MVC Kit for XRT Widgets

Programmer's Guide & Reference Manual

UNIX Edition ■ Version 2.0



Software Development Productivity

260 King Street East
Toronto, Ontario, Canada M5A 1K3
(416) 594-1026
www.klgroup.com

January 1999

Part No: KLMVC20

Copyright © 1997-1999 by KL Group Inc. All Rights Reserved.

KL Group, the KL Group logo, XRT, XRT/3d, XRT/care, XRT/field, XRT/gear, XRT/graph, XRT/table and XRT PDS are trademarks or registered trademarks of KL Group Inc. in Canada, United States and other countries.

Printed in Canada on recycled paper. 

Table of Contents

Preface 1
Introduction 1
Assumptions 1
Overview of Manual 1

Part I: Using the MVC Kit

1 Getting Started: Developing a Simple Program 5
1.1 Introduction 5
1.2 Environment Variables 5
1.3 Compiling and Authorizing a Program 6
1.4 A Basic Object 6
1.5 Creating a View of the data 8
A View Using XRT/table 10
A View Using XRT/graph 11
A View Using XRT/3d 12

2 The MVC Object 13
2.1 Introduction 13
2.2 Attributes 13
2.3 Creating an MVC Object 14
2.4 Utility Functions 14

3 MVC Object Data and Labels 15
3.1 Introduction 15
3.2 Getting Data into the MVC Object 15
3.3 Manipulating the Data 16
3.4 Manipulating Labels 16
3.5 Writing the Data 17
3.6 Example Using Static Data 17

4 View Objects 19
4.1 Introduction 19
4.2 Registering View Classes 19
4.3 Common Attributes 20
Display Policy 20
Range 20
Editability 21
4.4 XRT/table Views 21
4.5 XRT/graph Views 21
4.6 XRT/3d Views 22

5 Advanced Programming Topics 23
5.1 Source Code 23
5.2 Custom Views 23
Initializing a View 24
Destroying a View 24
Changing Editability 24
Updating a View 24

Part II: Reference Appendices

A Object Services Reference 29
A.1 MVC Object Services 29
A.2 MVC Data Services 30
A.3 MVC Label Services 31
A.4 View Object Services 32
Graph View Object Services 32
3d View Object Services 32
Table View Object Services 33
A.5 Utility Functions 34

B Procedures and Methods Reference 35

Preface

Introduction ■ *Assumptions*
Overview of Manual

Introduction

The Model View/Control (MVC) Kit is an add-on for the products XRT/graph, XRT/table and XRT/3d which allows the developer to present multiple views of the same data through any of the above products. Through the MVC Kit, changes made to the data in any view can be reflected in the base data and all other views.

This manual describes the MVC Kit only. The MVC Kit is intended to be used with XRT/table, XRT/graph and XRT/3d, and so the *XRT/table for Motif, Programmer's Guide and Reference Manual*, *XRT/graph for Motif, Programmer's Guide and Reference Manual* and *XRT/3d for Motif, Programmer's Guide and Reference Manual* should be used in conjuction with this manual.

Assumptions

This manual assumes that the reader is proficient with the C language and the Motif toolkit as well as one or more of XRT/table, XRT/graph and XRT/3d. C concepts such as "an array of `char *`" and "pointer to a structure" must be understood before beginning to program Motif and XRT-based applications. Additionally, it is expected that readers will have access to the manuals for at least one of XRT/table, XRT/graph or XRT/3d.

Overview of Manual

Part I describes how to use the MVC Kit.

Chapter 1, "Getting Started: Developing a Simple Program," provides a quick introduction to programming using the MVC Kit.

Chapter 2, "The MVC Object," provides information on the attributes and services of the MVC object.

Chapter 3, "MVC Object Data and Labels," provides a explanation of the use and services of MVC data and its labels.

Chapter 4, “View Objects,” describes the attributes and services of the View object and its subtypes.

Chapter 5, “Advanced Programming Topics,” discusses more advanced issues involving the MVC Kit.

Part II contains detailed technical reference information.

Appendix A, “Object Services Reference,” provides a concise reference of all MVC Kit procedures, grouped by object and function.

Appendix B, “Procedures and Methods Reference,” lists all MVC Kit procedures, methods, and convenience routines in alphabetical order.

Part I

Using the MVC Kit

1

Getting Started: Developing a Simple Program

Introduction ■ *Environment Variables*
Compiling and Authorizing a Program ■ *A Basic Object*
Creating a View of the data

1.1 Introduction

This chapter allows you to immediately try out the MVC Kit by compiling and running an example program. The first version of the program, *sample1.c*, will load data into an MVC object and display it to the terminal for verification. The second version, *sample2.c*, will create a view on the data in either an XRT/graph, XRT/table or XRT/3d widget.

1.2 Environment Variables

Before getting started, be sure your Unix environment is set up correctly. An environment variable called *XRTHOME* should be defined as the name of the directory containing the distribution files for one of the products XRT/table, XRT/graph or XRT/3d.

You can determine the current value for *XRTHOME* by entering:

```
echo $XRTHOME
```

If your *XRTHOME* environment variable is not set up correctly, see your system administrator or the *Release & Installation Notes*.

Before attempting to use the MVC Kit, you must be able to compile Motif applications and applications using XRT/graph, XRT/table or XRT/3d. Consult your Motif documentation to determine where the Motif library and include files are located. Make sure you can compile and run simple Motif and XRT applications before you begin to use the MVC Kit.

1.3 Compiling and Authorizing a Program

Since the first example program, *sample1.c*, does not use any XRT widgets, it may be run as compiled, without needing to be authorized. Accordingly the following command example is given for *sample2.c*.

If the X, Motif and Xrt libraries and include files are installed in the standard places (normally */usr/lib*, */usr/include/X11*, and */usr/include/Xm)*, *sample2.c* may be compiled and authorized by entering:[1]

```
cc sample2.c -I$XRTHOME/include \
    -L$XRTHOME/lib -o sample2 \
    -lxrtmvc -lxrtm -lxrt3d -lxrttable -lXm -lXt -lX11 -lm
$XRTHOME/bin/xrt_auth sample2
XRTHOME:[LIB]XRTMVC.OLB/LIBRARY
```

If the libraries and include files are not installed in standard locations, additional use of the `-I` and `-L` compiler flags will be required.

xrt_auth

Remember that all programs using XRT widgets need to be authorized using the *xrt_auth* utility. Consult your manual for XRT/graph, XRT/3d or XRT/table for details.

C++ and ANSI C

The MVC Kit *include* files are compatible with C++ and ANSI C as well as with traditional "Kernighan and Ritchie" C.

1.4 A Basic Object

Figure 1 lists the program *sample1.c*. This program is a minimal C program which loads the data from a file into an MVC object and then copies the data to the terminal. A data file called *sample.dat* should be created containing a few rows of comma-separated numbers.

1. Compilation options depend on your computer architecture. See the *Makefile* in *$XRTHOME/src/common-demos* for an example of how to compile on your system.

```
#include <stdio.h>
#include <Xm/XrtMvc.h>

int
main(int argc, char *argv[])

{
    XrtMvc  mvc;
    FILE    *fp;
    int     rows, columns, i, j;

    /* Create the MVC object. */
    mvc = XrtMvcCreate();

    /* Open the data file. */
    if (!(fp = fopen("sample.dat", "r"))) {
        perror("sample.dat");
        exit(0);
    }

    /* Read the data into the MVC object. */
    XrtMvcDataReadAscii(mvc, fp, ',');
    fclose(fp);

    /* Dump the MVC data structure to the terminal. */
    columns = XrtMvcDataGetColumnCount(mvc);
    rows = XrtMvcDataGetRowCount(mvc);

    for (i = 0; i < rows; i++) {
        for (j = 0; j < columns; j++) {
            printf("%7.2f ", XrtMvcDataGetValue(mvc, i, j));
        }
    printf("\n");

}
```

Figure 1 sample1.c

Figure 2 shows some possible example data:

1.0	2.0	3.0	4.0	5.0
2.0	4.0	6.0	8.0	10.0
3.0	6.0	10.0	16.0	24.0
4.0	8.0	16.0	30.0	40.0
5.0	10.0	24.0	40.0	60.0

Figure 2 Example data for the programs sample1.c and sample2.c

When the program is compiled and run, the output shown in Figure 3 displays.

```
% sample1
1.00    2.00    3.00    4.00    5.00
2.00    4.00    6.00    8.00   10.00
3.00    6.00   10.00   16.00   24.00
4.00    8.00   16.00   30.00   40.00
5.00   10.00   24.00   40.00   60.00
%
```

Figure 3 The output of the sample1 program

1.5 Creating a View of the data

The *sample1.c* program does not demonstrate the use of the MVC with an XRT widget. In order to fix this problem, we begin with a new program *sample2.c*.

The actual segment of code to create the View widget, register the View and associate the View with the widget is not included in this figure. Since it is not possible to assume that all customers will have a particular product, 3 code fragments follow the initial listing. There is a code listing for each of XRT/table, XRT/graph and XRT/3d with the code required to create a View of each type.

The listing for the common sections of *sample2.c* is shown in Figure 4.

```
#include <stdio.h>
#include <Xm/XrtMvc.h>

int
main(int argc, char *argv[])

{
    Widget          toplevel, view_widget;
    XrtMvcView  view_object;
    XrtMvc          mvc;
    XmString        str;
    FILE            *fp;

    /* Create the MVC object. */
    mvc = XrtMvcCreate();

    /* Open the data file. */
    if (!(fp = fopen("sample.dat", "r"))) {
        perror("sample.dat");
        exit(0);
    }

    /* Read the data into the MVC object. */
    XrtMvcDataReadAscii(mvc, fp, ',');
    fclose(fp);

    /* Initialise Intrinsics. */
    toplevel = XtInitialize(argv[0], "demo", NULL, 0, &argc, argv);

        /*
        ...Product dependent section (see figures below)...
        */

    XtRealizeWidget(toplevel);
    XtMainLoop();
}
```

Figure 4 sample2.c

1.5.1 A View Using XRT/table

If you have XRT/table, the following code segment will cause *sample2.c* to display the data of the MVC object in a table.

```
#include <Xm/XrtTable.h>

...

    /* Create a view widget to display the MVC data. */
    view_widget = XtVaCreateManagedWidget("mvc_view",
                            xtXrtTableWidgetClass, toplevel,
                            NULL);

    /* Register the table view class. */
    XrtMvcViewXrtTableRegister();

    /* Associate a view of the data with the table widget and display it. */
    view_object = XrtMvcViewCreate(mvc, XrtMvcXrtTableViewClass,
                                view_widget, NULL);
```

Figure 5 XRT/table-specific code for sample2.c

When the program is compiled and run, the window shown in Figure 6 displays.

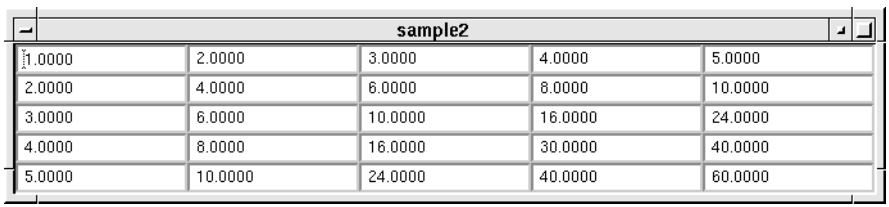


Figure 6 Window displayed by the XRT/table version of sample2

1.5.2 A View Using XRT/graph

If you have XRT/graph, the following code segment will cause *sample2.c* to display the data of the MVC object in a plot graph.

```
#include <Xm/XrtGraph.h>

...

    /* Create a view widget to display the MVC data. */
    view_widget = XtVaCreateManagedWidget("mvc_view",
                          xtXrtGraphWidgetClass,toplevel,
                          XtNxrtType,         XRT_TYPE_PLOT,
                          NULL);

    /* Register the graph view class. */
    XrtMvcViewXrtGraphRegister();

    /* Associate a view with the graph widget and display it. */
    view_object = XrtMvcViewCreate(mvc, XrtMvcXrtGraphViewClass,
                                view_widget, NULL);
```

Figure 7 XRT/graph-specific code for sample2.c

When the program is compiled and run, the window shown in Figure 8 displays.

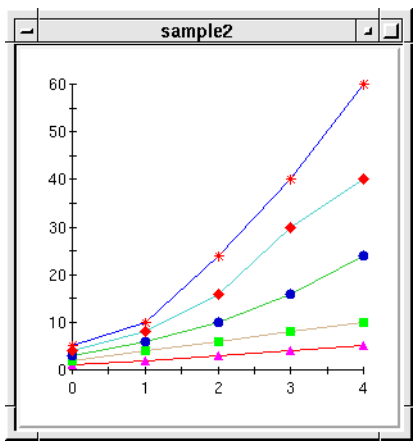


Figure 8 Window displayed by the XRT/graph version of sample2

1.5.3 A View Using XRT/3d

If you have XRT/3d, the following code segment will cause *sample2.c* to display the data of the MVC object as a 3D surface.

```
#include <Xm/Xrt3d.h>

...

    /* Create a view widget to display the MVC data. */
    view_widget = XtVaCreateManagedWidget("mvc_view",
                            xtXrt3dWidgetClass, toplevel,
                            XtNxrt3dDrawShaded, True,
                            XtNxrt3dDrawZones,  True,
                            NULL);

    /* Register the 3d view class. */
    XrtMvcViewXrt3dRegister();

    /* Associate a view with the 3d widget and display it. */
    view_object = XrtMvcViewCreate(mvc, XrtMvcXrt3dViewClass,
                                   view_widget, NULL);
```

Figure 9 XRT/3d-specific code for sample2.c

When the program is compiled and run, the window shown in Figure 10 displays.

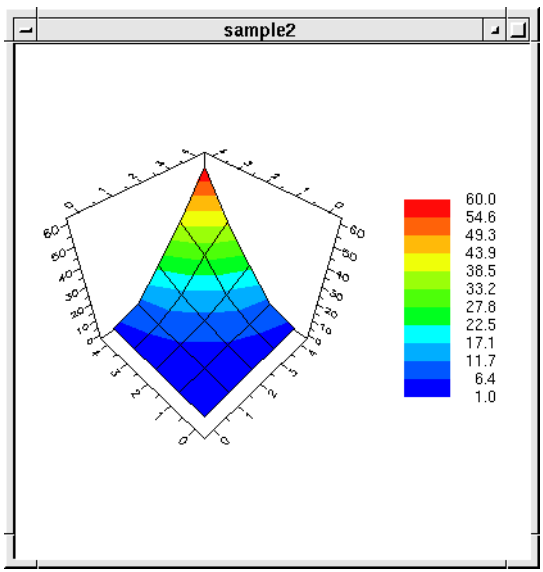


Figure 10 Window displayed by the XRT/3d version of sample2

2

The MVC Object

Introduction ■ *Attributes*
Creating an MVC Object ■ *Utility Functions*

2.1 Introduction

This section describes the MVC object and its attributes and related methods. The MVC object contains a data space which is referenced by the View objects, methods for modifying the data, some general methods and global attributes. The data space and related methods will be dealt with in the next section.

Appendix B on page 35 contains a reference of all MVC Kit methods.

2.2 Attributes

The MVC object has the following attributes which may be retrieved or modified by the indicated methods:

```
ColumnCount     integer         0
```

The number of columns of data in the MVC object. This attribute is read-only and accessed by *XrtMvcDataGetColumnCount()*.

```
Update          Boolean         True
```

Whether updates are being sent to Views. Accessed by *XrtMvcGetUpdate()* and set by *XrtMvcSetUpdate()*.

```
UserData        XtPointer       NULL
```

User data associated with the MVC object. Accessed by *XrtMvcGetUserData()* and set by *XrtMvcSetUserData()*.

```
RowCount        integer         0
```

The number of rows of data in the MVC object. This attribute is read-only and accessed by *XrtMvcDataGetRowCount()*.

The Update Attribute

The `Update` attribute controls whether changes in the MVC data are passed on to the Views or buffered. If it is `True` all changes will be passed to the Views as they occur; if it is `False` all changes will be queued until it is turned on again. This can be useful if a large number of changes are to be made to the MVC object, to prevent unnecessary redrawing of the Views at intermediate stages. The `Update` attribute can be set using the *XrtMvcSetUpdate()* procedure.

2.3 Creating an MVC Object

The MVC object itself is created using the *XrtMvcCreate()* procedure, which creates the bare MVC object structure. The application must then add data, using *XrtMvcDataReadAscii()* to input a file or building the array up using the *XrtMvcDataAppendColumn()*, *XrtMvcDataAppendRow()* or *XrtMvcDataSetValue()* procedures. If this is done after Views have been created, it may be useful to turn the `Update` attribute off to avoid the Views being repainted at intermediate stages.

2.4 Utility Functions

The MVC Kit provides a few utility functions, such as *XrtMvcIntersectRanges()* which can be useful for building custom Views, to determine if a range of changed elements in the MVC object overlaps the range of elements displayed in the View, and *XrtMvcWidgetToView()* which identifies the View associated with a widget. Descriptions and calling sequences for these procedures can be found in Appendix B on page 35.

3

MVC Object Data and Labels

Introduction ■ *Getting Data into the MVC Object*
Manipulating the Data ■ *Manipulating Labels*
Writing the Data ■ *Example Using Static Data*

3.1 Introduction

The information that the MVC object stores to build Views can be separated into two classes: data and labels. Data are all numbers of type `double`, and stored in a sparsely-populated structure that can be visualized as a two-dimensional array with rows and columns. Labels are stored in a list structure and are of type `String`.

Undefined elements of the MVC data are represented as `XRTMVC_VALUE_UNDEFINED`. This value will be passed to any View which is displaying an element which has not been set.

3.2 Getting Data into the MVC Object

The simplest way to get data into an MVC object is by reading it from a file. If the data exists in a Unixfile (and it does not need to be changed or updated in real time) and the data can be massaged so that the file conforms to a syntax understood by *XrtMvcDataReadAscii()* then this procedure can be called to load the data from the file into the MVC object. *XrtMvcDataReadAscii()* is documented in Appendix B on page 35.

Another approach is to populate the MVC object with data directly by building rows or columns that can be appended to the MVC object using *XrtMvcDataAppendRow()* or *XrtMvcDataAppendColumn()*, or by setting elements directly using *XrtMvcDataSetValue()*. The data can be read from a data file (perhaps using *fgets(3)* or *fscanf(3)*), generated internally, or provided by another application.

Labels can only be added by the mechanism of the *XrtMvcLabelAppendRow()* and *XrtMvcLabelInsertRow()* procedures.

3.3 Manipulating the Data

There are a number of procedures and methods that set and retrieve data inside a data object, and other methods to reorganize and delete the data. They are summarized in following table. Complete reference information is in Appendix B on page 35. The source code for these procedures is in *$XRTHOME/src/mvc*.

Adding and Removing Data	
XrtMvcDataAppendRow()	Appends a row of data to the MVC object.
XrtMvcDataAppendColumn()	Appends a column of data to the MVC object.
XrtMvcDataInsertRow()	Inserts a row of data into the MVC object.
XrtMvcDataInsertColumn()	Inserts a column of data into the MVC object.
XrtMvcDataDeleteRows()	Removes a set of rows of data from the MVC object.
XrtMvcDataDeleteColumns()	Removes a set of columns of data from the MVC object.
XrtMvcDataDeleteAll()	Removes all data from the MVC object.
Retrieving and Changing Data Values	
XrtMvcDataGetValue()	Retrieves the value of a number at a given (x, y) coordinate.
XrtMvcDataSetValue()	Sets the data value at a given (x, y) location.
XrtMvcDataMoveRows()	Reposition a set of rows within the data object.
XrtMvcDataMoveColumns()	Reposition a set of columns within the data object.

3.4 Manipulating Labels

Labels have virtually the same set of access procedures as data does, except for the read and write file and individual get and set procedures. Data manipulations do not communicate with the label procedures, so it is necessary to call both data and label functions for every append, insert, move and delete function to keep the data and labels in sync.

Complete reference information is in Appendix B on page 35. The source code for these procedures is in *$XRTHOME/src/mvc*.

XrtMvcLabelAppendRow()	Appends a row label to the MVC object's row label list.
XrtMvcLabelAppendColumn()	Appends a column label to the MVC object's column label list.
XrtMvcLabelInsertRow()	Inserts a row label into the MVC object's row label list.
XrtMvcLabelInsertColumn()	Inserts a column label into the MVC object's column label list.
XrtMvcLabelDeleteRows()	Removes a set of row labels from the MVC object.

XrtMvcLabelDeleteColumns()	Removes a set of column labels from the MVC object.
XrtMvcLabelDeleteAll()	Removes all labels from the MVC object.
XrtMvcLabelMoveRows()	Reposition a set of row labels within the row label list.
XrtMvcLabelMoveColumns()	Reposition a set of column labels within the column label list.

3.5 Writing the Data

The MVC data can be written to a text file using the procedure *XrtMvcDataWriteAscii()*. As with *XrtMvcDataReadAscii()*, this procedure does not operate on labels, just on the data. It takes a format to use to print the numbers and a delimiter to separate the numbers in a row of the MVC data, one row per line of the file.

3.6 Example Using Static Data

If the data to be displayed is known at compile time, it can simply be defined in static variables and then be appended to the MVC object.

The demo program *$XRTHOME/src/common_demos/mvc/main.c* contains an example of loading the MVC object with static data.

4

View Objects

Introduction ■ *Registering View Classes*
Common Attributes ■ *XRT/table Views*
XRT/graph Views ■ *XRT/3d Views*

4.1 Introduction

This section describes the various types of View objects and their attributes and related methods. A View object contains a representation of a range of the data stored in the MVC object.

The View object types must be registered using one of the Register methods before they can be instantiated.

Appendix B on page 35 contains a reference of all MVC Kit methods.

4.2 Registering View Classes

Before any View is created, the View class of which it will be an instance must be registered. This process defines certain constants and builds a class structure which includes the basic callbacks for maintaining the View.

For most simple purposes, it will suffice to call one of *XrtMvcViewXrtTableRegister()*, *XrtMvcViewXrtGraphRegister()* or *XrtMvcViewXrt3dRegister()*. If Views of two or more classes are to be created, the convenience function *XrtMvcViewRegisterAllClasses()* may be used to register all three predefined Views at once.

The procedure *XrtMvcViewRegisterClass()* allows the developer to define a custom class by specifying all the callbacks to be used to create, maintain and destroy the View.

4.3 Common Attributes

All View objects have the following common attributes:

`DisplayPolicy` `XrtMvcDisplayPolicy` `XRTMVC_DISPLAY_ALL`

The amount of the data to display: none, all or the specified range.

`Range` `XrtMvcRange (0, 0 -> XRTMVC_LAST_ROW, XRTMVC_LAST_COLUMN)`

The range of data displayed in a range-constrained view.

`Editability` `Boolean` `True`

If the data in the view can be modified.

`UserData` `XtPointer` `NULL`

Like `XmNuserData`, this is storage reserved for developer use.

For each attribute, there is a method to get the current value of the attribute, called *XrtMvcViewGet{attribute name}()*, and a method to change the value of the attribute, called *XrtMvcViewSet{attribute name}()*.

4.3.1 Display Policy

This specifies how much of the MVC data is to be displayed by the View. It has 3 possible values:

`XRTMVC_DISPLAY_ALL`	Display all the data.
`XRTMVC_DISPLAY_RANGE`	Display only data within the specified range.
`XRTMVC_DISPLAY_NOTHING`	Display no data.

If `XRTMVC_DISPLAY_RANGE` is set, then the range attribute should contain a valid range of the data to display.

4.3.2 Range

Defines a rectangular region of the MVC data which can be used to restrict the data displayed in the View. The attribute `DisplayPolicy` must be set to `XRTMVC_DISPLAY_RANGE` to use the `Range` attribute. The constants `XRTMVC_LAST_ROW` and `XRTMVC_LAST_COLUMN` can be used to mark the end of the array even if its extents are unknown or variable.

4.3.3 Editability

Controls whether local changes to the data in the View are reflected in the MVC data. If `True`, supported editing functions create corresponding changes in the MVC; if `False` all changes are transient, and will be overwritten the next time the View receives an update or redraws.

Care should be taken that editable views don't get out of step with the MVC data, as, for example, they might if updates are being buffered by the MVC object.

Warning: If views are editable, and the MVC data object is buffered, the two copies of the data will fall out of synchronization and may differ from that of the View.

4.4 XRT/table Views

XRT/table Views reference the data directly from the MVC object at each repaint. Currently the only operation which is supported for making changes in the MVC data from an editable table is a single cell edit. None of XRT/table's drag-and-drop features are supported for this.

Table Views have the following additional attributes:

```
DisplayFormat   String          "%.4lf"
```

The format to apply to table data on display.

```
ScanFormat      String          "%lf"
```

The format used to parse data input in the table.

```
UndefinedString String          "undefined"
```

The string to display for cells whose value is undefined.

`DisplayFormat` takes a *printf(3)* style format string, which must include a conversion character which accepts an argument of type double. `ScanFormat` takes a *scanf(3)* style format string which must have a single conversion of type "`le`," "`lf`" or "`lg`."

The prefixes for the method names are *XrtMvcViewXrtTableGet{name}()* and *XrtMvcViewXrtTableSet{name}()*.

4.5 XRT/graph Views

XRT/graph Views are given a local copy of the MVC data range they are displaying. Thus it is possible to make changes to the data, especially if the DataHandle is used, which do not reflect changes made in the MVC data. Any such changes will be overwritten when the View is updated from the MVC.

Graph Views have the following additional attribute:

`XValuesRow` `integer` `XRTMVC_UNSPECIFIED_ROW`

A row of data to be used to define the X-axis locations of the columns of data.

If the X values row is not set, it will contain the value `XRTMVC_UNSPECIFIED_ROW` and the graph View will use the column number to locate points along the X-axis. If this attribute is set, it is possible to change the values in the reference row by horizontally dragging points on the graph.

The method names are *XrtMvcViewXrtGraphGetXValuesRow()* and *XrtMvcViewXrtGraphSetXValuesRow()*.

4.6 XRT/3d Views

XRT/3d Views are also given a local copy of the MVC data, and the same considerations as for XRT/graph Views apply to modifying that data.

3D Views have the following additional attributes:

`XValuesRow` `integer` `XRTMVC_UNSPECIFIED_ROW`

A row of data to be used to define the X-axis locations of the columns of data.

`YValuesColumn` `integer` `XRTMVC_UNSPECIFIED_COLUMN`

A column of data to be used to define the Y-axis locations of the rows of data.

If the X values row of Y values column are not set, they will contain the values `XRTMVC_UNSPECIFIED_ROW` and `XRTMVC_UNSPECIFIED_COLUMN` respectively. The 3D View will use the column number to locate points along the X-axis, or the row number as the Y coordinate.

In order to get a sensible graph, it is necessary that the X values row and Y values column be regular (in other words, XRT/3d Views can only handle regular grid data).

The prefixes for the method names are *XrtMvcViewXrt3dGet{name}()* and *XrtMvcViewXrt3dSet{name}()*.

5

Advanced Programming Topics

Source Code
Custom Views

5.1 Source Code

The MVC Kit ships with complete source code for all the library procedures. The library code, unlike the header files, is not compatible with K&R C, it must be compiled with an ANSI-compliant compiler.

The source code can be found in *$XRTHOME/src/mvc*.

5.2 Custom Views

The procedure *XrtMvcViewRegisterClass()* allows the user to create a custom View class by defining the functions to initialize, destroy, update, change the editability of the custom View. The function templates, as defined in the header file *XrtMvc.h* are:

```
typedef void (*XrtMvcViewInitializeProc)
     (XrtMvc mvc, XrtMvcViewStruct *view);
```
Initialize a View

```
typedef void (*XrtMvcViewDestroyProc)
     (XrtMvc mvc, XrtMvcViewStruct *view);
```
Destroy a View

```
typedef void (*XrtMvcViewChangeEditProc)
     (XrtMvc mvc, XrtMvcViewStruct *view);
```
Set the editability of the View

```
typedef void (*XrtMvcViewUpdateProc)
     (XrtMvc mvc, XrtMvcViewStruct *view,
     XrtMvcChangeStruct *changes, int num_changed);
```
Update the View

5.2.1 Initializing a View

The initialize function will be called by *XrtMvcViewCreate()* to build the View. This function takes the MVC object and a pointer to the View.

This function will allocate and set any view-specific variables which are required. The attribute `view_data` is available in the `XrtMvcViewStruct` to hold view-specific data. (The View structure `XrtMvcViewStruct` is defined in *XrtMvc.h*.)

5.2.2 Destroying a View

The destroy function can be called by *XrtMvcViewDestroy()* or by the widget destruction callback to uncreate the View. Like the initialization function, this takes the MVC object and a pointer to the View.

The destroy function should free any memory allocated by the initalization function and perform any other view-specific cleanup tasks (closing a file, for instance).

5.2.3 Changing Editability

The change_edit function also gets passed the MVC and the View, and is called to change the editability of the View, passed down from *XrtMvcViewSetEditability()*.

5.2.4 Updating a View

The update function takes the MVC and the View and a pointer to a list of changes to the MVC data with a count of the number of changes. Each change is represented by the structures:

```
typedef struct {
    XrtMvcChangeAction action;
    XrtMvcRange range;
} XrtMvcChangeStruct;

typedef struct {
    int start_row;
    int start_column;
    int end_row;
    int end_column;
} XrtMvcRange;
```

The *action* can be one of the following values:

```
XRTMVC_APPENDED_COLUMNS
XRTMVC_APPENDED_COLUMN_LABELS
XRTMVC_APPENDED_ROWS
XRTMVC_APPENDED_ROW_LABELS
XRTMVC_CHANGED_ALL
XRTMVC_DELETED_COLUMNS
XRTMVC_DELETED_COLUMN_LABELS
XRTMVC_DELETED_ROWS
XRTMVC_DELETED_ROW_LABELS
XRTMVC_INSERTED_COLUMNS
XRTMVC_INSERTED_COLUMN_LABELS
XRTMVC_INSERTED_ROWS
XRTMVC_INSERTED_ROW_LABELS
XRTMVC_LABELS_CHANGED
XRTMVC_MOVED_COLUMNS
XRTMVC_MOVED_COLUMN_LABELS
XRTMVC_MOVED_ROWS
XRTMVC_MOVED_ROW_LABELS
XRTMVC_VALUES_CHANGED
```

The MVC object will perform certain limited optimization of the change actions; if an `XRTMVC_CHANGED_ALL` action is buffered, then all actions previously buffered will be removed.

The update function may find the procedure *XrtMvcIntersectRanges()* useful to filter the list of changes for only those changes which apply to the currently displayed range of the View.

Part II

Reference Appendices

A

Object Services Reference

MVC Object Services ■ *MVC Data Services*
MVC Label Services ■ *View Object Services*
Utility Functions

This appendix groups all the procedures included with the MVC kit into related units. Each function is expanded with a brief description. For further details and calling sequence for any procedure, refer to Appendix B, "Procedures and Methods Reference."

A.1 MVC Object Services

General Functions	
XrtMvcCreate()	Create an MVC object.
Get and Modify Attributes	
XrtMvcGetViewCount()	Retrieve the number of Views owned by the MVC object.
XrtMvcGetUpdate()	Returns the current update status.
XrtMvcSetUpdate()	Sets the update status.
XrtMvcGetUserData()	Retrieves any user data associated with the MVC object.
XrtMvcSetUserData()	Sets the user data of the MVC object.

A.2 MVC Data Services

Adding Data	
XrtMvcDataAppendRow()	Appends a row of data to the MVC object.
XrtMvcDataAppendColumn()	Appends a column of data to the MVC object.
XrtMvcDataInsertRow()	Inserts a row of data into the MVC object.
XrtMvcDataInsertColumn()	Inserts a column of data into the MVC object.
XrtMvcDataReadAscii()	Populate the MVC object with data read from a file.
Deleting Data	
XrtMvcDataDeleteRows()	Removes a row of data from the MVC object.
XrtMvcDataDeleteColumns()	Removes a column of data from the MVC object.
XrtMvcDataDeleteAll()	Removes all data from the MVC object.
Moving Data	
XrtMvcDataMoveRows()	Reposition a set of rows within the data object.
XrtMvcDataMoveColumns()	Reposition a set of columns within the data object.
Retrieving and Changing Data Values	
XrtMvcDataGetColumnCount()	Return the number of columns of data in the MVC object.
XrtMvcDataGetRowCount()	Return the number of rows of data in the MVC object.
XrtMvcDataGetValue()	Retrieves the value of a number at a given (x, y) coordinate.
XrtMvcDataSetValue()	Sets the data value at a given (x, y) location.
XrtMvcDataWriteAscii()	Write the data of the MVC object to a file.

A.3 MVC Label Services

Adding Labels	
XrtMvcLabelAppendRow()	Appends a row label to the row label list.
XrtMvcLabelAppendColumn()	Appends a column label to the column label list.
XrtMvcLabelInsertRow()	Inserts a set of row labels into the row label list.
XrtMvcLabelInsertColumn()	Inserts a set of column labels into the column label list.
Moving Labels	
XrtMvcLabelMoveRows()	Reposition a set of row labels within the row label list.
XrtMvcLabelMoveColumns()	Reposition a set of column labels within the column label list.
Deleting Labels	
XrtMvcLabelDeleteRows()	Removes a set of row labels.
XrtMvcLabelDeleteColumns()	Removes a set of column labels.
XrtMvcLabelDeleteAll()	Removes all labels from the MVC object.

A.4 View Object Services

Creating and Destroying Views	
XrtMvcViewCreate()	Create a View object.
XrtMvcViewDestroy()	Destroy a View object.
Widget and Registering Classes	
XrtMvcViewGetWidget()	Retrieve the widget that owns the View.
XrtMvcViewRegisterAllClasses()	Register all standard View classes.
XrtMvcViewRegisterClass()	Register a custom View class and callbacks.
Attributes	
XrtMvcViewGetDisplayPolicy()	Get the current use of the range to display data.
XrtMvcViewSetDisplayPolicy()	Set the policy for the use of the range to display.
XrtMvcViewGetEditability()	Get the status of the View for edits.
XrtMvcViewSetEditability()	Set the status of the View for accepting edits.
XrtMvcViewGetRange()	Retrieve the range of data to be displayed.
XrtMvcViewSetRange()	Set the range of data to be displayed.
XrtMvcViewGetUserData()	Retrieve developer-specified data associated with the View.
XrtMvcViewSetUserData()	Set the user data of the View.

A.4.1 Graph View Object Services

Register Class	
XrtMvcViewXrtGraphRegister()	Register the Graph View class and callbacks.
Attributes	
XrtMvcViewXrtGraphGetXValuesRow()	Get the row of data used to provide the X coordinates of displayed data.
XrtMvcViewXrtGraphSetXValuesRow()	Set the x coordinate reference row.

A.4.2 3d View Object Services

Register Class	
XrtMvcViewXrt3dRegister()	Register the 3d View class and callbacks.
Attributes	

XrtMvcViewXrt3dGetXValuesRow()	Get the row of data used to provide x coordinates for the data.
XrtMvcViewXrt3dSetXValuesRow()	Set the x coordinate reference row.
XrtMvcViewXrt3dGetYValuesColumn()	Get the column of data used to specify the y coordinates of the data.
XrtMvcViewXrt3dSetYValuesColumn()	Set the y coordinate reference column.

A.4.3 Table View Object Services

Register Class	
XrtMvcViewXrtTableRegister()	Register the Table View class and callbacks.
Attributes	
XrtMvcViewXrtTableGetDisplayFormat()	Retrieve the string used to format displayed data values.
XrtMvcViewXrtTableSetDisplayFormat()	Set the format string for displaying data values.
XrtMvcViewXrtTableGetScanFormat()	Get the format used to interpret values entered in a table cell.
XrtMvcViewXrtTableSetScanFormat()	Set the format string to be used to interpret values entered in a table cell.
XrtMvcViewXrtTableGetUndefinedString()	Get the string printed when an undefined value is displayed.
XrtMvcViewXrtTableSetUndefinedString()	Set the string to be displayed for an undefined element of the data.

A.5 Utility Functions

Utility Functions	
XrtMvcWidgetToView()	Get the View associated with a particular widget.
XrtMvcIntersectRanges()	Determine the intersection of two ranges.

B

Procedures and Methods Reference

This appendix lists the MVC Kit procedures and methods in alphabetical order. Source code for all the routines in this library is located in *$XRTHOME/src/mvc*. If you want to change a source routine, it should be copied to another directory (and linked ahead of the MVC library if its name is not changed).

XrtMvcCreate()

Creates and returns an MVC object.

```
XrtMvc
XrtMvcCreate(void)
```

XrtMvcDataAppendColumn()

Appends a column of numbers to the MVC object's data.

```
int
XrtMvcDataAppendColumn(
    XrtMvc          mvc,
    double          *values,
    int             num_values
    )
```

The array of *num_values* double *values* is added to the data of the MVC as a new column after the last existing column.

The number of the new column is returned.

XrtMvcDataAppendRow()

Append a row of numbers to the MVC object's data.

```
int
XrtMvcDataAppendRow(
    XrtMvc          mvc,
    double          *values,
    int             num_values
    )
```

The array of *values* is appended as a new row at the end of the MVC's data.

The return value is the position of the new row in the MVC data structure.

XrtMvcDataDeleteAll()

All data in the MVC object is deleted.

```
void
XrtMvcDataDeleteAll(
    XrtMvc          mvc
    )
```

This procedure only deletes the data, so labels must be deleted separately.

XrtMvcDataDeleteColumns()

Delete columns of data from the MVC object.

```
void
XrtMvcDataDeleteColumns(
    XrtMvc          mvc,
    int             start_column,
    int             num_columns
    )
```

num_columns columns of data, counting from position *start_column*, will be deleted from the MVC data structure.

XrtMvcDataDeleteRows()

Delete rows of data from the MVC object.

```
void
XrtMvcDataDeleteRows(
    XrtMvc          mvc,
    int             start_row,
    int             num_rows
    )
```

num_rows rows of data, beginning at row position *start_row*, are deleted.

XrtMvcDataGetColumnCount()

Returns the number of columns of data, which is the length of the longest row.

```
int
XrtMvcDataGetColumnCount(
    XrtMvc          mvc
    )
```

XrtMvcDataGetRowCount()

Returns the number of rows of data owned by the MVC object.

```
int
XrtMvcDataGetRowCount(
    XrtMvc          mvc
    )
```

XrtMvcDataGetValue()

Returns the value at location *(row, column)* in the MVC's data.

```
double
XrtMvcDataGetValue(
    XrtMvc          mvc,
    int             row,
    int             column
    )
```

XrtMvcDataInsertColumn()

Insert a column of numbers into the MVC's data.

```
void
XrtMvcDataInsertColumn(
    XrtMvc          mvc,
    int             target_column,
    double          *values,
    int             num_values
    )
```

An array of numbers, *values*, of length *num_values* is inserted into the MVC's data at position *target_column*.

XrtMvcDataInsertRow()

Insert a row of data into the MVC's data.

```
void
XrtMvcDataInsertRow(
    XrtMvc          mvc,
    int             target_row,
    double          *values,
    int             num_values
    )
```

num_values numbers in array *values* are inserted as a new row at position *target_row* into the MVC's data.

XrtMvcDataMoveColumns()

Move a set of columns to a new location.

```
void
XrtMvcDataMoveColumns(
    XrtMvc          mvc,
    int             target_column,
    int             source_column,
    int             num_columns
    )
```

A group of *num_columns* columns of data, beginning from *source_column*, are moved to begin at position *target_column*.

XrtMvcDataMoveRows()

Move a set of rows to a new location in the MVC's data.

```
void
XrtMvcDataMoveRows(
    XrtMvc          mvc,
    int             target_row,
    int             source_row,
    int             num_rows
    )
```

A group of *num_rows* rows, starting with *source_row*, are moved to begin at *target_row*.

XrtMvcDataReadAscii()

Read an ASCII file as rows of data for the MVC object.

```
void
XrtMvcDataReadAscii(
    XrtMvc          mvc,
    FILE            *fp,
    char            delimiter
    )
```

The open file given by *fp* is read to the End-of-File. Each line of the file is parsed as a series of numbers separated by the *delimiter* character and appended to the MVC as a row of data. If there is data already, these rows will be appended to the existing data.

Labels cannot be read in by this method.

XrtMvcDataSetValue()

Set the cell at the location *(row, column)* in the MVC data to *value*.

```
void
XrtMvcDataSetValue(
    XrtMvc          mvc,
    int             row,
    int             column,
    double          value
    )
```

XrtMvcDataWriteAscii()

Write the contents of the MVC data structure to an ASCII file.

```
Boolean
XrtMvcDataWriteAscii(
    XrtMvc          mvc,
    FILE            *fp,
    char            delimiter,
    String          format,
    XrtMvcRange     *range_ptr
    )
```

The data is written to the file *fp* one row per line. Each value is formatted according to *printf*-style *format*, and values in a row are separated by the character *delimiter*.

XrtMvcGetUpdate()

Returns the update status of the MVC object.

```
Boolean
XrtMvcGetUpdate(
    XrtMvc          mvc
    )
```

`True` means that the MVC will issue updates to all Views when data is changed. If the update status is `False`, updates will be queued until it is again set `True`.

XrtMvcGetUserData()

Returns the user data pointer of the MVC object.

```
XtPointer
XrtMvcGetUserData(
    XrtMvc          mvc
    )
```

XrtMvcGetViewCount()

Returns the number of Views currently associated with the MVC.

```
int
XrtMvcGetViewCount(
    XrtMvc          mvc
    )
```

XrtMvcIntersectRanges()

Determines the range that is the intersection of two given ranges.

```
Boolean
XrtMvcIntersectRanges(
    XrtMvcRange     *range1,
    XrtMvcRange     *range2,
    XrtMvcRange     *overlap_range
    )
```

The ranges *range1* and *range2* are checked to determine if they intersect. *XrtMvcIntersectRanges()* returns `True` if the ranges intersect, and *overlap_range* contains the bounds of the intersection.

XrtMvcLabelAppendColumn()

Adds the label *column_label* to the end of the list of column labels.

```
void
XrtMvcLabelAppendColumn(
    XrtMvc          mvc,
    String          column_label
    )
```

XrtMvcLabelAppendRow()

Adds the label *row_label* to the end of the list of row labels.

```
void
XrtMvcLabelAppendRow(
    XrtMvc          mvc,
    String          row_label
    )
```

XrtMvcLabelDeleteAll()

All row and column labels are deleted from the MVC object.

```
void
XrtMvcLabelDeleteAll(
    XrtMvc          mvc
    )
```

XrtMvcLabelDeleteColumns()

Deletes a set of column labels.

```
void
XrtMvcLabelDeleteColumns(
    XrtMvc          mvc,
    int             start_column,
    int             num_labels
    )
```

num_labels column labels, beginning in position *start_column* are deleted.

XrtMvcLabelDeleteRows()

Deletes *num_labels* row labels beginning at *start_row*.

```
void
XrtMvcLabelDeleteRows(
    XrtMvc          mvc,
    int             start_row,
    int             num_labels
    )
```

XrtMvcLabelInsertColumn()

Inserts the label *label* at column *target_column*.

```
void
XrtMvcLabelInsertColumn(
    XrtMvc          mvc,
    int             target_column,
    String          label
    )
```

XrtMvcLabelInsertRow()

Inserts the new label *label* at position *target_row* of the row labels.

```
void
XrtMvcLabelInsertRow(
    XrtMvc          mvc,
    int             target_row,
    String          label
    )
```

XrtMvcLabelMoveColumns()

Moves a set of column labels.

```
void
XrtMvcLabelMoveColumns(
    XrtMvc          mvc,
    int             target_column,
    int             source_column,
    int             num_columns
    )
```

A sequence of *num_columns* column labels beginning at *source_column* are moved to position *target_column*.

XrtMvcLabelMoveRows()

Move a group of row labels.

```
void
XrtMvcLabelMoveRows(
    XrtMvc          mvc,
    int             target_row,
    int             source_row,
    int             num_rows
    )
```

num_rows row labels starting at *source_row* are moved to *target_row*.

XrtMvcSetUpdate()

Sets the View update policy.

```
void
XrtMvcSetUpdate(
    XrtMvc          mvc,
    Boolean         update
    )
```

If the update attribute of the MVC is set to `False`, changes will be buffered instead of being sent to all Views. When update is `True`, changes will be sent immediately to all Views.

XrtMvcSetUserData()

Sets the user data of the MVC object.

```
void
XrtMvcSetUserData(
    XrtMvc          mvc,
    XtPointer       user_data
    )
```

XrtMvcViewCreate()

Creates a new View object.

```
XrtMvcView
XrtMvcViewCreate(
    XrtMvc              mvc,
    XrtMvcViewClass     view_class,
    Widget              widget,
    XtPointer           view_user_data
    )
```

A View object of class *view_class* will be created and associated with *widget*. *view_user_data* will be passed to the View initialization procedure.

XrtMvcViewDestroy()

Destroys the given View object.

```
void
XrtMvcViewDestroy(
    XrtMvcView          view
    )
```

XrtMvcViewGetDisplayPolicy()

Returns the display policy of the View.

```
XrtMvcDisplayPolicy
XrtMvcViewGetDisplayPolicy(
    XrtMvcView          view
    )
```

Display policy can have one of the following values:

`XRTMVC_DISPLAY_ALL`	All of the data in the MVC is displayed.
`XRTMVC_DISPLAY_RANGE`	The View displays only the data within the scope of its range attribute.
`XRTMVC_DISPLAY_NOTHING`	No data is displayed.

XrtMvcViewGetEditability()

Returns `True` if the data in the View can be edited.

```
Boolean
XrtMvcViewGetEditability(
    XrtMvcView          view
    )
```

XrtMvcViewGetRange()

Returns the range within which the View is displaying the MVC data.

```
XrtMvcRange
*XrtMvcViewGetRange(
    XrtMvcView          view
    )
```

XrtMvcViewGetUserData()

Returns a pointer to the View's user data.

```
XtPointer
XrtMvcViewGetUserData(
    XrtMvcView          view
    )
```

Note that this is not the same as the *view_user_data* attribute that is set in the *XrtMvcViewCreate()* procedure.

XrtMvcViewGetWidget()

Returns the widget associated with the View.

```
Widget
XrtMvcViewGetWidget(
    XrtMvcView          view
    )
```

XrtMvcViewRegisterAllClasses()

Registers all View object classes.

```
void
XrtMvcViewRegisterAllClasses(void)
```

XrtMvcViewRegisterClass()

Register a user-defined View object class.

```
XrtMvcViewClass
XrtMvcViewRegisterClass(
    XrtMvcViewInitializeProcinitialize,
    XrtMvcViewDestroyProc   destroy,
    XrtMvcViewUpdateProc    update,
    XrtMvcViewChangeEditProcchange_edit
    )
```

initialize, *destroy*, *update* and *change_edit* are user-defined functions the MVC will use to control a user-defined View.

XrtMvcViewSetDisplayPolicy()

Set the display policy of the View.

```
void
XrtMvcViewSetDisplayPolicy(
    XrtMvcView          view,
    XrtMvcDisplayPolicy display_policy
    )
```

The display policy controls the extent of the MVC data displayed by the View. It can take the following values:

XRTMVC_DISPLAY_ALL	All of the data in the MVC is displayed.
XRTMVC_DISPLAY_RANGE	The View displays only the data within the scope of its range attribute.
XRTMVC_DISPLAY_NOTHING	No data is displayed.

XrtMvcViewSetEditability()

Sets the editability attribute of the View.

```
void
XrtMvcViewSetEditability(
    XrtMvcView          view,
    Boolean             editable
    )
```

If the editability of the View is set to `True` then the values in the View may be adjusted to modify the MVC data.

XrtMvcViewSetRange()

Sets the range attribute of the View.

```
void
XrtMvcViewSetRange(
    XrtMvcView          view,
    XrtMvcRange         *range
    )
```

If the display policy is set to `XRTMVC_DISPLAY_RANGE` then *range* will define the extent of the MVC data which is displayed by the View.

XrtMvcViewSetUserData()

Set the user data attribute of the View.

```
void
XrtMvcViewSetUserData(
    XrtMvcView          view,
    XtPointer           user_data
    )
```

XrtMvcViewXrt3dGetXValuesRow()

Returns the row in the MVC data which is being used to provide X coordinates for the base grid in an XRT/3d View.

```
int
XrtMvcViewXrt3dGetXValuesRow(
    XrtMvcView          view
    )
```

If this attribute has not been set, it will contain the value `XRTMVC_UNSPECIFIED_ROW`, and the X coordinates will be the column number in the MVC data array.

XrtMvcViewXrt3dGetYValuesColumn()

Returns the column in the MVC which is being used to specify Y coordinates for the base grid in an XRT/3d View.

```
int
XrtMvcViewXrt3dGetYValuesColumn(
    XrtMvcView          view
    )
```

If this attribute has not been set, it will contain the value `XRTMVC_UNSPECIFIED_COLUMN`, and the Y coordinates will be the row number in the MVC data array.

XrtMvcViewXrt3dRegister()

Register the XRT/3d View object.

```
void
XrtMvcViewXrt3dRegister(void)
```

XrtMvcViewXrt3dSetXValuesRow()

Set the row of MVC data used to define the X coordinates of grid points in an XRT/3d View.

```
void
XrtMvcViewXrt3dSetXValuesRow(
    XrtMvcView      view,
    int             row
    )
```

This attribute may be assigned the value `XRTMVC_UNSPECIFIED_ROW` to return the X coordinates to a simple incremental numbering based on the column number.

XrtMvcViewXrt3dSetYValuesColumn()

Set the column of MVC data used to define the Y coordinates of grid points in an XRT/3d View.

```
void
XrtMvcViewXrt3dSetYValuesColumn(
    XrtMvcView      view,
    int             column
    )
```

This attribute may be assigned the value `XRTMVC_UNSPECIFIED_COLUMN` to return the Y coordinates to a simple incremental numbering based on the row number.

XrtMvcViewXrtGraphGetXValuesRow()

Get the row of MVC data used to define the X coordinates of graph points in an XRT/graph View.

```
int
XrtMvcViewXrtGraphGetXValuesRow(
    XrtMvcView      view
    )
```

If this attribute has not been set, it will contain the value `XRTMVC_UNSPECIFIED_ROW`, and the X coordinates will be the column number in the MVC data array.

XrtMvcViewXrtGraphRegister()

Register the XRT/graph View object.

```
void
XrtMvcViewXrtGraphRegister(void)
```

XrtMvcViewXrtGraphSetXValuesRow()

Set the row of data in the MVC which is used to provide X coordinates for graphing the displayed data.

```
void
XrtMvcViewXrtGraphSetXValuesRow(
    XrtMvcView      view,
    int             row
    )
```

This attribute may be assigned the value `XRTMVC_UNSPECIFIED_ROW` to return the X coordinates to a simple incremental numbering based on the column number.

XrtMvcViewXrtTableGetDisplayFormat()

Returns the string which is currently being used to format the MVC data on display to a table.

```
String
XrtMvcViewXrtTableGetDisplayFormat(
    XrtMvcView      view
    )
```

XrtMvcViewXrtTableGetScanFormat()

Returns the string which the table View is using to parse input values.

```
String
XrtMvcViewXrtTableGetScanFormat(
    XrtMvcView      view
    )
```

XrtMvcViewXrtTableGetUndefinedString()

Returns the string that is displayed in the table for values which are undefined.

```
String
XrtMvcViewXrtTableGetUndefinedString(
    XrtMvcView      view
    )
```

XrtMvcViewXrtTableRegister()

Register the XRT/table View object.

```
void
XrtMvcViewXrtTableRegister(void)
```

XrtMvcViewXrtTableSetDisplayFormat()

Set the string used to format values for display in a table.

```
void
XrtMvcViewXrtTableSetDisplayFormat(
    XrtMvcView      view,
    String          str
    )
```

str is a *printf*-style format string which is used to format values which are being displayed in a table View.

XrtMvcViewXrtTableSetScanFormat()

Set the string used to parse user-input values.

```
void
XrtMvcViewXrtTableSetScanFormat(
    XrtMvcView      view,
    String          str
    )
```

str is a *scanf*-style format string used to extract a numeric value from a user's entry in a table cell.

XrtMvcViewXrtTableSetUndefinedString()

Set the string which is displayed in the table for values which are undefined.

```
void
XrtMvcViewXrtTableSetUndefinedString(
    XrtMvcView      view,
    String          str
    )
```

XrtMvcWidgetToView()

Determine the View that is being displayed in a given widget.

```
XrtMvcView
XrtMvcWidgetToView(
    Widget          w,
    XrtMvc          *mvc
    )
```

Returns the View which is being displayed in widget *w*, and sets *mvc* to point to the MVC structure which owns that View. If the widget does not own that View, the return value is `NULL`.

Index

A

ANSI C, compatibility with the MVC Kit 6, 23
attributes
  display format 21
  display policy 20
  range 20
  reference column 22
  reference row 22
  scan format 21
  undefined string 21
  update 13, 14
  user data
    MVC object 13
    View object 20
authorizing XRT programs 6

C

C++
  compatibility with the MVC Kit 6
compiling MVC Kit programs 6
custom views 23

D

data
  adding 15
  local copies 21
  reading from file 15
  referenced 21
  static 17
  summary of procedures 16
  type of 15
  undefined 15
  writing to file 17
display format 21
display policy 20

E

editability 21
editing
  editability attribute 21
  local copies of data 21
  table restrictions 21
editing data
  reference row values 22
environment variables, XRTHOME 5

K

K&R C, incompatibility with library source 23

L

labels
  adding 15
  summary of procedures 16
  type of 15

M

MVC object
  attributes 13
  creating 14
  data structure 15
  update 13
  user data 13

R

range
  attribute 20
  displayed 20
  intersections 14
reference column
  3d View 22
  monotonicity 22
reference row
  3d View 22
  graph View 22
  monotonicity 22
registering
  custom classes 19, 23
  standard classes 19

S

sample1.c
  discussion of 6

listing of sample.dat file 7
SAMPLE2.C
XRT/3d version 12
XRT/graph version 11
XRT/table version 10
sample2.c
discussion of 8
listing of sample.dat file 7
XRT/3d version 12
XRT/graph version 11
XRT/table version 10
scan format 21
source code 23
static data 17

U

undefined string 21
upate attribute 14
update attribute 13, 14

V

View class
registering 19
View object
attributes 20
creating 8
display policy 20
editability 21
examples 8
XRT/3d 12
XRT/graph 11
XRT/table 10
reference column 22
reference row 22
user data 20
XRT/3d View 22
XRT/graph View 21
XRT/table View 21

W

widget
getting View 14

X

xrt_auth program
use of, see XRT/graph table or 3d manual
XRTHOME environment variable 5
XrtMvcCreate() 14, 35
XrtMvcDataAppendColumn() 14, 15, 35
XrtMvcDataAppendRow() 14, 15, 35
XrtMvcDataDeleteAll() 36
XrtMvcDataDeleteColumns() 36
XrtMvcDataDeleteRows() 36
XrtMvcDataGetColumnCount() 13, 36
XrtMvcDataGetRowCount() 14, 36
XrtMvcDataGetValue() 37
XrtMvcDataInsertColumn() 37
XrtMvcDataInsertRow() 37
XrtMvcDataMoveColumns() 37
XrtMvcDataMoveRows() 38
XrtMvcDataReadAscii() 14, 15, 17, 38
XrtMvcDataSetValue() 14, 15, 38
XrtMvcDataWriiteAscii() 17
XrtMvcDataWriteAscii() 38
XrtMvcGetUpdate() 13, 39
XrtMvcGetUserData() 13, 39
XrtMvcGetViewCount() 39
XrtMvcIntersectRanges() 14, 39
use in custom Views 25
XrtMvcLabelAppendColumn() 39
XrtMvcLabelAppendRow() 15, 39
XrtMvcLabelDeleteAll() 40
XrtMvcLabelDeleteColumns() 40
XrtMvcLabelDeleteRows() 40
XrtMvcLabelInsertColumn() 40
XrtMvcLabelInsertRow() 15, 40
XrtMvcLabelMoveColumns() 41
XrtMvcLabelMoveRows() 41
XrtMvcSetUpdate() 41
buffering updates 14
XrtMvcSetUserData() 41
XrtMvcViewCreate() 24, 42
XrtMvcViewDestroy() 24, 42
XrtMvcViewGetDisplayPolicy() 20, 42
XrtMvcViewGetEditability() 20, 42
XrtMvcViewGetRange() 20, 42
XrtMvcViewGetUserData() 20, 43
XrtMvcViewGetWidget() 43
XrtMvcViewRegisterAllClasses() 19, 43
XrtMvcViewRegisterClass() 43
creating a custom View 19, 23
XrtMvcViewSetDisplayPolicy() 20, 43
XrtMvcViewSetEditability() 20, 24, 44
XrtMvcViewSetRange() 20, 44
XrtMvcViewSetUserData() 20, 44
XrtMvcViewXrt3dGetXValuesRow() 22, 44
XrtMvcViewXrt3dGetYValuesColumn() 22, 44
XrtMvcViewXrt3dRegister() 19, 45
XrtMvcViewXrt3dSetXValuesRow() 22, 45
XrtMvcViewXrt3dSetYValuesColumn() 22, 45
XrtMvcViewXrtGraphGetXValuesRow() 22, 45
XrtMvcViewXrtGraphRegister() 19, 45
XrtMvcViewXrtGraphSetXValuesRow() 22, 46
XrtMvcViewXrtTableGetDisplayFormat() 21, 46
XrtMvcViewXrtTableGetScanFormat() 21, 46
XrtMvcViewXrtTableGetUndefinedString() 21, 46
XrtMvcViewXrtTableRegister() 19, 46
XrtMvcViewXrtTableSetDisplayFormat() 21, 46
XrtMvcViewXrtTableSetScanFormat() 21, 47
XrtMvcViewXrtTableSetUndefinedString() 21, 47
XrtMvcWidgetToView() 14, 47