove rose

38:19

you

38:46

finally connect posted item removed to a lambda the calls and remove rows

38:58

you

39:12

to match the begin reset model we also call end reset model

39:22

you now is a good time to build a project to

39:27

check that our compiles

39:33

I'm almost ready to use on model and qml let's implement the remaining functions

39:39

in row count a denial point to check to the if statement

39:50

you we're going to do the same father functions that use the aimless member

39:56

variable as is a good practice to safeguard against null point two accesses remove the fix me comment and

40:02

change your return statement to return the amount of items in the doo doo list

40:13

and data at the same null pointer check

40:22

you

40:28

remove the fixed me comment and store the current item in the local constable

40:39

you

40:54

in the switch cases return the relevant data phage roll

41:04

you

41:12

and set data add an if statement that returns false if the null pointer check fails

41:24

you

41:30

then stole the current item at the index in a local variable

41:40

you

41:53

we're going to check which role was set and updated copy the switch code from the auto

41:59

function and paste it after the last line we

42:06

added

42:11

modify the done roll case so that it looks like this

42:20

you

42:32

then modify the description role case

42:41

you remove the fix me comment

42:50

and change this statement

42:57

you

43:12

we take advantage of set Adam adds return value to determine whether or not the data has actually changed and

43:18

therefore only emit the data change signal if we need to finally if the data

43:23

has changed we return true otherwise false in the flags function remove the

43:30

fix me comment

43:35

the item is edible flag is enough for our use case open Maine dot CPP

43:43

and at the following after the cold to kill mirage the top

43:54

you

44:39

this forces to-do list is a qml type so that can be used as a property of to do

44:44

model as it is a back-end type we don't intend for it to be created in qml so we

44:50

use the qumar register uncredible type function this function prints the error message that we passed as the last

44:57

argument if we attempt to instantiate the type in QML in to do list dot qml

45:03

assignment to do list contacts property to the list property off to do model

45:15

you run the application

45:26

we can now edit the model data that comes from C++ in a cute quick ListView

45:39

before we can consider our application complete we need to make it possible for users to add and remove items from a

45:45

list we're going to add two buttons below the to-do list to achieve this close the application

45:53

and at a column layout as a parent of the frame

46:03

you to quickly indent a qml file select the

46:11

code and press ctrl I

46:16

after the frame at a rollout

46:23

within the rollout add two buttons

46:31

you

46:50

run the application to see how it looks so far

46:55

notice how the buttons happen to align nicely with the width of the ListView by convenient this is just a coincidence we

47:03

need to take a few steps to ensure the combined widths of the buttons match the width of the ListView and vice versa to

47:11

simulate how wide the buttons might be on a system with a larger default font size let's temporarily add some text to

47:18

the labels add three spaces at the end of each button stick string

47:30

you

47:42

next we want to make sure that each button fills the width of the rollout

47:53

you

47:58

then at the same bonding to the frame

48:07

finally make ListView fill the frame

48:15

run the application again

48:21

now if the button text is too wide this view will stretch accordingly

48:27

remove the extra spaces from the text

48:34

to complete our application we need to make the buttons do something add the following to the Add button

48:47

you

48:56

add the following to the remove button

49:03

you

49:15

from the application again

49:21

you we can now add edit and remove items

49:29

from our sleep loss past model

49:36

you

49:42

you [Music]