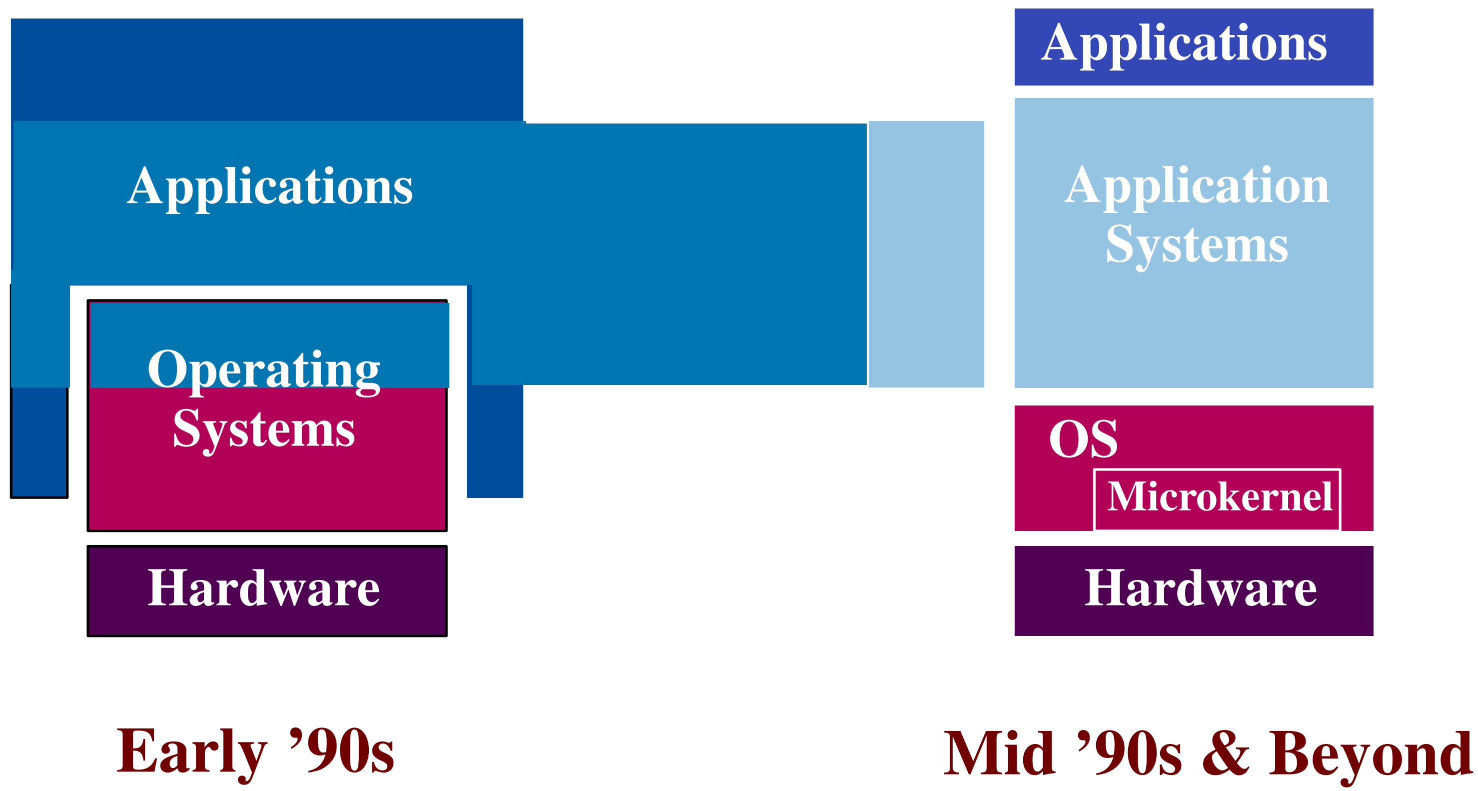
For: macphail
Printed on: Thu, Feb 2, 1995 15:50:32
From book: Programming
Document: Intro
Last saved on: Thu, Feb 2, 1995 11:32:44
Document: ticpres.h
Last saved on: Wed, Feb 1, 1995 08:46:34
Document: ticview.h
Last saved on: Wed, Feb 1, 1995 08:49:02
Document: ticcomma.h
Last saved on: Tue, Jan 31, 1995 15:04:07
Document: ticselec.h
Last saved on: Tue, Jan 31, 1995 15:17:28
Document: ticmodel.h
Last saved on: Wed, Feb 1, 1995 09:05:53
Document: ticpres.C
Last saved on: Wed, Feb 1, 1995 09:08:02
Document: ticview.C
Last saved on: Tue, Jan 31, 1995 15:58:13
Document: ticcomma.C
Last saved on: Tue, Jan 31, 1995 15:43:01
Document: ticselec.C
Last saved on: Tue, Jan 31, 1995 16:01:38
Document: ticmodel.C
Last saved on: Tue, Jan 31, 1995 16:08:10
(...)

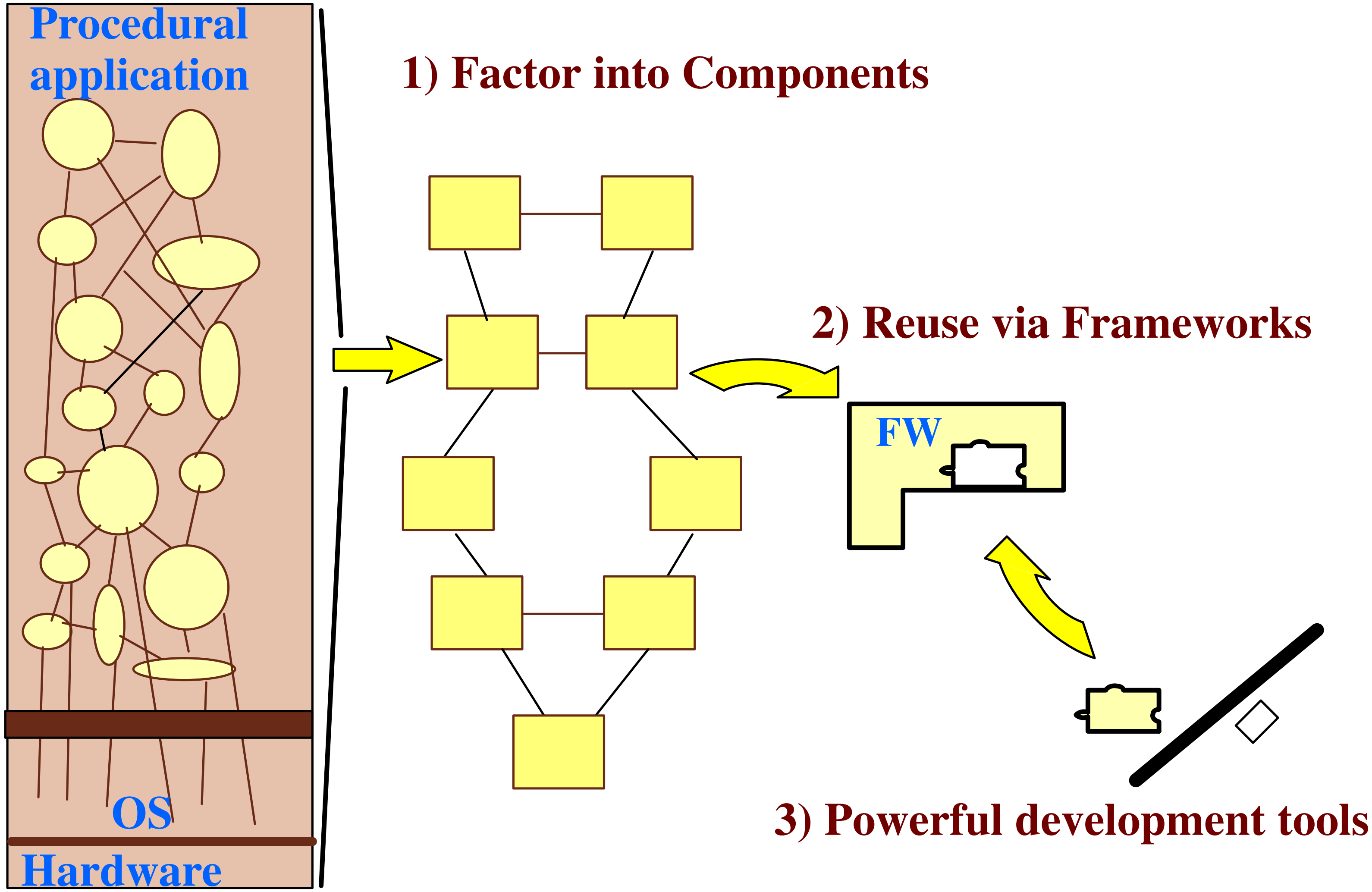
Building Applications with Frameworks

Presented by: Peg MacPhail
IBM Austin, Tx.

Application System Emerges



Taligent Architecture



CommonPoint™: Architecture Overview

Application Frameworks (18)

- Base Application Framework (1)
- Text Editor Framework (1)
- Compound Doc Frameworks (8)
- Graphics Editor Framework (1)
- User Workspace (3)
- User–Interface Frameworks (4)

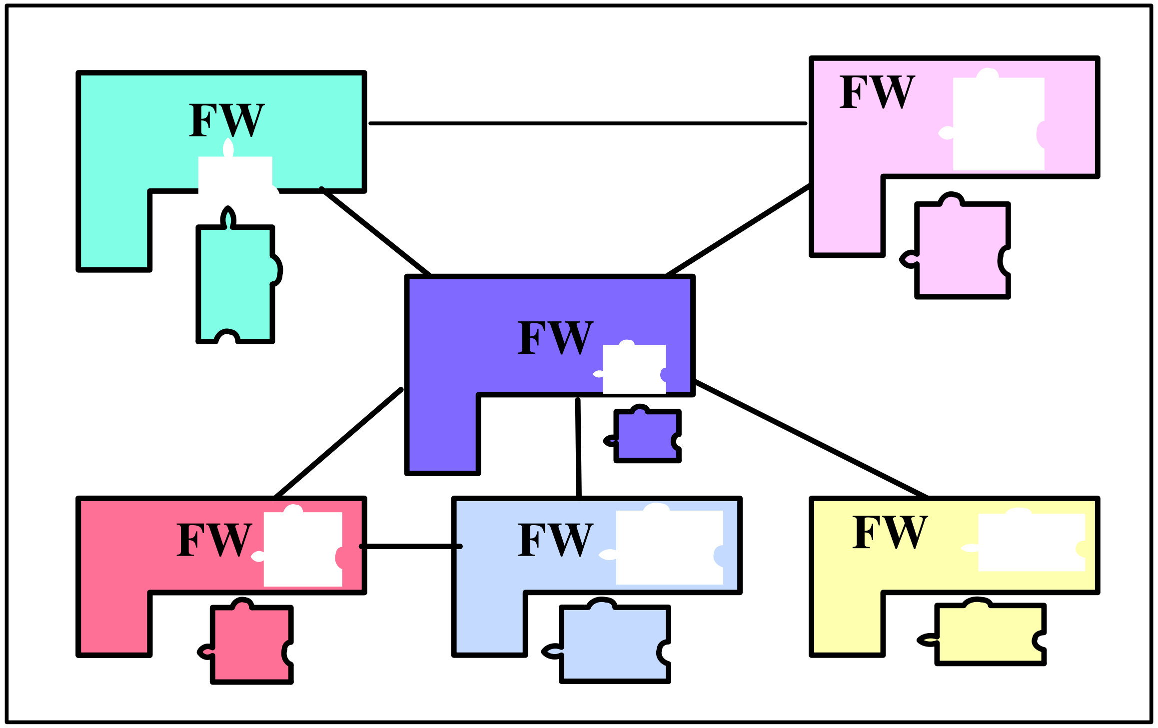
Domain Frameworks (21)

- 2–D Graphics Frameworks (4)
- 3–D Graphics Frameworks (2)
- Multimedia Frameworks (5)
- Basic Text Framework (3)
- International Text Frameworks (7)

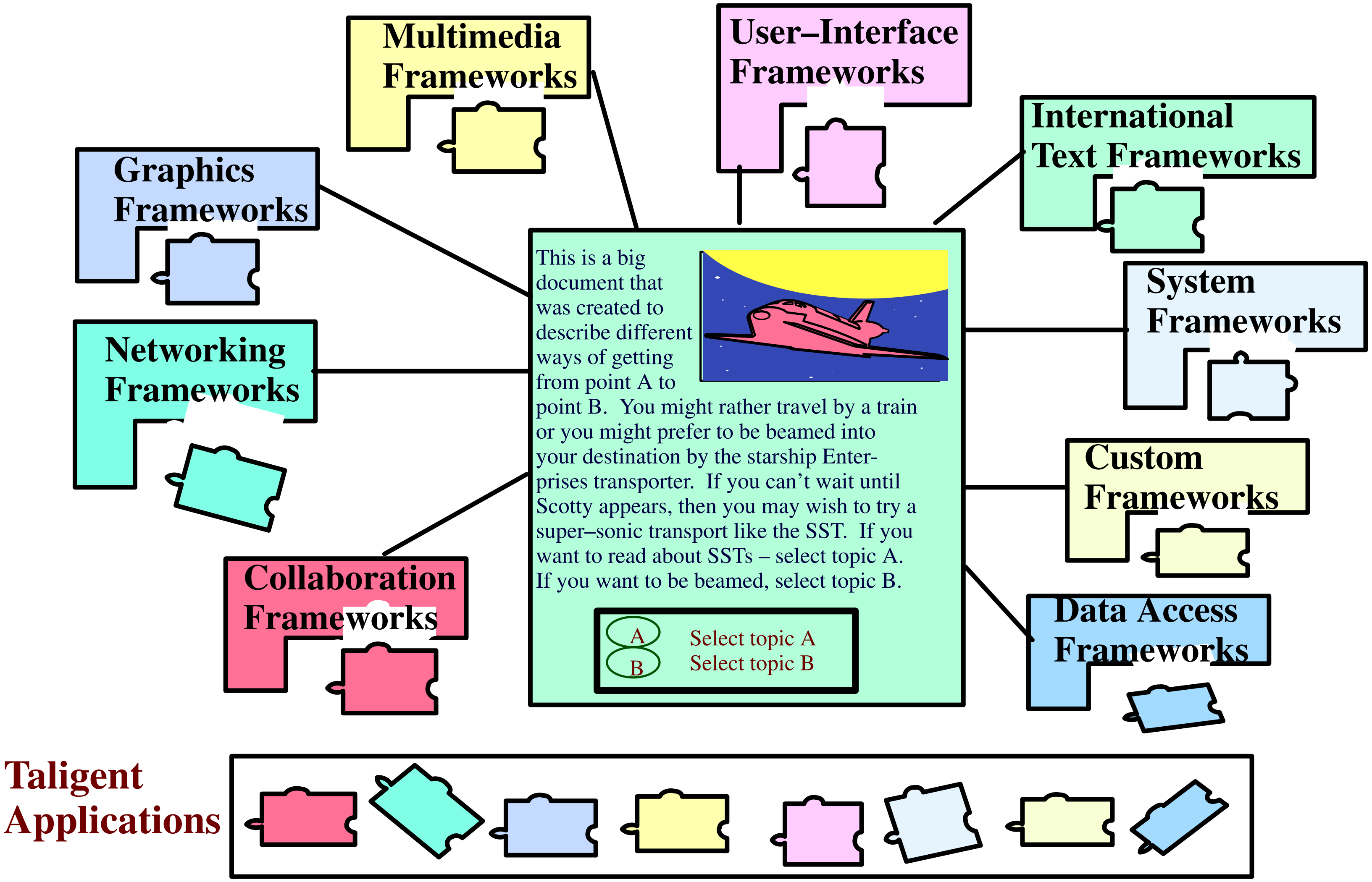
Support Frameworks (46)

- Application Services Frameworks (9)
- Distributed Computing Frameworks (4)
- Foundation Objects (6)
- Software Portability Frameworks (24)
- Data Management Frameworks (3)

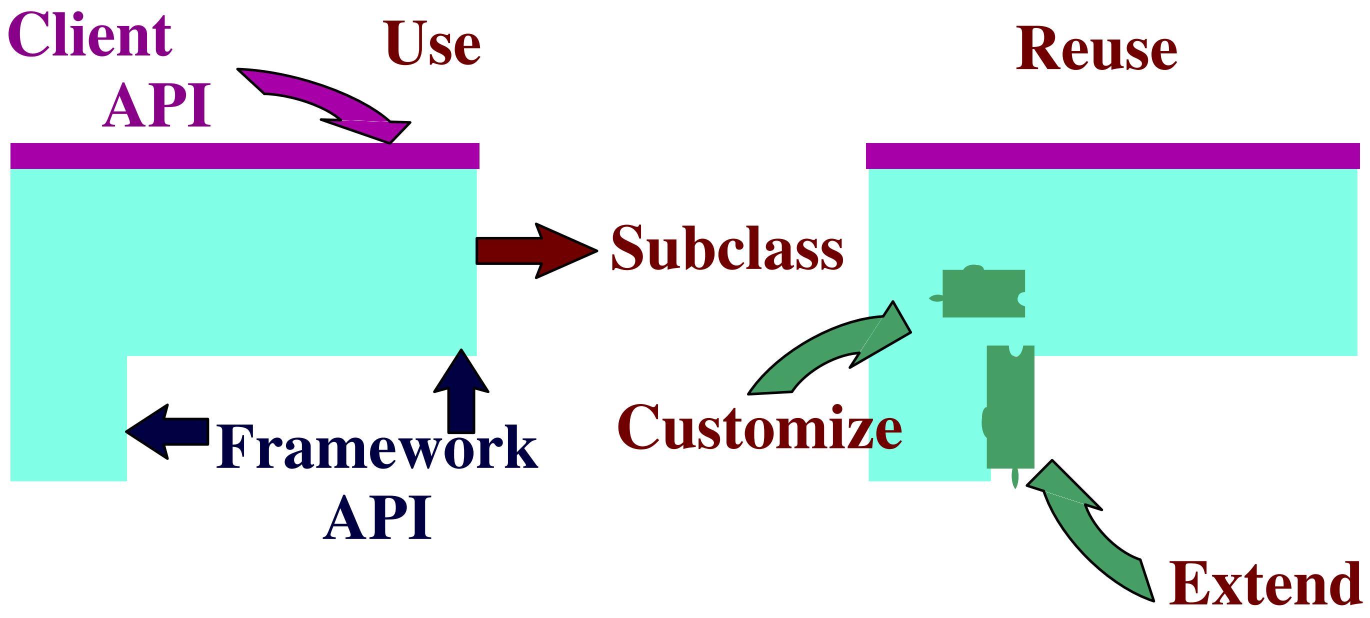
Building on Frameworks



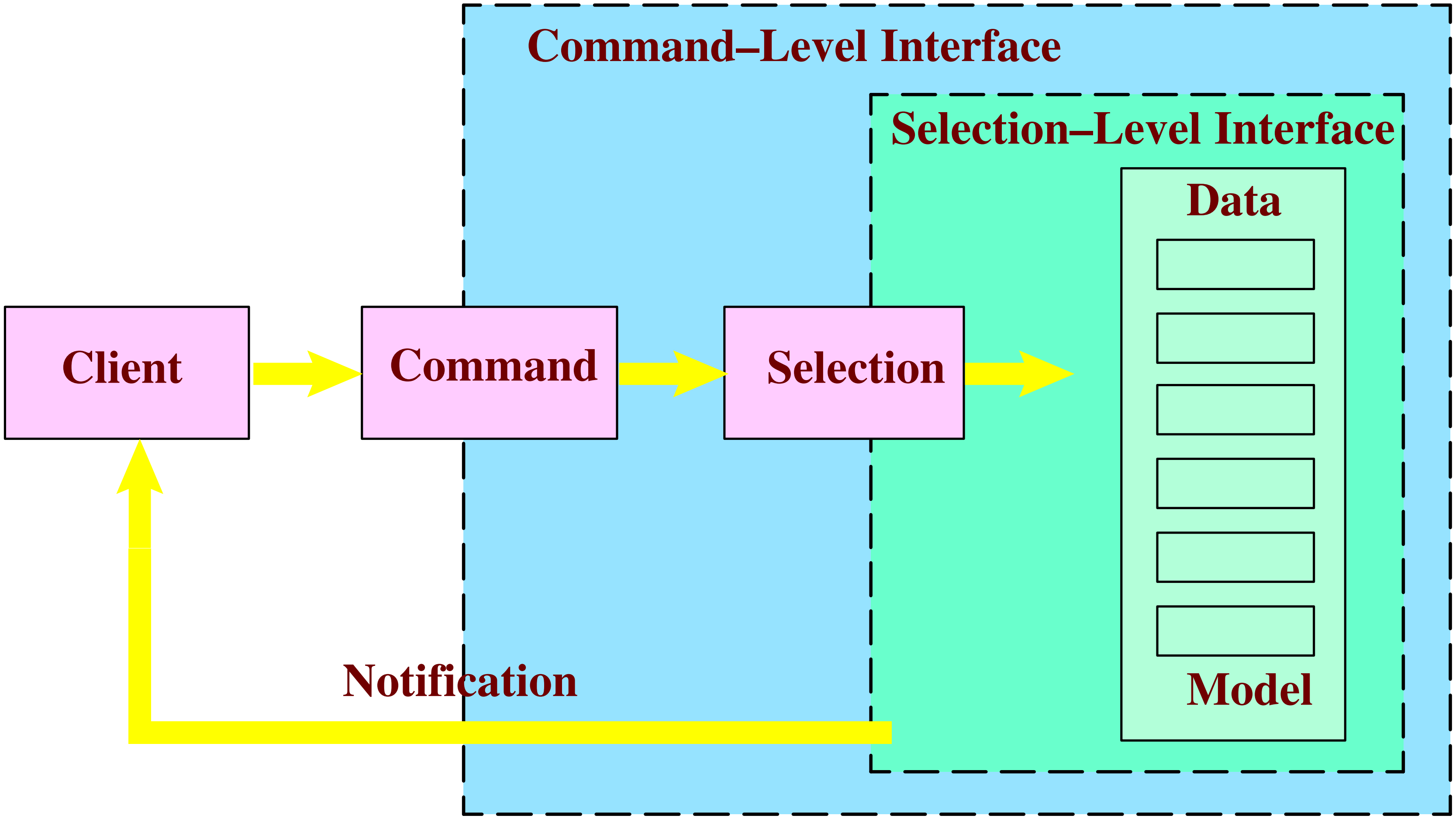
Extensive Functionality . . . with Reuse



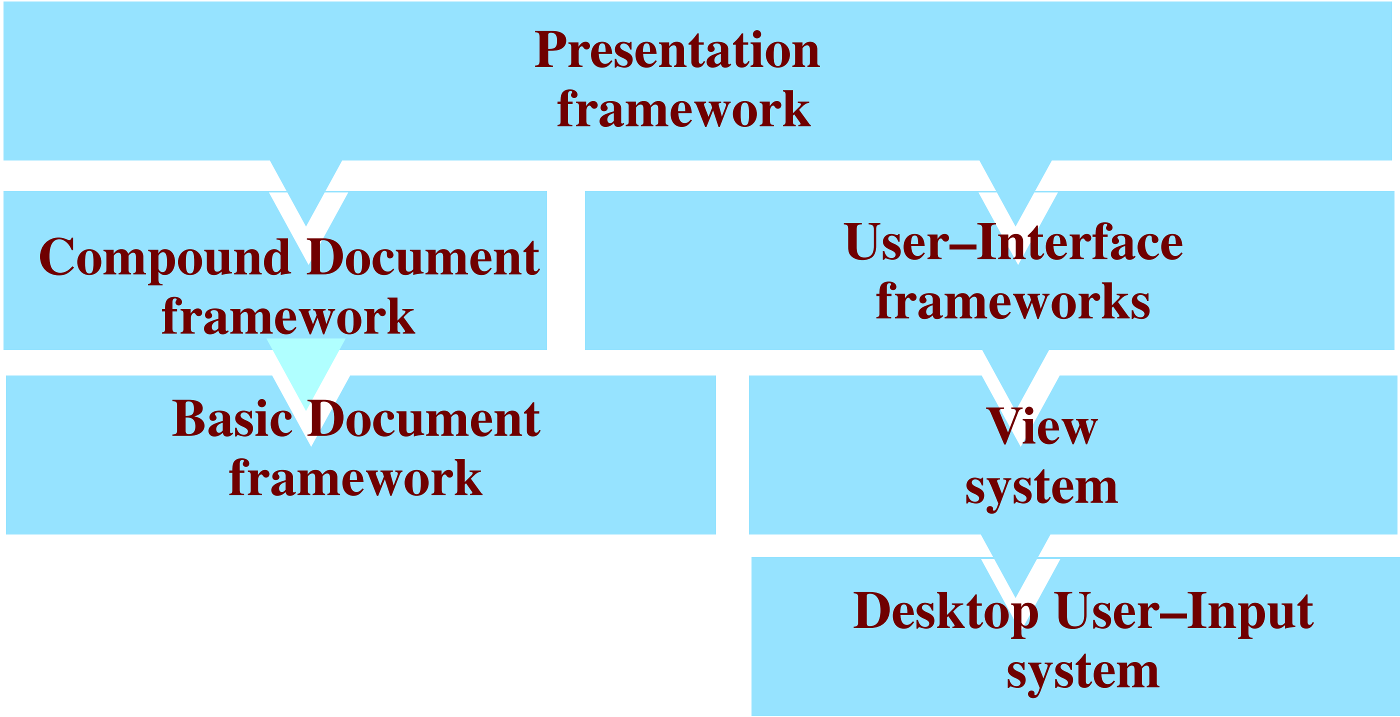
Framework


Client
API
Use
Reuse
Subclass
Framework
API
Customize
Extend

Model Interfaces



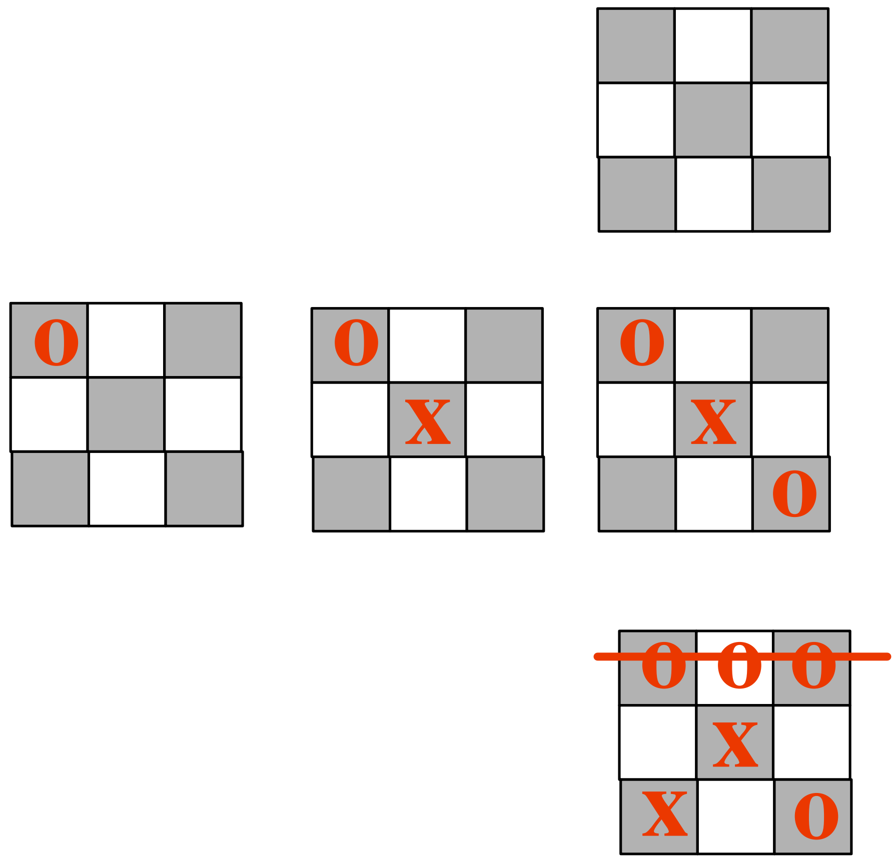
CommonPoint™ frameworks supporting model interfaces



A programming example

TicTacToe

TicTacToe



TicTacToe – A Simple Application

Objectives

- Illustrate how to use the Presentation Frameworks with a simple programming example.
- Show the Taligent programming style.

Caveats

- This program contains 263 lines of code. Most of this code is boiler–plate code which was copied from another example.
- Frequently, successful applications will start by "copying" another application's design and code.

TicTacToe – Source list

TBoardPresenter	**The presenter is responsible for managing all the "presentable" aspects of a document component. This includes views, menus, palettes, dialogs, etc. This TBoardPresenter will add an extended menu and will create the TicTacToe game board.**	**ticpres.h** **ticpres.C**
TPlayCommand	**TPlayCommand will set the letter of a square referred to by a TTicSelection.**	**ticcomma.h** **ticcomma.C**
TNewGameCommand	**TNewGameCommand will restart the game.**	

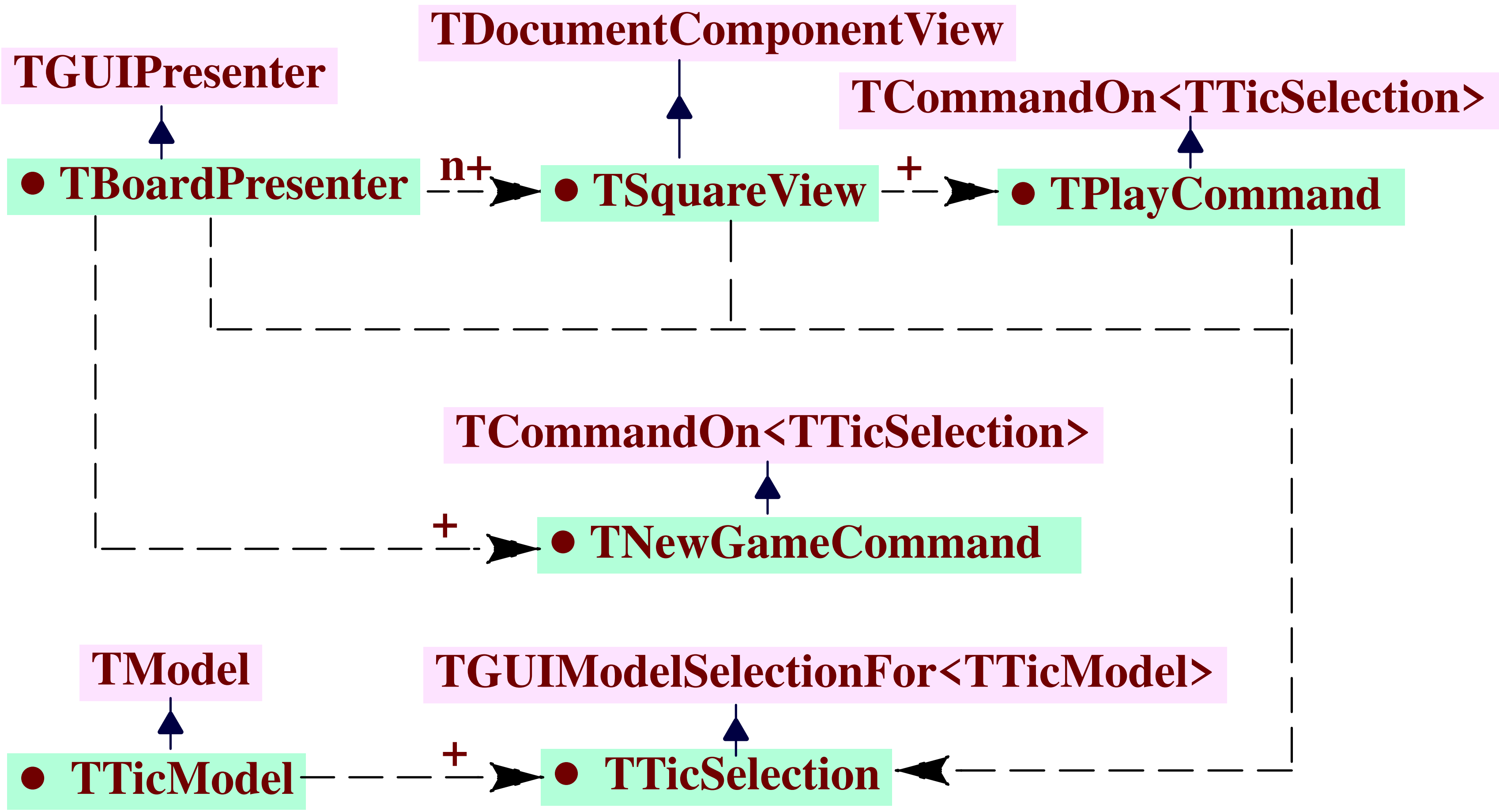
TicTacToe – Source list

TSquareView	TSquareView is responsible for drawing to the view area. This includes setting the background color and drawing the play letter. TSquareView has a method which is called when the player presses the left mouse button to select a square.	ticview.h ticview.C
TTicSelection	TTicSelection is the selection returned by TTicModel::CreateSelection(). The TTicSelection can be on an individual square, on the whole model or on none of the model (an “empty” selection).	ticselec.h ticselec.C
TTicModel	The TTicModel will contain an array of 9 characters representing each play of the game. TTicModel is independent of the visual presentation.	ticmodel.h ticmodel.C

TicTacToe – Source list

This file contains the information used by the Taligent make preprocessor to generate a make command.	**tic.PinkMake**
This file contains defines and pragmas needed to work with the C++ class templates used by this application.	**tempinst.C**
This file contains a script for launching TicTacToe using the CommonPoint™ RunDocument command.	**ticApp**

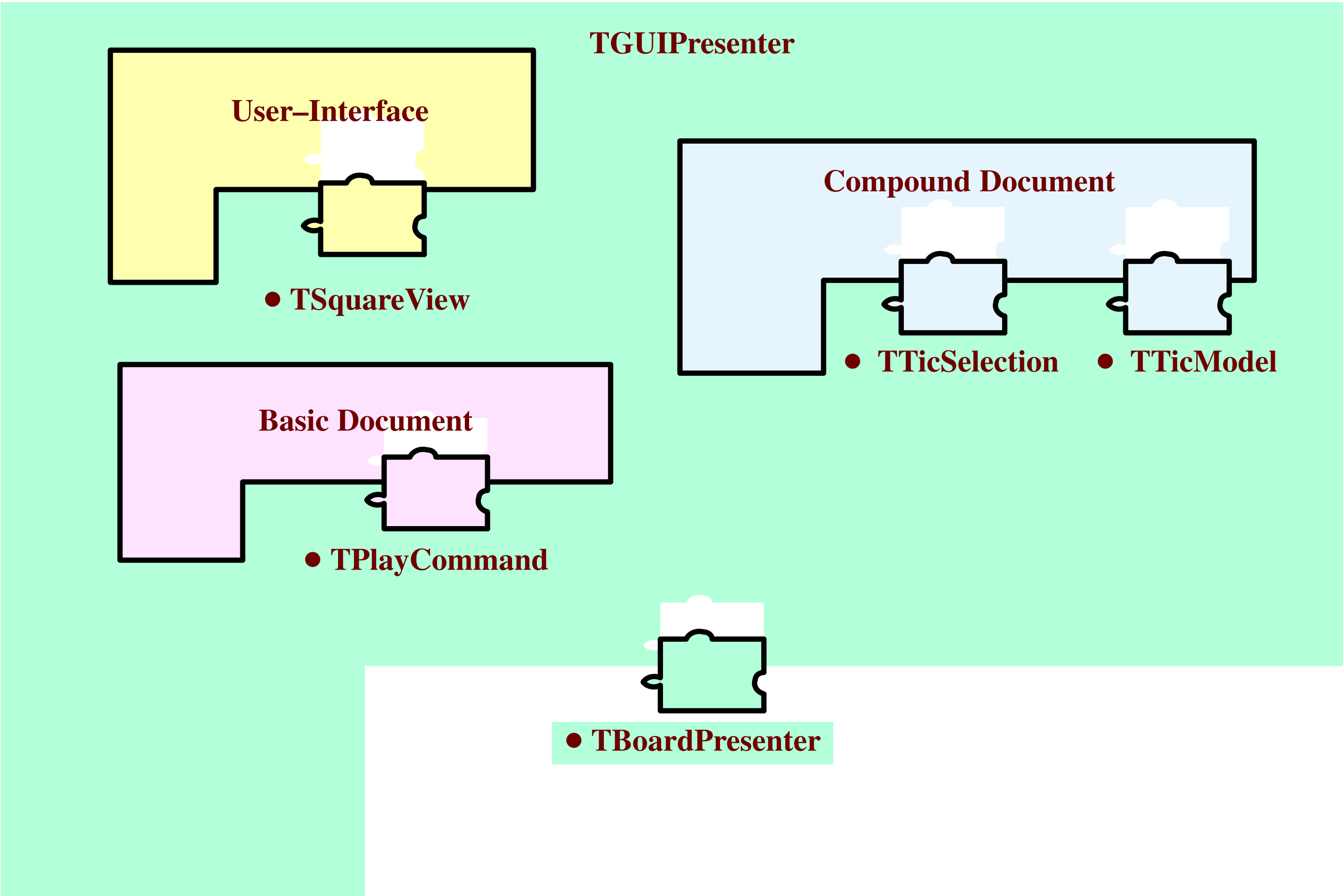
TicTacToe – Abbreviated Class Diagram



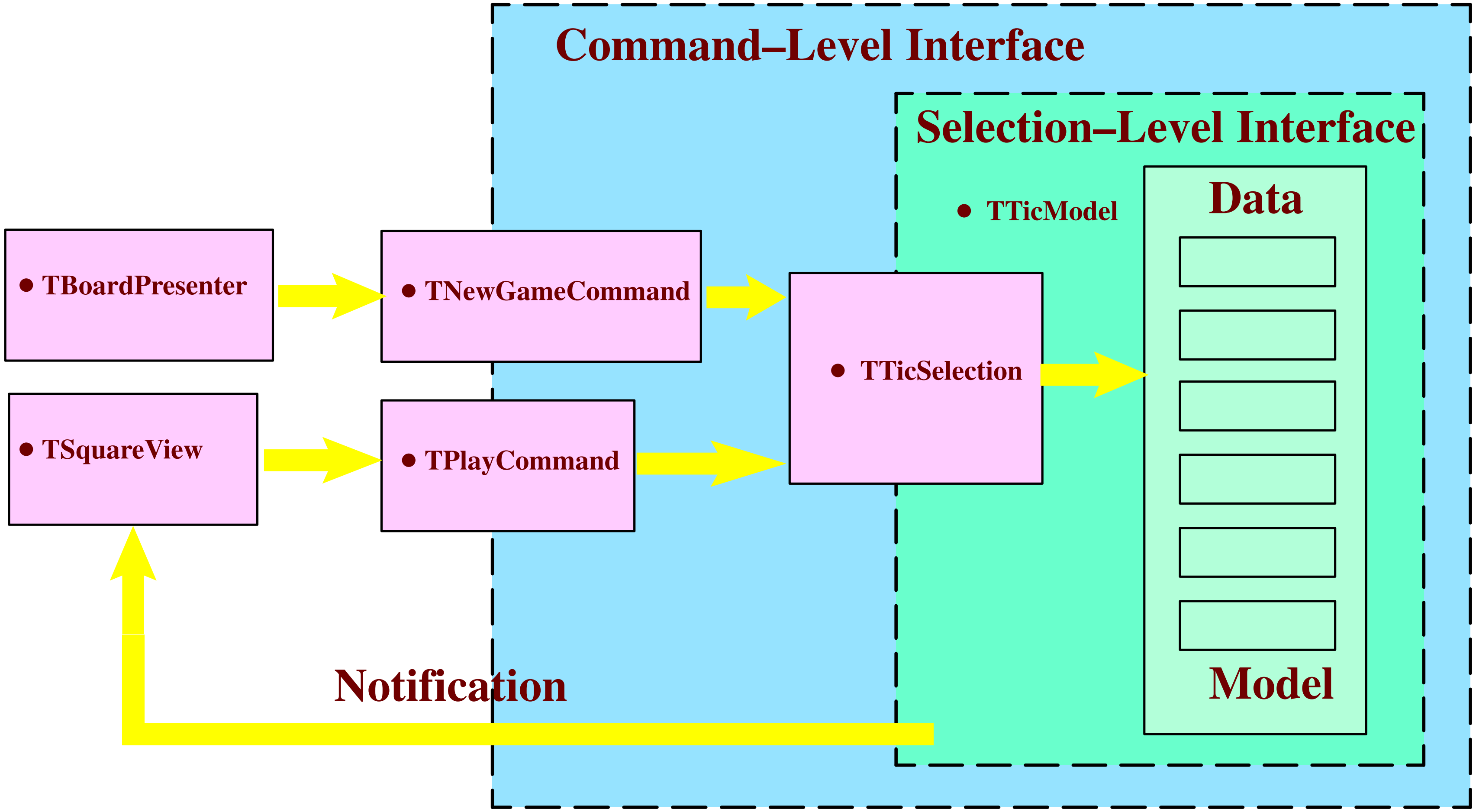
Taligent Class Notation Diagrams

TClassName	Shaded boxes show diagram focus
TClassName	Unshaded boxes out of diagram focus
TClassName ●	This is a developer created class
∆ TClassName	This is an abstract class
ClassA ↑ ClassB	ClassB inherits from ClassA
ClassA ——→ ClassB	Objects of ClassA know of objects of ClassB
ClassA - - - -→ ClassB	Objects of ClassA own objects of ClassB
ClassA - - () - -→ ClassB	ClassA uses ClassB as a parameter
ClassA - - + - -→ ClassB	ClassA creates but doesn't own ClassB objs
ClassA - - + - -→ ClassB	ClassA creates and owns ClassB objects
ClassA —n—→ ClassB	ClassA owns "n" objects of ClassB
ClassA - - - - -→ ClassB	ClassA creates but doesn't own ClassB objs

TicTacToe – Abbreviated Framework Diagram



Model Interfaces



TicTacToe – ticpres.h


```
// ticpres.h

#ifndef IBMSample_TICPRES
#define IBMSample_TICPRES

// Include for Presentation Framework
#ifndef Taligent_GUICOMPOUNDDOCUMENT
  #include "GUICompoundDocument.h"
#endif

// Include for TStandardText and TTextLabel
#ifndef Taligent_LABEL
  #include <Label.h>
#endif

#ifndef IBMSample_TICVIEW
  #include "ticview.h"
#endif
```

TicTacToe – ticpres.h

```
// The presenter is responsible for managing all the "presentable"
// aspects of a document component.  This includes views, menus,
// palettes, dialogs, etc.  This TBoardPresenter will add an
// extended menu through the overridden HandleCreateExtendedMenus
// function and will create the TicTacToe game board through the
// overriden HandleCreateMainView.

class TBoardPresenter : public TGUIPresenter
{
  public:
    // The constructor for TBoardPresenter must take a TGUIBundle
    // which it passes on to the constructor for TGUIPresenter.
    TBoardPresenter (const TGUIBundle&);
    virtual          ~TBoardPresenter ();

    // The following macro must be called when version information
    // is required.  The macro defines a set of versioning utility
    // functions.  Use this macro when versioning is needed, but not
    // the full capabilities of MCollectible. This macro call matches
    // a call to VersionDefinitionsMacro in ticpres.C
    VersionDeclarationsMacro (TBoardPresenter);

    // Build the TicTacToe game board
    virtual TView *HandleCreateMainView(TGUIBundle *) const;

    // Build the TicTacToe menu.
    virtual void  HandleCreateExtendedMenus (TMenu& theMainMenu)
          const;

    // We will remove Select All from the Edit menu.
    virtual void  HandleCreateEditMenu (TMenu& theMainMenu) const;
```

TicTacToe – ticpres.h

```
  protected:
    virtual void CallBackMenuItemNewGame() const; // has to be const

  private:
    // Version number used for streaming.
    enum            {kOriginalVersion};
};

#endif // IBMSample_TICPRES
```

TicTacToe – ticview.h

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicView.h

#ifndef IBMSample_TICVIEW
#define IBMSample_TICVIEW

// Include for Presentation Framework
#ifndef Taligent_GUICOMPOUNDDOCUMENT
  #include "GUICompoundDocument.h"
#endif

// Include for drawing services, TGrafPort
#ifndef Taligent_GRAFPORT
  #include <GrafPort.h>
#endif

#ifndef Taligent_ACTIVATIBLEVIEW
  #include <ActivatibleView.h>
#endif

#ifndef TALIGENT_MOUSEINPUT            // MMouseEventHandler
  #include <MouseInput.h>
#endif


 // TRequestBasedMemberFunctionConnectionTo
#ifndef TALIGENT_NOTIFICATION
  #include <Notification.h>
#endif

#ifndef IBMSample_TICSELEC
  #include "ticselec.h"
#endif
```

TicTacToe – ticview.h

```
class TSquareView : public TDocumentComponentView,
            public MMouseEventHandler
{
  public:

    // Our constructor
    TSquareView ( TGUIBundle* b,
                const long pos,
                const TColorPaint& bgndCol = TColorPaint::GetWhite() );

    // Our destructor
    virtual        ~TSquareView ();

    // We will do our initialization here
    virtual  void   Initialize();

    //MouseEventHandler MouseDown method overrides
    virtual Boolean MouseDown (TMouseDownEvent&);

    // A convenience function which calls MGUIBundle's
    // GetCurrentModelSelection and safely downcasts the
    // TModelSelection to a TTicSelection*.  The user should
    // always check for a 0 pointer after calling this function.
    const TTicSelection * GetTicSelection () const;

    // We don't need full mcollectible support. We only need metainfo
    // so we make our empty constructor protected and use
    // VersionDeclarationMacro instead of
    // MCollectibleDeclarationsMacro
    VersionDeclarationsMacro (TSquareView);

  protected:
    virtual void    HandleAddedToHierarchy (const TGrafPort*
            thePortToAlias);
    virtual void    HandleRemovedFromHierarchy();

    virtual void    DrawContents (TGrafPort& port) const;
```

```
    void   DrawLetter (TGrafPort& port,TGArea& area, char letter)
           const;

    TSquareView ();
    TSquareView (const TSquareView& copy);

    // Methods to handle notification

    // HandleCurrentSelectionChanged when not overridden just calls
    // HandleNotification
    virtual void HandleCurrentSelectionChanged(const TNotification& );

    // HandleCurrentModelChanged when not overridden just calls
    // HandleNotification
    virtual void HandleModelChanged(const TNotification& );

    // HandleNotification default behavior is to invalidate the view
    virtual void HandleNotification(const TNotification& );

    typedef
           TRequestBasedMemberFunctionConnectionTo<TSquareView>
           TSquareViewModelConnection;

  private:
    // Used for versioning during streaming
    enum          {kOriginalVersion};

    // Used to store background color of the view
    const         TColorPaint& fBcolor;

    // The  tictactoe board has 9 squares numbered from 0 to 8
    // This data member will hold the number of this square
    const long      fSquareIndex;

    TSquareViewModelConnection*   fConnection;
};

#endif // IBMSample_TICVIEW
```

TicTacToe – ticcomma.h

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicComma.h

#ifndef IBMSample_TICCOMMA
#define IBMSample_TICCOMMA

// Include for Presentation Framework commands
#ifndef Taligent_GUICOMPOUNDDOCUMENTCOMMAND
#include <GUICompoundDocumentCommand.h>
#endif

#ifndef IBMSample_TICMODEL
#include ”ticmodel.h”
#endif

#ifndef IBMSample_TICSELEC
#include ”ticselec.h”
#endif

// TPlayCommand will change the letter of a square referred to
// by a TTicSelection. TTicSelection is a model selection which
// can specify either the whole model or just one square.
// A TTicSelection can also be empty.

class TPlayCommand : public TCommandOn<TTicSelection>
{
  public:
    // The following macro must be called within MCollectible class
    // declarations.The macro defines a set of MCollectible utility
    // functions. This macro call matches a macro call to
    // MCollectibleDefinitionsMacro in TicComma.C
    MCollectibleDeclarationsMacro (TPlayCommand);

    // A true default (void) constructor
              TPlayCommand ();
    virtual    ~TPlayCommand ();
```

TicTacToe – ticcomma.h

```
    //  copy constructor
                TPlayCommand (const TPlayCommand &copy);
    // stream–out and stream–in operators
    virtual TStream&        operator>>= (TStream& toStream) const;
    virtual TStream&        operator<<= (TStream& fromStream);

protected:

    // These are the primary methods to concern yourself with.  For
    // incremental commands you would also need to override
    // HandleDoIncrement().

    // HandleDoBegin() will start (and, in this case, also finish)
    // the command.  Information which will be useful for undo will
    // also be saved.
    virtual void        HandleDoBegin (TTicSelection& target);

    // HandleUndo() uses the information saved from HandleDoBegin()
    // to undo the operation.
    virtual void        HandleUndo (TTicSelection& target);

    // HandleRedo() works like HandleDoBegin() except that it will
    // not need to save information for HandleUndo().
    virtual void        HandleRedo (TTicSelection& target);

    // HandleCanDo() returns true when the command can be performed,
    // in this case, when the target is not empty.  HandleCanDo is used
    // by the Presentation Framework to activate/deactive menu items
    // when the command is bound to the menu item with a
    // TCommandControlStateOn<>.
    virtual Boolean     HandleCanDo (const TTicSelection& target)
        const;
```

TicTacToe – ticcomma.h

```
  private:
    // Version number
    enum        {kOriginalVersion};

    // Place operator= in private section to prevent it from being called.
    TPlayCommand&    operator= (const TPlayCommand& copy)
{return *this;}
};

// The command binding binds an command instance to a selection
// instance.  The document adopts the command binding and invokes the
// command.Do(selection) operation.  The document then streams out the
// command/selection pair for undo/redo, saveless model, etc.
typedef
        TGUIDocumentComponentCommandBindingTo<TTicSelection>
        TTicCommandBinding;

class TNewGameCommand : public TCommandOn<TTicSelection>
{
  public:
    // The following macro must be called within MCollectible class
    // declarations.The macro defines a set of MCollectible utility
    // functions. This macro call matches a macro call to
    // MCollectibleDefinitionsMacro in TicComma.C
     MCollectibleDeclarationsMacro (TNewGameCommand);

    // A true default (void) constructor is also needed by MCollectible.
                TNewGameCommand ();
    virtual     ~TNewGameCommand ();
```

TicTacToe – ticcomma.h

```
// MCollectibles require that copy constructors be
// defined as well as assignment and streaming operators.
            TNewGameCommand (const TNewGameCommand
                     &copy);
virtual TStream&        operator>>= (TStream& toStream) const;
virtual TStream&        operator<<= (TStream& fromStream);

protected:

// These are the primary methods to concern yourself with.  For
// incremental commands you would also need to override
// HandleDoIncrement().

// HandleDoBegin() will start (and, in this case, also finish)
// the command.  Information which will be useful for undo will
// also be saved.
virtual void        HandleDoBegin (TTicSelection& target);

// HandleUndo() uses the information saved from HandleDoBegin()
// to undo the operation.
virtual void        HandleUndo (TTicSelection& target);

// HandleRedo() works like HandleDoBegin() except that it will
// not need to save information for HandleUndo().
virtual void        HandleRedo (TTicSelection& target);

// HandleCanDo() returns true when the command can be performed,
// in this case, when the target is not empty.  HandleCanDo is used
// by the Presentation Framework to activate/deactive menu items
// when the command is bound to the menu item with a
// TCommandControlStateOn<>.
virtual Boolean     HandleCanDo (const TTicSelection& target)
    const;
```

TicTacToe – ticcomma.h

```
  private:
    // Version number
    enum      {kOriginalVersion};

    // The squares to save to.
    char fOldSquares[9];
    char fNextLetter;

    // Place operator= in private section to prevent it from being called.
    TNewGameCommand&    operator= (const TNewGameCommand&
        copy) {return *this;}
};

#endif // IBMSample_TICCOMMA
```

TicTacToe – ticselec.h

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicSelect.h

#ifndef IBMSample_TICSELEC
#define IBMSample_TICSELEC

#ifndef IBMSample_TICMODEL
  #include <ticmodel.h>
#endif

// TTicSelection is the selection return by TTicModel::CreateSelection().
// The TTicSelection can be on an individual square, on the whole model
// or on none of the model.

class TTicSelection : public TGUIModelSelectionFor<TTicModel>
{
  public:

    // The following macro must be called within MCollectible class
    // declarations. The macro defines a set of MCollectible utility
    // functions. This macro call matches a macro call to
    // MCollectibleDefinitionsMacro in TicSelec.C
    MCollectibleDeclarationsMacro (TTicSelection);

    // The following macro must be called when dynamic type casting
    // is required. The macro defines a set of dynamic casting utility
    //  functions.  This macro call matches a macro call to
    // DynamicCastDefinitionsMacro in TicSelec.C
    DynamicCastDeclarationsMacro ();

    // The constructor takes a TModelReference& which it uses to
    // initialize its TGUIModelSelectionFor<TTicModel> base class.
                    TTicSelection (const TModelReference& theModel);

    virtual         ~TTicSelection ();
```

TicTacToe – ticselec.h

```
// copy constructor
                TTicSelection (const TTicSelection& copy);
// stream–out and stream–in operators
virtual TStream&   operator>>= ( TStream& towhere ) const;
virtual TStream&   operator<<= ( TStream& fromwhere );

// TGUIModelSelectionFor<TTicModel> overrides.   Support for
// selection of the whole model, none of the model or an
// undefined selection.
virtual void       DeselectAll ();
virtual void       SelectWholeModel ();
virtual void       SetUndefined ();

// Support for selection of one Square.
void               SelectSquare (const long whichSquare);

// The command will use these methods to access the model
void               PlayNextLetter ();
void               UndoPlayLetter ();

// Return true if the selection contains a specific SquareIndex.
// This is used by the view to provide current selection feedback
// by drawing a frame around each view in the selection.
Boolean            IsSquareSelected (const long whichSquare) const;
```

TicTacToe – ticselec.h

```
  protected:
    //  default (void) constructor
              TTicSelection ();
  private:
    // Used for versioning during streaming.
    enum                    {kOriginalVersion};

    // holds the square that has been selected
    long        fSelectedIndex;

    enum                    {kSelectedAll = 100};

    // Place operator= in private section to prevent it from being called.
    TTicSelection&    operator= (const TTicSelection& copy)
            {return *this;}
};

#endif // IBMSample_TICSELECT
```

TicTacToe – ticmodel.h

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicModel.h

#ifndef IBMSample_TICMODEL
#define IBMSample_TICMODEL

// Include file for the Presentation Framework
#ifndef Taligent_GUICOMPOUNDDOCUMENT
#include <GUICompoundDocument.h>
#endif

// The TTicModel will contain an array of 9 chars representing
// each play of the game .  TTicModel is derived from TModel and
// contains a data representation which should be independent of
// the visual presentation.
class TTicModel : public TModel
{
  public:
    // All TModel's should include the following delcaration macro.
    // This macro declares MCollectible and DynamicCast functions.
    // This macro call matches the ModelDefinitionsMacro in
    // TicModel.C
    ModelDeclarationsMacro(TTicModel);

    // TModel is MCollectible so default and copy constructors must be
    // written as well as assignment and streaming operators.
           TTicModel ();
           TTicModel (const TTicModel &copy);
    virtual  ~TTicModel ();
```

TicTacToe – ticmodel.h

```
// Streaming operators stream out/in a full representation of the
// model. These methods are called automatically by the Framework
// to store and retrieve models.  >>= operators should  provide
// version stamping
// of the data so that <<= operators can convert from older versions.
virtual TStream&   operator>>= (TStream& towhere) const;
virtual TStream&   operator<<= (TStream& fromwhere);


// Create a default selection.
virtual TModelSelection* CreateSelection () const;

// Get a square data for reading, used by view for rendering.
// Made const so clients call call this function when the model
// is read–only locked.
char     GetSquareLetter (const long whichSquare) const;

// Write a play into the model.  A client cannot call non–const
// methods when the model is read–only locked.
void     SetSquareLetter (const long  whichSquare, char letter);

// Methods to control the next play
char     GetNextPlay ();
void     SetNextPlay (char letter);  // for Undo use

// Resets the model to start a new game
void     NewGame();

// Returns the interest used for drastic changes in the model
// a drastic change in this program is a modification in the
// model that requires all views to be updated.
// All views will register interest  in a drastic change
virtual  TInterest       GetDrasticChangeInterest() const;
```

TicTacToe – ticmodel.h

```
    // Sends a drastic change notification
    virtual  void           NotifyOfDrasticChange();

  private:
    // Used for versioning during streaming
    enum          {kOriginalVersion};

    // The 9 possible plays
    char fSquares[9];

    // Toggle variable that will be set to "X" or "O"
    char fNextLetter;

    // Place operator= in private section to prevent it from being called.
    TTicModel&   operator= (const TTicModel& model) { return *this;}
};

#endif // IBMSample_TICMODEL
```

TicTacToe – ticpres.C

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicPres.C

// Include for Presentation Framework commands
#ifndef Taligent_GUICOMPOUNDDOCUMENTCOMMAND
  #include <GUICompoundDocumentCommand.h>
#endif

// Include for TMomentaryMenuItem, TMenu
#ifndef Taligent_MENUCONTROLS
  #include <MenuControls.h>
#endif

#ifndef Taligent_LAYOUTVIEWS   // TBoxView , THBoxView
  #include <LayoutViews.h>
#endif

#ifndef IBMSample_TICPRES
  #include "ticpres.h"
#endif

#ifndef IBMSample_TICMODEL
  #include "ticmodel.h"
#endif

#ifndef IBMSample_TICCOMMA
  #include "ticcomma.h"
#endif

#ifndef IBMSample_TICSELEC
  #include "ticselec.h"
#endif

// Use static to force instantiation of a class template from the
// template class. Future C++ syntax will support a template keyword
// for doing this.
static TGUIModelPresenterStationeryFor<TTicModel,
              TBoardPresenter> theStationery;
```

TicTacToe – ticpres.C

```
// This macro call matches a macro call to VersionDeclarationsMacro
// in the TBoardPresenter class declaration.  This macro defines the
// versioning utility functions.  kOriginalVersion is the version number.
VersionDefinitionsMacro(TBoardPresenter, kOriginalVersion);

// The constructor for TBoardPresenter must take a TGUIBundle which it

// passes on to the constructor for TGUIPresenterFor<TTicView>.
TBoardPresenter::TBoardPresenter(const TGUIBundle& bundle)
      : TGUIPresenter(bundle)
{
  qprintf(”TBoardPresenter::TBoardPresenter(const TGUIBundle&
           bundle)\n”);
}

TBoardPresenter::~TBoardPresenter()
{
  qprintf(”TBoardPresenter::~TBoardPresenter()\n”);
}

TView *
TBoardPresenter::HandleCreateMainView(TGUIBundle *bundle) const
{
  qprintf(”TBoardPresenter::HandleCreateMainView(TGUIBundle
           *bundle)\n”);
  TBoxView * fBoxView = new TBoxView(  3, 3,
                             TBoxView::kCentered, // h align
                             TBoxView::kCentered, // v align
                             TBoxView::kExpand,   // h expand
                             TBoxView::kExpand,   // v expand
                             TGPoint(0,0) );      // gap
```

TicTacToe – ticpres.C

```
  fBoxView–>AdoptChildren (
            new TSquareView(bundle,0,TColorPaint::GetLightGray53  ()),
            new TSquareView(bundle,1,TColorPaint::GetWhite ()),
            new TSquareView(bundle,2,TColorPaint::GetLightGray53  ()),
            new TSquareView(bundle,3,TColorPaint::GetWhite ()) );
  fBoxView–>AdoptChildren (
            new TSquareView(bundle,4,TColorPaint::GetLightGray53  ()),
            new TSquareView(bundle,5,TColorPaint::GetWhite ()),
            new TSquareView(bundle,6,TColorPaint::GetLightGray53  ()),
            new TSquareView(bundle,7,TColorPaint::GetWhite ()) );
  fBoxView–>AdoptChildren (
            new TSquareView(bundle,8,TColorPaint::GetLightGray53  ()),
            NIL, NIL, NIL );

  return  fBoxView;
}

void
TBoardPresenter::CallBackMenuItemNewGame() const
{
  qprintf(”TBoardPresenter::CallBackMenuItemNewGame() const\n”);

  TModelPointerTo<TTicModel> theModel(GetModelReference());

  // Create a selection and set it to the square chosen
  // or to an empty selection.
  TTicSelection* theSelection;
  DynamicCastTo(theSelection, theModel–>CreateSelection ());
  theSelection–>SelectWholeModel();

  TNewGameCommand* theCommand = new TNewGameCommand ();
```

TicTacToe – ticpres.C

```
  // Create a command binding to bind command and selection
  // Pass the GUIBundle and a text string for the Undo/Redo menu item.
  TTicCommandBinding* theCommandBinding =
          new TTicCommandBinding ( theCommand,
                        theSelection,
                        *GetGUIBundle(),
                        TStandardText("NewGame"));

  // Access the document and have it adopt and do the command binding
  ::NonConst(this)->AdoptAndDo (theCommandBinding);

}

// Create the TicTacToe submenu and add to theMainMenu
void
TBoardPresenter::HandleCreateExtendedMenus(TMenu&
          theMainMenu) const
{
  qprintf("TBoardPresenter::HandleCreateExtendedMenus(TMenu&
          theMainMenu)\n");

  TMomentaryMenuItem* item;
  TMenu* ticMenu;

  // This menu will become the TicTacToe menu
  ticMenu = new TMenu;
  ticMenu->SetControlLayout(MControl::kTopToBottom);
  ticMenu->SetItemLayout( MControl::kLeftToRight );

  // First create a menu item labeled NewGame
  item = new TMomentaryMenuItem();
  item->AdoptLabel(new TTextLabel(TStandardText("NewGame")));

  item->AdoptState(
          new TMomentaryPresenterCallbackTo<TBoardPresenter> (
               this,CallBackMenuItemNewGame ));
```

TicTacToe – ticpres.C

```
  // Attach to the ticMenu
  ticMenu–>AdoptLast(item);

  // Create a submenu from the ticMenu and label it "TicTacToe"
  // Attach the submenu to the main menu
  theMainMenu.AdoptLast (new TSubMenuItem (ticMenu,
              new TTextLabel(TStandardText("TicTacToe"))));
}

// We will remove the Select All item from the Edit submenu
void
TBoardPresenter::HandleCreateEditMenu(TMenu& theMainMenu)
          const
{
  qprintf("TBoardPresenter::HandleCreateEditMenu(TMenu&
          theMainMenu)\n");

  // Let the framework add the Edit submenu to the main menu
  TGUIPresenter::HandleCreateEditMenu(theMainMenu);


  // At this moment, the Edit item is the last item in the
  // main menu. Lets get it.
  MMenuItem * editItem     = theMainMenu.GetLast();

  // Since actually it is a submenu
  TMenu * editMenu = ((TSubMenuItem *)editItem)–>GetSubMenu();

  // We know that SelectAll is the last item in the Edit submenu
  // so we get it.
  MMenuItem * selectallItem = editMenu–>GetLast();

  // We then orphan it. It will no longer be shown
  editMenu–>Orphan(*selectallItem);

  // Now we own the item and are responsible for getting rid of it

 }
```

TicTacToe – ticview.C

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// ticview.C

#ifndef IBMSample_TICVIEW
  #include "ticview.h"
#endif

#ifndef IBMSample_TICMODEL
  #include "ticmodel.h"
#endif

#ifndef IBMSample_TICCOMMA
  #include "ticcomma.h"
#endif

#ifndef Taligent_GLYPHRUN
  #include <GlyphRun.h>
#endif
#ifndef Taligent_FONT
  #include <Font.h>
#endif
#ifndef Taligent_LOCALIZATION
  #include <Localization.h>
#endif
#ifndef Taligent_GLYPHRUNGEOMETRY
  #include <GlyphRunGeometry.h>
#endif
```

TicTacToe – ticview.C

```
//*****************************************************
VersionDefinitionsMacro(TSquareView, kOriginalVersion);

TSquareView::TSquareView(TGUIBundle* bundle,
                    const long pos,
                    const TColorPaint& bgndCol)
        : TDocumentComponentView (bundle),
          fSquareIndex (pos),
          fBcolor (bgndCol),
          VViewInitialize (&gMetaInfo)  // required for views
{
  qprintf(”TSquareView::TSquareView(TGUIBundle* bundle) –
          %ld\n”,fSquareIndex);

  // required for views
  CheckForInitialize(&gMetaInfo);
}

// Default constructor
TSquareView::TSquareView()
   : TDocumentComponentView (),
     fSquareIndex (0),
     fBcolor (TColorPaint::GetWhite()),
     VViewInitialize(&gMetaInfo)      // required for views
{
  qprintf(”TSquareView::TSquareView()  – %ld\n”,fSquareIndex);

  // required for views
  CheckForInitialize (&gMetaInfo);
}
```

TicTacToe – ticview.C

```
// Copy constructor
TSquareView::TSquareView(const TSquareView& copy)
    : TDocumentComponentView (copy),
      fSquareIndex (copy.fSquareIndex),
      fBcolor (TColorPaint::GetWhite()),
      VViewInitialize(&gMetaInfo)       // required for views
{
   qprintf(”TSquareView::TSquareView(const TSquareView& copy)  –
            %ld\n”,fSquareIndex);

   // required for views
   CheckForInitialize(&gMetaInfo);

// All views must call CheckForFinalize in their destructors.
TSquareView::~TSquareView()
{
   qprintf(”TSquareView::~TSquareView()  – %ld\n”,fSquareIndex);

   // required for views
   CheckForFinalize(&gMetaInfo);
}

void
TSquareView::Initialize()
{
   qprintf(”TSquareView::Initialize() – %ld\n”,fSquareIndex);

   TContentView::Initialize();
   SetAllocatedArea (TGArea(TGRect(TGPoint(0,0), TGPoint(50, 50))));
}
```

TicTacToe – ticview.C

```
Boolean
TSquareView::MouseDown(TMouseDownEvent& mouseEvent)
{
  qprintf("TSquareView::MouseDown() – %ld\n",fSquareIndex);

  // Create a smart pointer which will lock the model for writing,
  // and will release the lock on destruction of the pointer.
  TModelPointerTo<TTicModel> model(GetModelReference());

  // Create a selection and set it to the square chosen or
  // to an empty selection.
  TTicSelection* selection =
          (TTicSelection*) model–>CreateSelection ();
  selection–>SelectSquare (fSquareIndex);

  TPlayCommand* command = new TPlayCommand ();

  // Create a command binding to bind command and selection
  // Pass the GUIBundle and a text string for the Undo/Redo menu item.
  TTicCommandBinding* commandBinding =
        new TTicCommandBinding (command,
                          selection,
                          *GetGUIBundle(),
                          TStandardText("Play"));

  // Access the document and have it adopt and do the command binding
  AdoptAndDo (commandBinding);

  // mouse event has been handle so return true
  return TRUE;
}
```

TicTacToe – ticview.C

```
void
TSquareView::HandleAddedToHierarchy( const TGrafPort*
thePortToAlias )
{
  qprintf("TSquareView::HandleAddedToHierarchy( const TGrafPort*
          thePortToAlias ) – %ld\n",fSquareIndex);

  TDocumentComponentView::HandleAddedToHierarchy(
          thePortToAlias );

  // create the object that will dispatch notifications
  fConnection= new TSquareViewModelConnection(
                  this,                   // the receiver
                  GetRequestProcessor(),  // the request queue
                  TDocumentComponentView::HandleModelChanged);
// callback function
  {
    const TModelPointerTo<TTicModel> model(GetModelReference());

    // register the interest in the notification
    fConnection–>AddInterest(model–>GetDrasticChangeInterest());
  }

  // connect for notification
  fConnection–>Connect();
//  SetCoordinateView(TViewHandle(*this));
}
```

TicTacToe – ticview.C

```
void
TSquareView::HandleRemovedFromHierarchy()
{
  qprintf(”TSquareView::HandleRemovedFromHierarchy() –
           %ld\n”,fSquareIndex);

  TDocumentComponentView::HandleRemovedFromHierarchy();
  fConnection–>Disconnect();
  fConnection–>RemoveAllInterests();
  delete fConnection;
}

void
TSquareView::DrawContents(TGrafPort& port) const
{
  qprintf(”TSquareView::DrawContents(TGrafPort& port) const –
           %ld\n”,fSquareIndex);

  // Fill the background
  const TGrafBundle bundle( new TColorPaint( fBcolor ),
                    TGrafBundle::kFill) ;

  TGArea area;
  GetBounds(area);
  port.Draw (area, bundle);

  // Create a smart pointer which will lock the model for reading,
  // and will release the lock on destruction of the pointer.
  const TModelPointerTo<TTicModel> model(GetModelReference());

  // Read the letter from the model and draw it
  DrawLetter (port,area, model–>GetSquareLetter(fSquareIndex));

}
```

TicTacToe – ticview.C

```
//Draw an individual Letter into the port.
void
TSquareView::DrawLetter (TGrafPort& port,TGArea& area, char letter)
const
{
  qprintf(”TSquareView::DrawLetter () const – %ld\n”,fSquareIndex);

  //Create a locale and initialize to our current locale
  TLocale aLocale;
  TLocale::GetCurrentLocale(aLocale);

  //Create orange paint
  TColorPaint *anOrangeColorPaint = new TColorPaint(
                                   TRGBColor( 1.0,0.5,0.0 ) );

  //Create the font
  TFont *theGlyphFont = new
TFont(aLocale,TStandardText(”Chicago”),
                  TStandardText(”Bold”),GCoordinate(10.0));

  // allocate area for our string of one character
  char str[2];
  str[0] = letter; str[1] = 0;

  //Create a Glyph Run Geometry
  TGGlyphRun theGGlyphedText( str, theGlyphFont, TGPoint(0.0,0.0)
);

  //Build the graf bundle for our text, using a fill of orange paint
  TGrafBundle *txtGrafBundle =
        new TGrafBundle(anOrangeColorPaint,TAttributeState::kFill);

  //Create the Glyph Run
  TGlyphRun theGlyphRun(theGGlyphedText,txtGrafBundle);
```

TicTacToe – ticview.C

```
  theGlyphRun.SetOrigin(
    TGPoint( (area.GetBounds().fRight –
            theGlyphRun.GetGeometricBounds().fRight –
            theGlyphRun.GetGeometricBounds().fLeft)/2.0 ,
           (area.GetBounds().fBottom –
            theGlyphRun.GetGeometricBounds().fBottom –
            theGlyphRun.GetGeometricBounds().fTop)/2.0 ) );

  // Because Glyph is an MGraphic we can transform it
  // We will scale it to match the size of our view area
  theGlyphRun.ScaleBy(

TGPoint(area.GetBounds().fRight/10.0,area.GetBounds().fBottom/10.0),

TGPoint(area.GetBounds().fRight/2.0,area.GetBounds().fBottom/2.0));

  //Tell the Glyph Run to draw
  theGlyphRun.Draw(port);

}

// A convenience function which calls MGUIBundle's
// GetCurrentModelSelection and safely downcasts the TModelSelection
// to a TTicSelection*.  The user should always check for a 0 pointer
// after calling this function.
const TTicSelection *
TSquareView::GetTicSelection () const
{
  qprintf("TSquareView::GetTicSelection () const –
           %ld\n",fSquareIndex);
```

TicTacToe – ticview.C

```
  // Get a pointer to the current selection, if any.
  const TModelSelection* modelSelection =
          GetCurrentModelSelection();
  if (modelSelection)
  {
    // Attempt to safely downcast the TModelSelection to a
    // TTicSelection. IsA will return 0 if the downcast is not safe.
    return (const TTicSelection*)
          modelSelection–>IsA(TTicSelection::kType);
  }
  else
  {
    return 0;
  }
}

// The notification handled by this method is sent by the framework
// when a selection changes
void
TSquareView::HandleCurrentSelectionChanged(const TNotification&
          theNotification)
{
  qprintf(”TSquareView::HandleCurrentSelectionChanged(const
          TNotification& ) – %ld\n”,fSquareIndex);

  qprintf(”.......Token = ”);
  theNotification.GetName( ).PrintName();
  qprintf(”\n”);
  // Get a pointer to the current selection.  May receive a 0 pointer if
  // there is no current selection.
  const TTicSelection* currentSelection = GetTicSelection();
```

TicTacToe – ticview.C

```
  // If the new selection is the square that is represented by this
  // view then let the notification be passed to HandleNotification
  // to invalidate the view
  if (currentSelection &&
          currentSelection–>IsSquareSelected(fSquareIndex))
    TDocumentComponentView::HandleCurrentSelectionChanged(
          theNotification);

}

void
TSquareView::HandleModelChanged(const TNotification&
theNotification)
{
  qprintf(”TSquareView::HandleModelChanged(const TNotification& )
          – %ld\n”,fSquareIndex);

  TDocumentComponentView::HandleModelChanged(theNotification);
}

// HandleNotification default behavior is to invalidate the view
void
TSquareView::HandleNotification(const TNotification& theNotification)
{
  qprintf(”TSquareView::HandleNotification(const TNotification& ) –
          %ld\n”,fSquareIndex);

  qprintf(”.......Token = ”);
  theNotification.GetName( ).PrintName();
  qprintf(”\n”);

  TDocumentComponentView::HandleNotification(theNotification);
}
```

TicTacToe – ticcomma.C

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicComma.C

#ifndef IBMSample_TICCOMMA
  #include "ticcomma.h"
#endif

// This macro call matches a macro call to
// MCollectibleDeclarationsMacro in the TPlayCommand class
// declaration.  This macro defines the MCollectible utility functions.
// kOriginal version is the version level of the command.
MCollectibleDefinitionsMacro(TPlayCommand, kOriginalVersion) ;

// This is the constructor used in this program
TPlayCommand::TPlayCommand()
// kSerialUndo must be specified in the base class constructor to make
// this command undoable.  The default is kCantUndo.
:
TCommandOn<TTicSelection>(TCommandOn<TTicSelection>::kSerial
            Undo)
{
   qprintf("TPlayCommand::TPlayCommand()\n");
}

// Copy constructor, required by MCollectible
TPlayCommand::TPlayCommand(const TPlayCommand &copy) :
      TCommandOn<TTicSelection>(copy)
{
   qprintf("TPlayCommand::TPlayCommand(const TPlayCommand
            &copy)\n");
}

TPlayCommand::~TPlayCommand()
{
   qprintf("TPlayCommand::~TPlayCommand() \n");
}
```

TicTacToe – ticcomma.C

```
// Out streaming operator
TStream&
TPlayCommand::operator>>=(TStream& toStream) const
{
  qprintf("TPlayCommand::operator>>=(TStream& toStream) const\n");
  // Place the version number onto the stream
  WriteVersion(toStream);

  // Stream out the base class data
  TCommandOn<TTicSelection>::operator>>=(toStream);

  // Stream out TPlayCommand data

  return toStream;
}

// In streaming operator
TStream&
TPlayCommand::operator<<=(TStream& fromStream)
{
  qprintf("TPlayCommand::operator<<=(TStream& fromStream)\n");
  // Read in the version
  VersionInfo version = ReadVersion(fromStream);

  switch (version)
  {

    case kOriginalVersion:
    {
      short temp;
      // Stream in base class portion
      TCommandOn<TTicSelection>::operator<<=(fromStream);

      // Stream in the TPlayCommand data

      break;
    }
```

TicTacToe – ticcomma.C

```
    // New versions would be added here to support changes in the
    // TTicSelection

    default:
      throw TInvalidVersionError();
  }

  return fromStream;
}

// Execute the command on the selection.
// Save information needed to undo (i.e., the old letter).
// Commands assume that the caller has taken care of locking the model
void
TPlayCommand::HandleDoBegin(TTicSelection& target)
{
  qprintf("TPlayCommand::HandleDoBegin(TTicSelection& target)\n");

  // Change the letter in the selection
  target.PlayNextLetter ();
}

// Use the information saved in HandleDoBegin to undo the
// effects of the command.
// Commands assume that the caller has taken care of locking the model
void
TPlayCommand::HandleUndo(TTicSelection& target)
{
  qprintf("TPlayCommand::HandleUndo(TTicSelection& target)\n");

  target.UndoPlayLetter ();
}
```

TicTacToe – ticcomma.C

```
// Similar to HandleDoBegin except there is no need to save
// information for undo purposes.
// Commands assume that the caller has taken care of locking the model
void
TPlayCommand::HandleRedo(TTicSelection& target)
{
  qprintf("TPlayCommand::HandleRedo(TTicSelection& target) \n");

  // Change the letter  in the selection
  target.PlayNextLetter ();
}

// HandleCanDo() returns true when the command can be performed, in
// this case, when the target is not empty.  HandleCanDo is used by
// the Presentation Framework to activate/deactive menu items when
// the command is bound to the menu item with a
// TCommandControlStateOn<>.
Boolean
TPlayCommand::HandleCanDo(const TTicSelection& target) const
{
  qprintf("TPlayCommand::HandleCanDo(const TTicSelection& target)
          const\n");

  return !target.IsEmpty();
}

MCollectibleDefinitionsMacro(TNewGameCommand,
          kOriginalVersion) ;
```

TicTacToe – ticcomma.C

```
// This is the constructor used in this program
// Default constructor
TNewGameCommand::TNewGameCommand()
// kSerialUndo must be specified in the base class constructor to make
// this command undoable.  The default is kCantUndo.
: TCommandOn<TTicSelection>(TCommandOn<TTicSelection>::
            kSerialUndo),
            fNextLetter(0)
{
  qprintf("TNewGameCommand::TNewGameCommand()\n");

  for (int i = 0;i < 9;i++)
    fOldSquares[i] = 0;
}

// Copy constructor
TNewGameCommand::TNewGameCommand(const
TNewGameCommand &copy) :
      TCommandOn<TTicSelection>(copy),
      fNextLetter(copy.fNextLetter)
{
  qprintf("TNewGameCommand::TNewGameCommand(const
TNewGameCommand &copy)\n");

  for (int i = 0;i < 9;i++)
    fOldSquares[i] = copy.fOldSquares[i];
}

TNewGameCommand::~TNewGameCommand()
{
  qprintf("TNewGameCommand::~TNewGameCommand() \n");
}
```

TicTacToe – ticcomma.C

```
// Out streaming operator
TStream&
TNewGameCommand::operator>>=(TStream& toStream) const
{
  qprintf("TNewGameCommand::operator>>=(TStream& toStream)
           const\n");
  // Place the version number onto the stream
  WriteVersion(toStream);

  // Stream out the base class data
  TCommandOn<TTicSelection>::operator>>=(toStream);

  // Stream out TNewGameCommand data
     for (int i = 0;i < 9;i++)
       fOldSquares[i] >>= toStream;
     fNextLetter >>= toStream;

  return toStream;
}
```

TicTacToe – ticcomma.C

```
// In streaming operator
TStream&
TNewGameCommand::operator<<=(TStream& fromStream)
{
  qprintf("TNewGameCommand::operator<<=(TStream&
          fromStream)\n");
  // Read in the version
  VersionInfo version = ReadVersion(fromStream);

  switch (version)
  {

    case kOriginalVersion:
    {
      short temp;
      // Stream in base class portion
      TCommandOn<TTicSelection>::operator<<=(fromStream);

      // Stream in the TNewGameCommand data
      for (int i = 0;i < 9;i++)
        fOldSquares[i] <<= fromStream;
      fNextLetter <<= fromStream;

      break;
    }

    // New versions would be added here to support changes in the
    // TTicSelection

    default:
      throw TInvalidVersionError();
  }

  return fromStream;
}
```

TicTacToe – ticcomma.C

```
// Execute the command on the selection.  Save information needed to
// undo (i.e., the old letter).
// Commands assume that the caller has taken care of locking the model
void
TNewGameCommand::HandleDoBegin(TTicSelection& target)
{
   qprintf("TNewGameCommand::HandleDoBegin(TTicSelection&
            target)\n");

   // Save the old game
   for (int i = 0;i < 9;i++) {
      fOldSquares[i] =
            target.GetModelForReading()->GetSquareLetter(i);
   }

   target.GetModelForWriting()->GetNextPlay();
   fNextLetter = target.GetModelForWriting()->GetNextPlay();
   target.GetModelForWriting()->NewGame();
}

// Use the information saved in HandleDoBegin to undo the effects of the
// command.  Commands assume that the caller has taken care of locking
// the model
void
TNewGameCommand::HandleUndo(TTicSelection& target)
{
  qprintf("TNewGameCommand::HandleUndo(TTicSelection&
            target)\n");

   for (int i = 0;i < 9;i++) {
      target.GetModelForWriting()->SetSquareLetter(i,fOldSquares[i]);
   }
   target.GetModelForWriting()->SetNextPlay(fNextLetter);
   target.GetModelForWriting()->NotifyOfDrasticChange();
}
```

TicTacToe – ticcomma.C

```
// Similar to HandleDoBegin except there is no need to save information
// for undo purposes. Commands assume that the caller has taken care of
// locking the model
void
TNewGameCommand::HandleRedo(TTicSelection& target)
{
  qprintf(”TNewGameCommand::HandleRedo(TTicSelection& target)
            \n”);
  HandleDoBegin(target);
}

// HandleCanDo() returns true when the command can be performed, in
// this case, when the target is not empty.  HandleCanDo is used by
// the Presentation Framework to activate/deactive menu items when
// the command is bound to the menu item with a
// TCommandControlStateOn<>.
Boolean
TNewGameCommand::HandleCanDo(const TTicSelection& target)
            const
{
  qprintf(”TNewGameCommand::HandleCanDo(const TTicSelection&
            target) const\n”);


//   return !target.IsEmpty() && !target.IsWholeModelSelected();
  return !target.IsEmpty();
}
```

TicTacToe – ticselec.C

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicSelect.C

#ifndef IBMSample_TICSELEC
  #include "ticselec.h"
#endif

// This macro call matches a macro call to
// MCollectibleDeclarationsMacro in the TTicSelection class declaration.
// This macro defines the MCollectible utility functions.  kOriginal
// version is the version level of the selection.
MCollectibleDefinitionsMacro(TTicSelection, kOriginalVersion);

// This macro call matches a macro call to
// DynamicCastDeclarationsMacro in the TTicSelection class
// declaration.  This macro defines the dynamic cast utility functions.
// Use DynamicCastDefinitionsMacroTwo if there are two base classes,
// etc.
DynamicCastDefinitionsMacroOne(TTicSelection,
            TGUIModelSelectionFor<TTicModel>);

// Constructor used to create a TTicSelection in this program
TTicSelection::TTicSelection(const TModelReference& theModel) :
      TGUIModelSelectionFor<TTicModel>(theModel)
{
  qprintf("TTicSelection::TTicSelection(const TModelReference&
            theModel)\n");

  // This selection always starts out undefined
  SetUndefined();
}
```

TicTacToe – ticselec.C

```
// default constructor, required by MCollectible
TTicSelection::TTicSelection() :
      TGUIModelSelectionFor<TTicModel>()
{
  qprintf(”TTicSelection::TTicSelection()\n”);
  // This selection always starts out undefined

  SetUndefined();
}


// copy constructor
TTicSelection::TTicSelection(const TTicSelection& copy) :
      TGUIModelSelectionFor<TTicModel>(copy),
      fSelectedIndex(copy.fSelectedIndex)
{qprintf(”TTicSelection::TTicSelection(const TTicSelection&
            copy)\n”);}

TTicSelection::~TTicSelection()
{qprintf(”TTicSelection::~TTicSelection()\n”);}

// Stream out operatorTStream&
// Must match operator<<=.  Should stream out a version level.
TStream&
TTicSelection::operator>>=( TStream& toStream ) const
{
  qprintf(”TTicSelection::operator>>=( TStream& toStream ) const\n”);
  // Place the version number onto the stream
  WriteVersion(toStream);

  // Stream out the base class data
  TGUIModelSelectionFor<TTicModel>::operator>>=(toStream);

  // Stream out TTicSelection data
  fSelectedIndex >>= toStream;

  return toStream;
}
```

TicTacToe – ticselec.C

```
// Stream in operator.
// Must match operator>>=.  Should be able to handle different versions.
TStream&
TTicSelection::operator<<=( TStream& fromStream )
{
  qprintf("TTicSelection::operator<<=( TStream& fromStream )\n");
  // Read in the version
  VersionInfo version = ReadVersion(fromStream);

  switch (version)
  {

    case kOriginalVersion:
    {
      TGUIModelSelectionFor<TTicModel>::operator<<=(fromStream);
      fSelectedIndex <<= fromStream;
      break;
    }

    // New versions would be added here to support changes in the
    // TTicSelection

    default:
      throw TInvalidVersionError();
  }

  return fromStream;
}
```

TicTacToe – ticselec.C

```
// This function is called by the Framework when the user chooses
// "Select All" and is also used whenever a whole model selection
// is needed. The base class function SelectWholeModel() must be
// called.
void
TTicSelection::SelectWholeModel()
{
  qprintf("TTicSelection::SelectWholeModel()\n");

  TGUIModelSelectionFor<TTicModel>::SelectWholeModel();

  fSelectedIndex = kSelectedAll;
}

// This function is called when an empty selection is needed
// The base class function DeselectAll() must be called.
void
TTicSelection::DeselectAll()
{
  qprintf("TTicSelection::DeselectAll()\n");
  TGUIModelSelectionFor<TTicModel>::DeselectAll();
  // Now IsEmpty() will return true

  fSelectedIndex = 0;
}

// This function is called whenever the selection is undefined and
// cannot be operated on by a command.
// The base class function SetUndefined() must be called.
void
TTicSelection::SetUndefined()
{
  qprintf("TTicSelection::SetUndefined()\n");
```

TicTacToe – ticselec.C

```
  // The default for SetUndefined is DeselectAll
  TGUIModelSelectionFor<TTicModel>::SetUndefined();
  // Now IsDefined() will return false

  fSelectedIndex = 0;
}

void
TTicSelection::SelectSquare(const long whichSquare)
{
  qprintf("TTicSelection::SelectSquare(const long whichSquare)\n");

  // Check for illegal selection
  if (whichSquare >= 0 && whichSquare <= 8)
  {
    // if the square is not empty cheating is not allowed
    if (!GetModelForReading()–>GetSquareLetter(whichSquare))
    {
      fSelectedIndex = whichSquare + 1;
      qprintf(".....Square selected index = %ld\n",fSelectedIndex);
      SetState (kNotEmpty);
      // Now IsEmpty() will return false and IsDefined() will return true
    }
    else
    {
      SetUndefined() ;
    }
  }
  else
  {
    // illegal selection
    DeselectAll();
  }
}
```

TicTacToe – ticselec.C

```
Boolean
TTicSelection::IsSquareSelected (const long whichSquare) const
{
  qprintf(”TTicSelection::IsSquareSelected (const long whichSquare)
          const\n”);

  return (!IsEmpty() && whichSquare == fSelectedIndex – 1);
}

void
TTicSelection::PlayNextLetter ()
{
  qprintf(”TTicSelection::PlayNextLetter ()\n”);
  if (!IsEmpty() && IsDefined())
  {
    GetModelForWriting()–>SetSquareLetter(fSelectedIndex–1,
                    GetModelForWriting()–>GetNextPlay());
  }
  else
  {
    // Throw an exception?
    qprintf (”TTicSelection::PlayNextLetter () illegal selection\n”);
  }

}
```

TicTacToe – ticselec.C

```
void
TTicSelection::UndoPlayLetter ()
{
  qprintf("TTicSelection::UndoPlayLetter ()\n");
  if (!IsEmpty() && IsDefined())
  {
GetModelForWriting()->SetNextPlay(GetModelForWriting()->GetNext
Play());
    GetModelForWriting()->SetSquareLetter(fSelectedIndex-1, 0);
  }
  else
  {
    // Throw an exception?
    qprintf ("TTicSelection::UndoPlayLetter () illegal selection\n");
  }
}
```

TicTacToe – ticmodel.C

```
// Copyright (C) 1994 IBM Corp. All rights reserved.

// TicModel.C

#ifndef IBMSample_TICMODEL
  #include "ticmodel.h"
#endif

#ifndef IBMSample_TICSELEC
  #include "ticselec.h"
#endif

// This macro call matches the ModelDeclarationsMacro in the
// TTicModel class declaration.  This macro defines the MCollectible and
// DynamicCast utility functions.  Use ModelDefinitionsMacroTwo if
// there are two  base classes, etc. kOriginalVersion is used to "version
// stamp" this edition of TTicModel and should be incremented for newer
// versions if the data stream changes.
ModelDefinitionsMacroOne(TTicModel, kOriginalVersion, TModel);

// Default constructor
TTicModel::TTicModel()
    : TModel()
{
   qprintf("TTicModel::TTicModel()\n");

   NewGame();

}

// Copy constructor
TTicModel::TTicModel(const TTicModel& copy)
        : TModel(copy) , fNextLetter(copy.fNextLetter)

{
   qprintf("TTicModel::TTicModel(const TTicModel& copy) \n");

   for (int i = 0;i < 9;i++)
      fSquares[i] = copy.fSquares[i];
}
```

TicTacToe – ticmodel.C

```
// Destructor
TTicModel::~TTicModel()
{
  qprintf(”TTicModel::~TTicModel()\n”);
}

// Stream out operator
TStream&
TTicModel::operator>>=(TStream& toStream) const
{
  qprintf(”TTicModel::operator>>=(TStream& toStream) const \n”);

  // Place the version number onto the stream
  WriteVersion(toStream);

  // Stream out base class portion
  TModel::operator>>=(toStream);

  // Now stream out the array.
  for (int i = 0;i < 9;i++)
    fSquares[i] >>= toStream;
  fNextLetter >>= toStream;

  return toStream;
}
```

TicTacToe – ticmodel.C

```
// Stream in operator
TStream&
TTicModel::operator<<=(TStream& fromStream)
{
  qprintf("TTicModel::operator<<=(TStream& fromStream) \n");
  // Read in the version
  VersionInfo version = ReadVersion(fromStream);

  switch (version)
  {
    case kOriginalVersion:
    {
      // Stream in the base class portion
      TModel::operator<<=(fromStream);

      // Stream in the array
      for (int i = 0;i < 9;i++)
        fSquares[i] <<= fromStream;
      fNextLetter <<= fromStream;

      break;
    }

    default:
      throw TInvalidVersionError();
  }

  return fromStream;
}
```

TicTacToe – ticmodel.C

```
// Return a new, default selection on the model.
TModelSelection*
TTicModel::CreateSelection() const
{
  qprintf("TTicModel::CreateSelection() const\n");

  TTicSelection* selection =
            new TTicSelection(TModelReference(*this));
  selection->SelectDefault();
  return selection;
}

// Get a letter from the model.  Model should be locked for reading
// when this function is called
char
TTicModel::GetSquareLetter (const long whichSquare) const
{
  qprintf("TTicModel::GetSquareLetter (long whichSquare) const\n");

  return fSquares[whichSquare];
}

// Write a letter to the model.  Model should be locked
// for reading or writing when this function is called
void
TTicModel::SetSquareLetter (const long  whichSquare, char letter)
{
  qprintf("TTicModel::SetSquareLetter (long  whichSquare, char letter)
            \n");

  fSquares[whichSquare] = letter;
}
```

TicTacToe – ticmodel.C

```
char
TTicModel::GetNextPlay ()
{
  qprintf("TTicModel::GetNextPlay () \n");
  return fNextLetter = fNextLetter == 'X' ? 'O' : 'X' ;
}

void
TTicModel::SetNextPlay (char letter)
{
  qprintf("TTicModel::SetNextPlay (char letter) \n");
  fNextLetter = letter;
}

void
TTicModel::NewGame ()
{
  qprintf("TTicModel::NewGame () \n");
  fNextLetter = 'X';
  for (int i = 0;i < 9;i++)
    fSquares[i] = 0;
  NotifyOfDrasticChange();
}

TInterest
TTicModel::GetDrasticChangeInterest() const
{
  qprintf("TTicModel::GetDrasticChangeInterest() const  \n");

  static const TToken kDrasticChange("TTicModel::kDrasticChange");

  return TModelInterest(*this, kDrasticChange);
}
```

TicTacToe – ticmodel.C

```
void
TTicModel::NotifyOfDrasticChange()
{
  qprintf(”TTicModel::NotifyOfDrasticChange()  \n”);

  NotifyOfChange(TNotification(GetDrasticChangeInterest()));
}
```

TicTacToe – tic.PinkMake


```
library tic6Lib {

source:
	tempinst.C
	ticpres.C
	ticview.C
	ticmodel.C
	ticselec.C
	ticcomma.C

data:
	tic6App

link:
	ViewSystemLib
	BasicDocumentLib
	CompoundDocumentLib
	GUICompoundDocumentLib
	PropertyInterfaceLib
	ActionLib
	ConstructorArchiveLib
	InputLib
	MouseInputLib
	HighLevelAlbert
	LowLevelAlbert
	NewGraphicApplicationLib
	NewControlsLib

	{SimpleLinkList}
}
```

TicTacToe – ticApp

```
#!/bin/ksh
if [ –n ”$1” –a –d $TaligentRoot/$1 ]; then
   RunDocument –o $1
else
    RunDocument –c –o ”TGUIModelPresenterStationeryFor<TTicModel,TBoardPresenter>” tic6Lib
$1
fi
```

TicTacToe – tempinst.C


```
#ifndef HIDE_TEMPLATE_DEFINITIONS
#define HIDE_TEMPLATE_DEFINITIONS
#endif

#define MAKE_TRequestBasedMemberFunctionConnectionTo_DEFINITIONS_VISIBLE
#define MAKE_TNotificationMemberFunctionOf_DEFINITIONS_VISIBLE
#define MAKE_TMomentaryPresenterCallbackTo_DEFINITIONS_VISIBLE
#define MAKE_TPresenterCallbackRequestTo_DEFINITIONS_VISIBLE
#define MAKE_TDocumentComponentCommandBindingTo_DEFINITIONS_VISIBLE
#define MAKE_TGUIDocumentComponentCommandBindingTo_DEFINITIONS_VISIBLE
#define MAKE_TCommandControlStateOn_DEFINITIONS_VISIBLE
#define MAKE_TCommandOn_DEFINITIONS_VISIBLE
#define MAKE_TGUIModelSelectionFor_DEFINITIONS_VISIBLE
#define MAKE_TGUIModelPresenterStationeryFor_DEFINITIONS_VISIBLE
#define MAKE_TGUIPresenterStateFor_DEFINITIONS_VISIBLE
#define MAKE_TModelPointerTo_DEFINITIONS_VISIBLE

#pragma defineonly

#include "ticmodel.h"
#include "ticpres.h"
#include "ticselec.h"
#include "ticcomma.h"
#include "ticview.h"

#pragma define( TRequestBasedMemberFunctionConnectionTo<TSquareView>)
#pragma define( TNotificationMemberFunctionOf<TSquareView>)
#pragma define( TMomentaryPresenterCallbackTo<TBoardPresenter>)
#pragma define( TPresenterCallbackRequestTo<TBoardPresenter>)
#pragma define( TDocumentComponentCommandBindingTo<TTicSelection> )
#pragma define( TGUIDocumentComponentCommandBindingTo<TTicSelection> )
#pragma define( TCommandControlStateOn<TTicSelection> )
#pragma define( TCommandOn<TTicSelection> )
#pragma define( TGUIModelSelectionFor<TTicModel> )
#pragma define( TGUIModelPresenterStationeryFor<TTicModel,TBoardPresenter> )
#pragma define( TGUIPresenterStateFor<TBoardPresenter> )
#pragma define( TModelPointerTo<TTicModel> )
```

TicTacToe – tempinst.C

```
#undef MAKE_TRequestBasedMemberFunctionConnectionTo_DEFINITIONS_VISIBLE
#undef MAKE_TNotificationMemberFunctionOf_DEFINITIONS_VISIBLE
#undef MAKE_TMomentaryPresenterCallbackTo_DEFINITIONS_VISIBLE
#undef MAKE_TPresenterCallbackRequestTo_DEFINITIONS_VISIBLE
#undef MAKE_TDocumentComponentCommandBindingTo_DEFINITIONS_VISIBLE
#undef MAKE_TGUIDocumentComponentCommandBindingTo_DEFINITIONS_VISIBLE
#undef MAKE_TCommandControlStateOn_DEFINITIONS_VISIBLE
#undef MAKE_TCommandOn_DEFINITIONS_VISIBLE
#undef MAKE_TGUIModelSelectionFor_DEFINITIONS_VISIBLE
#undef MAKE_TGUIModelPresenterStationeryFor_DEFINITIONS_VISIBLE
#undef MAKE_TGUIPresenterStateFor_DEFINITIONS_VISIBLE
#undef MAKE_TModelPointerTo_DEFINITIONS_VISIBLE
```