# Appendix B

TANDEM APPLICATION PROGRAMMER’S INTERFACE

## Contents

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TcAction — Abstract base template class for all Action classes</td>
<td>104</td>
</tr>
<tr>
<td>1.1</td>
<td>Handler — Nested class Handler</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>TcActionEvent — Abstract base class for TcActionEvents</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>TcActionEventObserver — ActionEvent observer class</td>
<td>110</td>
</tr>
<tr>
<td>4</td>
<td>TcApplication — Tandem core application base class</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>TcAudioPlayer — AudioPlayer Class</td>
<td>118</td>
</tr>
<tr>
<td>6</td>
<td>TcAvatarManager — Base Avatar Manager class</td>
<td>123</td>
</tr>
<tr>
<td>7</td>
<td>TcExControl — Core Execution Control Class</td>
<td>124</td>
</tr>
<tr>
<td>8</td>
<td>TcInputManager — Core InputManager class</td>
<td>125</td>
</tr>
<tr>
<td>9</td>
<td>TcInteractionManager — Core InteractionManager Class</td>
<td>127</td>
</tr>
<tr>
<td>10</td>
<td>TcIterator — Abstract base template Iterator Class</td>
<td>132</td>
</tr>
</tbody>
</table>
Appendix B (Continued)

<table>
<thead>
<tr>
<th></th>
<th>Class/Module Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>TcLoader — Class TcLoader</td>
<td>134</td>
</tr>
<tr>
<td>12</td>
<td>TrackerData — Tracker Data Class</td>
<td>136</td>
</tr>
<tr>
<td>13</td>
<td>TcTrackerManager — Core TrackerManager Class</td>
<td>137</td>
</tr>
<tr>
<td>14</td>
<td>TcTravelScheme — TcTravelScheme class</td>
<td>139</td>
</tr>
<tr>
<td>15</td>
<td>TcTraveler — Core Traveler Class</td>
<td>141</td>
</tr>
<tr>
<td>16</td>
<td>TcUserComponent — Abstract Base User Component Class</td>
<td>144</td>
</tr>
<tr>
<td>17</td>
<td>TcGfxPfBackgroundLoader — Background loader class</td>
<td>145</td>
</tr>
<tr>
<td>17.1</td>
<td>TcGfxLoadRequest — Nested Class TcGfxPfBackgroundLoader</td>
<td>146</td>
</tr>
<tr>
<td>17.1.1</td>
<td>TcGfxBgLoaderObserver — Nested Class TcGfxLoadRequest::TcGfxBgLoaderObserver</td>
<td>148</td>
</tr>
<tr>
<td>18</td>
<td>TcGfxCompList — Class TcGfxCompList</td>
<td>151</td>
</tr>
<tr>
<td>19</td>
<td>TcGfxIterator — Template iterator class for Gfx classes</td>
<td>154</td>
</tr>
<tr>
<td>20</td>
<td>TcGfxComponent — Abstract Base Class TcGfxComponent</td>
<td>156</td>
</tr>
<tr>
<td>20.1</td>
<td>LoadType — LoadType enumeration</td>
<td>159</td>
</tr>
<tr>
<td>20.2</td>
<td>LoadState — LoadState enumeration</td>
<td>160</td>
</tr>
<tr>
<td>21</td>
<td>TcGfxDisplayLibrary — Class TcGfxDisplayLibrary</td>
<td>162</td>
</tr>
<tr>
<td>22</td>
<td>TcGfxDspLibComponent — Abstract base class for Display Library Component Adapters .......................................................... 164</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>TcGfxGObject — Class is used to hold GL Gfx objects that implement the draw and update methods .................................................. 167</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>TcGfxGroup — Abstract base class for GfxGroups .................................................. 170</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>TcGfxIntersectionManager — Intersection Manager Class ....................... 173</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>TcGfxIntersectionTest — Abstract base class for Intersection Tests ...... 174</td>
<td></td>
</tr>
<tr>
<td>26.1</td>
<td>IsectState — enum IsectState .......................................................... 176</td>
<td></td>
</tr>
<tr>
<td>26.2</td>
<td>CoordType — enum CoordType .......................................................... 176</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>TcGfxLoadRequestPtr — Smart pointer class TcGfxLoadRequestPtr This class is instantiated on the stack in order to send a load request .......... 178</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>TcGfxManager — Class: TcGfxManager Role: This class manages Tandem’s graphics subsystem .......................................................... 179</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>TcGfxObject — Abstract Class TcGfxObject .................................................. 183</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>TcGfxPfComponentAdapter — Base class for all performer component adapters .......................................................... 185</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>TcGfxPfLibAdapter — Core Graphics Performer Library Adapter ........... 188</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>TcGfxVisitor — Abstract base class for visitors and controllers .......... 190</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B (Continued)

33 TdGfxGLGroup — This class is used to setup a tree of GL components

34 TdGfxPfDCSAdapter — pfDCS adapter class

35 TdGfxPfGrabController — Performer grab controller class

36 TdGfxPfGroup — Class TdGfxPfGroup

37 TdGfxPfGroupAdapter — pfGroup adapter class

38 TdGfxPfIntersectionTest — Performer Cavelib Intersection Test

38.1 Sensor — Sensor enum

39 TdGfxPfNodeAdapter — Class TdGfxPfNodeAdapter

40 TdGfxPfObject — Class TdGfxPfObject

41 TdGfxPfSCSAdapter — pfSCS adapter class

42 TdGfxPfTravAttributes — Performer traversal attributes class

43 TcNetCavConnectionManager — Role: This class plays the role of Tc-NetEvent shipper and receiver

44 TcNetCavEvent — CAVERNsoft Network Event

45 TcNetConnectionManager — Abstract Base Class for Network Connection Manager

46 TcNetEvent — Abstract Base Class for Network Events.
Appendix B (Continued)

<table>
<thead>
<tr>
<th>Page</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td>TcNetEventManager — Abstract base class for Event Manager</td>
</tr>
<tr>
<td>48</td>
<td>TcNetManager — Abstract base class for Network Manager</td>
</tr>
<tr>
<td>49</td>
<td>TdNetCavConnection — Class used to encapsulate a CAVERNsoft Connection</td>
</tr>
<tr>
<td></td>
<td>49.1 connectionType — Connection type enumeration</td>
</tr>
<tr>
<td>50</td>
<td>TdNetCavKey — Class used to encapsulate a CAVERNsoft key.</td>
</tr>
<tr>
<td>51</td>
<td>TdNetReceivingDock — Template class for queuing asynchronous network events.</td>
</tr>
<tr>
<td>52</td>
<td>TcUtDebug — Debug utility class used throughout Tandem</td>
</tr>
<tr>
<td>53</td>
<td>TcUtMutex — Base class for Tandem mutex</td>
</tr>
<tr>
<td></td>
<td>53.1 lockResult_t — enum lock Results</td>
</tr>
<tr>
<td>54</td>
<td>TcUtSemMutex — Mutex class built on posix semaphore for single process, threaded apps.</td>
</tr>
<tr>
<td>55</td>
<td>TcUtSemMutex_mp — Mutex class built on posix semaphore for multi process apps.</td>
</tr>
<tr>
<td>56</td>
<td>TcUtObserver — Abstract Base class for observers</td>
</tr>
<tr>
<td>57</td>
<td>TcUtString — Tandem String Class</td>
</tr>
<tr>
<td>58</td>
<td>operator&lt;&lt; — Output</td>
</tr>
</tbody>
</table>
Appendix B (Continued)

59  operator>> — Input ................................................................. 280
60   getline — Read line ............................................................... 281
61  operator== — Compare == .......................................................... 282
62  operator!= — Compare != ............................................................ 283
63  operator< — Compare < ............................................................... 284
64  operator<= — Compare <= ............................................................ 285
65  operator> — Compare > ............................................................... 286
66  operator>= — Compare >= ............................................................ 287
67  TcUtStringHashTable — Template String Hash Table ....................... 288
67.1  iterator — Nested Iterator class ............................................. 289
68  TcUtVector3f — Tandem 3 float utility vector ................................ 293

Class Graph ................................................................. 295
Appendix B (Continued)

```cpp
template <class T> class TcAction
```

*Abstract base template class for all Action classes*

**B.0.0.0.1 Public Members**

1.1 class **Handler**

```
Nested class Handler ................. 105
```

virtual ~**TcAction** ()

```
Destructor
```

1.2 static void **addHandler** (Handler* aHandler)

```
This method will register an action handler for notification ................. 106
```

1.3 static void **removeHandler** (Handler* aHandler)

```
This method will unregister an action handler ......................... 107
```

```
void **broadcast** ()
```

```
This method will notify each registered action handler.
```

```
static void **ts_addHandler** (Handler* aHandler)
```

```
Thread safe version of addHandler.
```

```
static void **ts_removeHandler** (Handler* aHandler)
```

```
Thread safe version of removeHandler.
```

```
void **ts_broadcast** ()
```

```
Thread safe version of broadcast.
```

**B.0.0.0.2 Protected Members**

1.4 **TcAction** ()

```
Constructor ......................... 107
```

```
static void **ts_notify** (TcAction<T>* t)
```

```
Thread safe version of notify.
```

Appendix B (Continued)

Abstract base template class for all Action classes. Clients must inherit from this class in order to define an Action.

1.1

```cpp
class Handler
```

*Nested class Handler*

B.0.0.3 Public Members

1.1.1 `Handler (bool autoRegister = true)`

*Default Constructor*

1.1.2 `virtual ~Handler ()`  

*Destructor*

1.1.2 `virtual int handleAction (const T& t)`

*Pure virtual member function*

Nested class Handler. This class is a nested member class of each action class derived from `TcAction`. Action handlers are defined by inheriting from this nested class.

1.1.1 `Handler (bool autoRegister = true)`

*Default Constructor*

Default Constructor. User can turn off automatic action handling. Default value sets up any class inheriting from Handler as an implicit action handler. Clients must implement int
handleAction<\texttt{const T& t}); To turn off automatic action handling you would have code similar to the following in the constructor for an action handler: MyActionHandler::MyActionHandler() : MyAction::Handler(false)

1.1.2

virtual 
\texttt{int handleAction (const T& t)}

\textit{Pure virtual member function}

Pure virtual member function. This method must be implemented by each Action Handler. It is called whenever a broadcast occurs for a given action.

1.2

\texttt{static void addHandler (Handler* aHandler)}

\textit{This method will register an action handler for notification}

This method will register an action handler for notification. To turn on action handling you would call the static method MyAction::addHandler(myActionHandler) where myActionHandler is an instance of MyActionHandler.
1.3

static void removeHandler (Handler* aHandler)

This method will unregister an action handler.

This method will unregister an action handler. To turn off action handling you would call the static method MyAction::removeHandler(myActionHandler) where myActionHandler is an instance of MyActionHandler.

1.4

TcAction ()

Constructor

Constructor. This makes TcAction an abstract base class.
Appendix B (Continued)

2

class TcActionEvent

Abstract base class for TcActionEvents

B.0.0.0.4 Public Members

TcActionEvent ()  Default Constructor.

2.1 TcActionEvent (const TcUtString &name)  Constructor  .........................  108

virtual ~TcActionEvent ()  Destructor.

2.2 void setName (const TcUtString & name)  Sets the name of the ActionEvent  .....  109

const TcUtString &

getName ()  Returns the name of this ActionEvent.

2.3 virtual void doEvent ()  This method must be overridden by derived classes  .................  109

Abstract base class for TcActionEvents. Clients must inherit from this class in order to define an ActionEvent. Objects of this Class can be associated with TcGfxComponents. See TcGfxComponent::addEvent

2.1

TcActionEvent (const TcUtString &name)  

Constructor
Parameters: name — An optional string name can be used to identify an ActionEvent. This must be set either here or by the setName method before adding to a TcGfxComponent.

void setName (const TcUtString& name)

Sets the name of the ActionEvent.

Parameters: name — This is the name used by the TcGfxComponent::getEvent method.

virtual void doEvent ()

This method must be overridden by derived classes. This implements the behavior of the event. It is called by TcGfxComponent::triggerEvent.
class TcActionEventObserver

ActionEvent observer class

B.0.0.5 Public Members

TcActionEventObserver ()
  Constructor.

virtual ~TcActionEventObserver ()
  Destructor.

virtual void handleActionEvent (TcActionEvent *event)
  All ActionEventObserver classes must implement this method.

ActionEvent observer class. Clients may wish to specify observers of a given ActionEvent. They may do so by inheriting from this class and implementing the handleActionEvent method. The management of observer to ActionEvent mappings is left to the client. Clients must store a list of observers for a given ActionEvent. Upon triggering that ActionEvent via the doEvent call, they must iterate through the observer list for that event, and call handleActionEvent. The event instance is passed as argument to this method. This may be handled by the ActionEvent itself in the future.
Appendix B (Continued)

class TcApplication

Tandem core application base class

B.0.0.0.6 Public Members

TcApplication (int argc, char *argv[]) Constructor.

virtual ~TcApplication () Destructor.

void tdAppUpdate () Not yet implemented .................

void runTcApp () This is called after the call to config() to

start the application execution loop.

int config () This must be called in the constructor

of derived classes to trigger the factory

methods.

B.0.0.0.7 Protected Members

4.2 virtual TcNetManager*
Appendix B (Continued)

`createNetworkManager` (int argc, char* argv[],
long& clientLocalPort,
long& clientRemotePort,
bool& clientRemoteConnect,
TcUtString& clientUserName,
TcUtString& clientWorldServer,
TcUtString& clientWorldName)

Factory Method ......................... 114

4.3 virtual TcInteractionManager*
`createInteractionManager` (TcGfxManager &gfxManager)
Factory Method ......................... 115

4.4 virtual TcGfxManager*
`createGfxManager` (int argc, char *argv[],
TcLoader *clientLoader)
Factory Method ......................... 116

4.5 virtual TcAvatarManager*
`createAvatarManager` ()
Factory Method ......................... 116

4.6 virtual TcLoader*
`createLoader` ()
Factory Method ......................... 117

4.7 virtual TcAudioPlayer*
`createAudioPlayer` ()
Factory Method ......................... 117

const TcNetManager*
`getNetManager` ()  This provides access to Tandem’s net-
working manager class.

TcGfxManager*
`sceneManager`  The scene manager provides access to
Tandem’s graphics subsystem.

TcAudioPlayer*
Appendix B (Continued)

audioPlayer  

The audioPlayer provides access to Tandem’s audio client.

Tandem core application base class. Every tandem application must inherit from this class. TcApplication is based on the Factory Method design pattern. Clients override the create methods to get the desired modules.

4.1

void tdAppUpdate ()

Not yet implemented

Not yet implemented. If need be this method will allow clients to add functionality outside of the interaction methods.
virtual TcNetManager* createNetworkManager (int argc, char* argv[],
long& clientLocalPort,
long& clientRemotePort,
bool& clientRemoteConnect,
TcUtString& clientUserName, 
TcUtString& clientWorldServer,
TcUtString& clientWorldName)

Factory method

Factory method. This method will return a client network manager, and NULL if one is not provided.
Appendix B (Continued)

**Parameters:**

- **clientLocalPort** — the port the client will receive on.
- **clientRemotePort** — the port on the server that the client will connect to.
- **clientRemoteConnect** — a flag indicating whether to connect to a server or not.
- **clientUserName** — The client's user name
- **clientWorldServer** — The url of the host.
- **clientWorldName** — The name of the world to connect to.

```cpp
virtual TcInteractionManager* createInteractionManager (TcGfxManager &gfxManager)
```

*Factory Method*

Factory Method. This method will return a default interactionManager if not overridden by the client.

**Parameters:**

- **gfxManager** — This is the scene manager.
virtual TcGfxManager* createGfxManager (int argc, char *argv[],
TcLoader *clientLoader)

Factory Method

Factory Method. Generally this method is not overridden and the Tandem core graphics manager
is used.

virtual TcAvatarManager* createAvatarManager ()

Factory Method

Factory Method. The application base class will hold a bare bones TcAvatarManager. Eventually Tandem will include a default avatar implementation. For now look at the
MarsBuilderapplicationforanexample.

virtual TcLoader* createLoader ()

Factory Method
Factory Method. This method will return a client loader and NULL if one is not provided.

4.7

```cpp
virtual TcAudioPlayer* createAudioPlayer()
```

*Factory Method*

Factory Method. This method will return the default Tandem audio client class. It is available from within the InteractionManager, set by this class on startup.
### 5

**class TcAudioPlayer**

**AudioPlayer Class**

#### B.0.0.0.8 Public Members

| 5.1 | `TcAudioPlayer` (const `TcUtString &soundDir`) | Constructor | 119 |
| 5.2 | `~TcAudioPlayer` () | Destructor | 120 |
| 5.3 | void `playSound` (const `TcUtString &name`, int isLooped) | Starts play of a sound clip | 120 |
| 5.4 | void `forwardSound` () | Stops the currently playing sound clip and plays the next clip | 120 |
| 5.5 | void `backSound` () | Stops the currently playing sound clip and plays the previous clip | 121 |
| 5.6 | void `addSound` (const `TcUtString &path`, const `TcUtString &filename`) | Adds a new playable clip to the list of sound files | 121 |
| 5.7 | void `playFirstSound` () | Start playing the first sound in the list as determined by an ascending alphabetical sorting of the sound file names |
| 5.8 | void `stopSound` (const `TcUtString &name`) | Stops the sound from playing | 122 |
| 5.9 | void `stopCurrentSound` () | | |
Appendix B (Continued)

Stops the last sound clip in which play was initiated.

5.8 void rewindSound (const TcUtString &name)
   Rewind the sound file

AudioPlayer Class. This class is currently implemented using Dave Pape’s Bergen Sound Server. A directory of .aiff files is read at class instantiation. Offers similar functionality to a CD player. Clients can play, loop, stop/pause, rewind, add and delete sound clips. The server is assumed to be running on the machine on which this class is instantiated.

5.1

TcAudioPlayer (const TcUtString &soundDir)

Constructor

Parameters: soundDir — This is the full path name of the directory containing the .aiff files.

5.2

~TcAudioPlayer ()

Destructor
Destructor. This will kill all currently playing sounds.

### 5.3

```cpp
void playSound (const TcUtString &name, int isLooped)
```

*Starts play of a sound clip*

Starts play of a sound clip. This works similar to the function of a CD player in that will resume play from the point the clip was paused, or start from the beginning if it was not previously playing. If a sound has already been played it must be rewound before it can be heard again.

**Parameters:**

- The — name of the soundfile to be played. This must have been read in at class instantiation or by an explicit call to add sound.
- isLooped — If the value is 0 the sound will be played once. Otherwise it will be looped.

### 5.4

```cpp
void forwardSound ()
```

*Stops the currently playing sound clip and plays the next clip*

Stops the currently playing sound clip and plays the next clip. The order is sorted alphabetically determined by ascii values.
5.5

```cpp
void backSound ()
```

*Stops the currently playing sound clip and plays the previous clip*

Stops the currently playing sound clip and plays the previous clip. The order is sorted alphabetically determined by ascii values.

5.6

```cpp
void addSound (const TCHARString &path, const TCHARString &filename)
```

*Adds a new playable clip to the list of sound files*

Adds a new playable clip to the list of sound files. This sound file be inserted alphabetically into the list determined by ascii value. String literals are accepted as arguments.

**Parameters:**

- `path` — This is the absolute path not including the file name of the file to be added.
- `filename` — This is the file name to be added.
5.7

void stopSound (const TcUtString &name)

*Stops the sound from playing*

Stops the sound from playing. If the sound is currently playing it serves as a pause. If the sound has reached the end it serves as a rewind.

**Parameters:**

name — The name of the sound file that one wishes to stop.

5.8

void rewindSound (const TcUtString &name)

*Rewind the sound file*

Rewind the sound file. Exactly the same behavior as stopSound. Provided only as a naming convenience, to maintain coherency. Whenever a sound is played it is a good idea to call rewindSound to ensure that is not already at the end from a previous play.
class TcAvatarManager

Base Avatar Manager class

B.0.0.0.9 Public Members

   TcAvatarManager () Constructor.
   virtual ~TcAvatarManager () Destructor.

Base Avatar Manager class. This class is intended to provide Core avatar management features. Currently this is implemented in the MarsBuilder application. Eventually it will be moved up into this class.
Appendix B (Continued)

class TcExControl

Core Execution Control Class

B.0.0.0.10 Public Members

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

TcNetManager* clientNetManager,

TcGfxManager & clientSceneManager,

TcInteractionManager &

clientInteractionManager)

Constructor

virtual ~TcExControl () Destructor

void startApp () Starts the execution control loop.

Core Execution Control Class. This class encapsulates the core execution control loop. It is not intended to be used by clients.
class TcInputManager

Core InputManager class

8.0.0.0.11 Public Members

enum buttonChange Possible values returned by getButtonChange

enum buttonState Possible values returned by getButtonState

void tdUpdateInputDevices ()

Called each frame by the Core Interaction manager to update device state.

8.1 buttonChange

getButtonChange (const int &buttonNum) const

Returns whether there has been a change in button state since the last frame

8.2 buttonState getButtonState (const int &buttonNum) const

Returns the current button state

Core InputManager class. This provides an interface to the obtaining the state of wand buttons. Clients can access this class through the Interaction manager.
Appendix B (Continued)

Returns whether there has been a change in button state since the last frame

Parameters:

buttonNum — Determines what button number to query.

---

8.2

buttonState  **getButtonState** (const int &buttonNum) const

*Returns the current button state*

Parameters:

buttonNum — Determines what button number to query.
### Core InteractionManager Class

#### Public Members

**TcInteractionManager** (TcGfxManager &gfxM)

Constructor.

```cpp
virtual ~TcInteractionManager ()
```

Destructor.

9.1 int **config ()**

This must be called in the Constructor of a derived class before any of the aggregate members can be accessed

```cpp
bool isConfigured ()
```

This determines if config has been called.

```cpp
void updateInteractions ()
```

This is called from the Core Execution Control loop.

9.2 virtual TcTrackerManager*

```cpp
createTrackerManager ()
```

Factory Method

9.3 virtual TcGfxIntersectionManager*

```cpp
createIntersectionManager ()
```

Clients must override this if they wish to provide a custom IntersectionManager

9.4 virtual TcInputManager*

```cpp
createInputManager ()
```

Factory Method

```cpp
const TcInputManager&
```
Appendix B (Continued)

getInputManager() Returns a reference to the inputManager.

virtual void doTdInteractions() This must be overridden by derived classes to implement customized interactions.

virtual void doTdTravel() This must be overridden by derived class to implement customized travel.

const TcGfxManager& getGfxManager() Returns the graphicsManager.

9.5 void setTreeDebug(const bool &b) Sets a debugging mode attached to buttons 2 3 Button 2 will toggle Load the current TcGfxComponent

B.0.0.0.13 Protected Members

TcTrackerManager* trackerManager TrackerManager
TcGfxIntersectionManager* intersectionManager IntersectionManager
TcInputManager* inputManager InputManager
TcGfxManager& sceneManager GraphicsManager

Core InteractionManager Class. Role: This class performs the core interaction updates. It is an aggregate consisting of a TrackerManager, a GraphicsIntersectionManager, an InputManager, a Traveler, a GraphicsManager and a NetworkManager.
Appendix B (Continued)

Responsibilities: It is responsible for instantiating the Input Manager, the TrackerManager, and the GraphicsIntersectionManager. These are the components updated during core interaction updates.

9.1

```cpp
int config()
```

*This must be called in the Constructor of a derived class before any of the aggregate members can be accessed.*

This must be called in the Constructor of a derived class before any of the aggregate members can be accessed. This includes the InputManager, TrackerManager, and GraphicsIntersectionManager.

9.2

```cpp
virtual TcTrackerManager* createTrackerManager()
```

*Factory Method*

Factory Method. Clients must override this if they wish to provide a custom TrackerManager. They will have to implement the interface of the default tracker manager. Currently one is provided that is implemented on top of the CAVElib.
Clients must override this if they wish to provide a custom IntersectionManager.

Clients must override this if they wish to provide a custom IntersectionManager. They will have to implement the interface of the default intersectionManager. Currently one is provided.

Factory Method.

Factory Method. Clients must override this if they wish to provide a custom InputManager. They will have to implement the interface of the default inputManager. Currently one is provided that is implemented on top of the CAVElib.

Sets a debugging mode attached to buttons 2 3 Button 2 will toggle load the current TcGfxComponent.
Appendix B (Continued)

Sets a debugging mode attached to buttons 2, 3. Button 2 will toggle. Load the current TeGfxComponent. Button will iterate through the tree. Results are printed to standard out.
Appendix B (Continued)

```
template <class Item> class TcIterator

Abstract base template Iterator Class
```

B.0.0.0.14 Inheritance

```
TcIterator
```

B.0.0.0.15 Public Members

```
virtual void first () Set the iterator to the first element.
virtual void next () Advance the iterator to the next element.

virtual bool done () const Returns true if the end of the set has been reached.
virtual Item currentItem () const Returns the item the iterator is currently at.
```
Appendix B (Continued)

Abstract base template Iterator Class. This class is used to provide an interface for Tandem Iterators. In particular it is used by the TcGfxIterator.
Appendix B (Continued)

11

class TcLoader

Class TcLoader

B.0.0.0.16 Public Members

TcLoader () Constructor.
virtual ~TcLoader () Destructor.
virtual TcGfxComponent* init TanComponents ()
This method must be overridden to initialize the scene from compiled code.

virtual void initGLComponents ()
This must be overridden to initialize openGL components.

11.1 virtual void initSharedMem ()
This method is optionally overloaded to initialize shared memory ..............

void setSceneComponent (TcGfxComponent *sceneNode)
Used by the Core classes.
void setSoundPlayer (TcAudioPlayer *sndPlayer)
Used by the Core classes.
void setNavComponent (TcGfxComponent *navNode)
Used by the Core classes.
void setGLComponent (TcGfxComponent *glComp)
Used by the Core classes.

B.0.0.0.17 Protected Members
Appendix B (Continued)

TcGfxComponent*
  navNode  The navigation node.
TcGfxComponent*
  sceneNode  The root scene node.
TcGfxComponent*
  glRoot  The GL node.
TcAudioPlayer*
  soundPlayer  The Core Audio Player.
ifstream  gfxFileIn  Used to read a scene file.

Class TcLoader. This class is core the TcGfxComponent loader. Clients must inherit from this
class and override the interface.

11.1

virtual void initSharedMem ()

This method is optionally overloaded to initialize shared memory

This method is optionally overloaded to initialize shared memory. This call is made just before
Performer forks off its app, cull, draw processes.
class TrackerData

Tracker Data Class

B.0.0.0.18 Public Members

TrackerData () Constructor.

~TrackerData () Destructor.

const TcUtVector3f&
getPosition () const Returns the position vector.

const TcUtVector3f&
getOrientation () const Returns the orientation vector.

const TrackerData&
TrackerData::operator Assignment operator.

Tracker Data Class. This class encapsulates a rotation and position vector for a given tracker sensor.
Appendix B (Continued)

class TcTrackerManager

Core TrackerManager Class

B.0.0.0.19 Public Members

13.1 TcTrackerManager (const int numT =

        DEFAULT_NUMTRACKERS)
Constructor ............................  138
virtual ~TcTrackerManager ()
Destructor.
TrackerData*
        getTrackerData (int sensorNum)
Returns a pointer to tracker data for a
given sensor number.

const TcUtVector3f&
        getSensorPosition (int sensorNum )
Returns a vector containing the position
of a given sensor number in world coor-
dinate system.

const TcUtVector3f&
        getSensorOrientation (int sensorNum)
Returns a vector containing 3 rotation
angles (x, y, z) for a given sensor number
in world coordinate system.

13.2 void tdUpdateTrackers () Called by the Core Interaction manager
during core interaction updates .......  138
Core TrackerManager Class. This class takes care of updating the set of tracker sensors for a VR system.

13.1

**TcTrackerManager** (const int numT = DEFAULT_NUMTRACKERS)

*Constructor*

**Parameters:**

- `numT` — Determines how many sensors will be updated. Default is 2. Typically sensor 0 is the head tracker and sensor 1 is the wand tracker.

13.2

`void tdUpdateTrackers ()`

*Called by the Core Interaction manager during core interaction updates*

Called by the Core Interaction manager during core interaction updates. It is marked td to indicate that a default implementation is provided.
class TcTravelScheme

TcTravelScheme class

B.0.0.0.20 Public Members

TcTravelScheme (TcGfxComponent *navComp)
Constructor.

virtual ~TcTravelScheme () Destructor.

virtual void travel () Perform a travel update.

float compute_nav_height () Computes the navigation height.

int checkForWall (float xvec, float yvec, float zvec) Ensures the user isn’t walking through a wall.

int checkCaveWalk () Moves the world back if a user physically tried to walk through a wall.

void setMaxHeight (const float &height) Sets the maximum height the user can jump.

B.0.0.0.21 Protected Members

TcTravelScheme () Default constructor called by derived classes.

TcTravelScheme class. This class encapsulates the algorithm used to define travel in a VE.
Clients can inherit from this class and override the travel method to define their own algorithms. By default a simple walk scheme with collision detection is provided as the default travel scheme. This scheme is automatically instantiated by default.
Appendix B (Continued)

class TcTraveler

Core Traveler Class

B.0.0.0.22 Public Members

15.1 TcTraveler (TcTravelScheme *scheme)  
    Constructor  ......................  142

15.2 TcTraveler (TcGfxComponent *nav)  
    Constructor  ......................  142

    void handleTravel ()  
    This method is called by the core interaction manager.

15.3 void setTravelScheme (TcTravelScheme *scheme)  
    This method allows clients to change the travel scheme at run-time  ..........  143

    TcTravelScheme*  
    getTravelScheme () const  
    Returns the current travelScheme

    virtual ~TcTraveler ()  
    Destructor.

B.0.0.0.23 Protected Members

    TcTraveler ()  
    Constructor for derived classes.

Core Traveler Class. This class is the user’s traveler object. It is responsible for updating the users viewpoint by the handleTravel method. The TravelScheme encapsulates the algorithm used. It may be changed at run time.
15.1

TcTraveler (TcTravelScheme *scheme)

Constructor

Parameters: scheme — A travelScheme object must passed as argument.

15.2

TcTraveler (TcGfxComponent *nav)

Constructor

Parameters: nav — This is a TcGfxComponent containing the navigation node.

15.3

void setTravelScheme (TcTravelScheme *scheme)

This method allows clients to change the travel scheme at run-time
This method allows clients to change the travel scheme at run-time. The client is responsible for deleting the previous scheme.
class TcUserComponent

Abstract Base User Component Class

B.0.0.24 Public Members

TcUserComponent ()

Constructor.

~TcUserComponent ()

Destructor.

const TcUtString&

gGetComponentId () const

Returns the component id given to the instance.

B.0.0.25 Protected Members

void setComponentId (const TcUtString &value)

Used by derived classes to set the component id.

Abstract Base User Component Class. This class is used to associate user classes with TcGfxComponents. Any class that is derived from this class can be added as a user class to a graphics component. The class can then be retrieved by its string component id.
class TcGfxPfBackgroundLoader

Background loader class

B.0.0.0.26 Inheritance

TcGfxPfBackgroundLoader

B.0.0.0.27 Public Members

17.1 class TcGfxLoadRequest Nested Class TcGfxPfBackgroundLoader:

TcGfxPfBackgroundLoader ()
Constructor

~TcGfxPfBackgroundLoader ()
Destructor

static void addRequest (TcGfxLoadRequest *req)
Add a request to the request queue.

void initSharedMem ()
Initialize shared memory.

17.2 void handleRequest ()
Handle load requests in the application main process

void setPfScene (pfScene *scene)
Appendix B (Continued)

Used in initialization to place the scene node pointer into shared memory.

```c
void setPfNav (pfDCS *nav)
```

Used to place the navigation node pointer into shared memory.

```c
static void backgroundLoadCB (void *data)
```

This is the callback triggered by the DBASE process.

```c
static pfNode*
manualPfLookup (pfGroup *, const TcUtString&)
```

This performs a manual search for a node in performer when pfLookup fails.

B.0.0.0.28 Protected Members

```c
struct TcGfxPfSharedLoaderData
```

Nested Structure TcGfxPfBackgroundLoader::TcGfxPfSharedLoaderData

Background loader class. This class takes care of asynchronous loading of performer models. It must be properly initialized with the performer library and clients must inherit from the nested observer class to complete load requests. The classes are nested to avoid having to declare everything in the public interface or declare each class as a friend.

```c
class TcGfxLoadRequest
```

Nested Class TcGfxPfBackgroundLoader::TcGfxLoadRequest
Appendix B (Continued)

B.0.0.0.29  Public Members

17.1.1  class  

    TcGfxBgLoaderObserver

    Nested  Class  TcGfxLoad-

    Request::TcGfxBgLoaderObserver.

    TcGfxLoadRequest ()
    Constructor

    ~TcGfxLoadRequest ()
    Destructor

17.1.2  void  setCompName (const TcUtString& name)

    Set the name to be set in the pfNode ..

    void  setFileName (const TcUtString& name)

    Set the file name that the loader will

    search for in the path set by the PFPATH

    environment variable

17.1.3  void  setParentName (const TcUtString& name)

    Set the name of the parent node ....

    const TcUtString&
    getCompName () const

    Returns the component name.

    const TcUtString&
    getFileName () const

    Returns the file name.

    const TcUtString&
    getParentName () const

    Returns the parent name.

    void  registerRequest (TcGfxBgLoaderObserver *obs)

    Registers the observer and sends the re-

    quest.

    void  sendAnonymousRequest ()
Appendix B (Continued)

Sends an anonymous load request with no observer.

bool isAccepted ()

Returns true if the load request has been accepted.

int completeRequest ()

Informs observers that the request has been completed.

Nested Class TcGfxPfBackgroundLoader::TcGfxLoadRequest. This class is used internally by the TcGfxLoadRequestPtr and should not be instantiated by clients. Calls to operator-> sent to the TcGfxLoadRequestPtr class are redirected here.

17.1.1

class TcGfxBgLoaderObserver

Nested Class TcGfxLoadRequest::TcGfxBgLoaderObserver.

B.0.0.0.30 Public Members

TcGfxBgLoaderObserver (bool observeClass = false)

Constructor.

virtual ~TcGfxBgLoaderObserver ()

Destructor.

virtual int gfxLoadUpdate (const TcGfxPfBackgroundLoader::TcGfxLoadRequest& request)
Appendix B (Continued)

Clients must inherit from this class and override this function if they wish to be informed when the loader has completed the request.

17.1.2

```cpp
void setCompName (const TcUtString& name)
```

*Set the name to be set in the pfNode*

Set the name to be set in the pfNode. This will be the same name given to the TcGfxComponent.

17.1.3

```cpp
void setParentName (const TcUtString& name)
```

*Set the name of the parent node*

Set the name of the parent node. This node must already exist in the performer hierarchy prior to sending an ASYNC load request.
17.2

void handleRequest ()

Handle load requests in the application main process. This method completes processing of handled request and sends a new request to the background DBASE process.
class TcGfxCompList

Class: TcGfxCompList

B.0.0.31 Public Members

template <class Item> friend class
    TcGfxChildIterator Friend declaration.
    TcGfxCompList ()    Constructor
    ~TcGfxCompList ()   Destructor
const TcGfxCompList&
    operator=(const TcGfxCompList &right)    Assignment Operation

18.1 int addListItem (TcGfxComponent *item)
    Add a TcGfxComponent to the list .... 152

18.2 int removeListItem (TcGfxComponent *item)
    Remove a TcGfxComponent from the list

................................. 152

void printDebug ()    Print the items in the list to standard out
                      by CompName.

18.3 const TcGfxComponent*
    getCompByName (const TcUtString &name)
    Return a pointer to the item with the
    string name, name .................... 153

bool isEmpty ()    Returns true if the list is empty, and
                    false otherwise.

Class: TcGfxCompList.
Appendix B (Continued)

Role: This class provides an interface to a list of TcGfxComponents. Each child can be leaf nodes or other groups containing lists.

Responsibilities: It adds and removes TcGfxComponents. Returns dynamically allocated iterators to its children.

18.1

```
int addListItem (TcGfxComponent *item)
```

Add a TcGfxComponent to the list

Add a TcGfxComponent to the list. The compName must be defined in the item, and it must be unique in the list. A return value of 1 indicates a successful insertion, 0 means that the name already exists.

18.2

```
int removeListItem (TcGfxComponent *item)
```

Remove a TcGfxComponent from the list

Remove a TcGfxComponent from the list. A return value of 1 indicates a successful deletion, 0 means that the name already exists.
const TcGfxComponent* getCompByName (const TcUtfString &name)

Return a pointer to the item with the string name, name. Returns NULL if not found.

Parameters: name — String name of a TcGfxComponent.
Appendix B (Continued)

```
template <class Item> class TcGfxIterator : public TcIterator<Item>
```

Template iterator class for Gfx classes

B.0.0.0.32 Inheritance

```
TcIterator

\downarrow

TcGfxIterator
```

B.0.0.0.33 Public Members

19.1 TcGfxIterator () Constructor .......................... 155

19.2 TcGfxIterator (const Item root) Constructor .......................... 155

virtual ~TcGfxIterator () Destructor.

virtual void first () Sets the iterator to the root set by the construc-

virtual void next () Iterate to the next item.

virtual bool isDone () const Returns true if iterator has reached the

end of the set.

virtual Item currentItem () const Returns the item the iterator is currently
 pointing to.
Template iterator class for Gfx classes. Clients can instantiate an iterator by simply calling TcGfxComponent::iterator i(comp). This class can then be used to iterate through gfx components in a breadth first fashion.

19.1

TcGfxIterator ()

Constructor

Constructor. Returns an uninitialized iterator.

19.2

TcGfxIterator (const Item root)

Constructor

Constructor. This is the most common constructor to be used.

Parameters: root — This is the node from which iteration starts. The iterator is set to this node whenever first is called.
Appendix B (Continued)

class TcGfxComponent

Abstract Base Class TcGfxComponent

B.0.0.0.34 Inheritance

B.0.0.0.35 Public Members

enum LoadType
    LoadType enumeration

enum LoadState
    LoadState enumeration

virtual int addChild (TcGfxComponent *gfxNode)
    Add a child to this component.

virtual int removeChild (TcGfxComponent *gfxNode)
    Remove a child from this component.

virtual void loadGeometry (const LoadType &ldType = ASYNC)
    Loads the geometry.

virtual void unLoadGeometry ()
    Unload the geometry.

virtual void hideGeometry ()
    Hide the geometry.

virtual void unHideGeometry ()
    Unhide the geometry.
Appendix B (Continued)

virtual void toggleLoadGeometry ()
    Toggles load/unload geometry depending
    on the current state.

virtual void toggleHideGeometry ()
    Toggles hide/unhide geometry depending
    on the current state.

virtual void accept (TcGfxVisitor &)
    This is a visitor design pattern method.

virtual const TcGfxComponent*
    getCompByName (const TcUtString &)
    Searches for a given component in the
    tree rooted by this node.

virtual void triggerEvent (const TcUtString &eventName)
    Triggers the event with the given name
    associated with this component.

virtual TcActionEvent*
    getEvent (const TcUtString &eventName)
    Returns the event with the given name
    associated with this component.

virtual TcActionEvent*
    getIterativeEvent (const TcUtString &eventName)
    Returns the iterative event with the given
    name associated with this component.

virtual void handleIterativeEvents ()
    Iterates through the list of iterative events
    and triggers each one.

const TcUtString&
    getCompName ()
    Returns the name of this component.

void addEvent (TcActionEvent *)
    Associate an event with this component.

void removeEvent (TcActionEvent *)
Removes an event from this component.

```c
void addIterativeEvent (TcActionEvent *)
    Associates an iterative event to this component.
```

```c
void removeIterativeEvent (TcActionEvent *)
    Removes an iterative event from this component.
```

20.4 void addUserComponent (const TcUtString &name,
                             TcUserComponent *component)
    Add a user component with the given name to the UserClass table.

```c
TcUserComponent* getUserComponent (const TcUtString &name)
    Return the component with the given name and NULL if it is not found.
```

```c
void removeUserComponent (const TcUtString &)
    remove from table but doesn't delete component
```

```c
const vector <TcActionEvent *> * getEventList ()
    Returns a pointer to the entire list of events associated with this component.
```

```c
TcUtString getGeoFileName ()
    Returns the file name of the geometry associated with this component.
```

```c
void setGeoLoaded (bool) Sets the geometry loaded flag.
```

```c
void setGeoHidden (bool) Sets the geometry hidden flag.
```

```c
void setCompName (const TcUtString &name)
    Sets the component name.
```

```c
bool geoLoaded ()
    Returns true if the geometry is loaded.
```

```c
bool geoHidden ()
    Returns true of the geometry is hidden.
```

```c
TcGfxComponent *
```
Appendix B (Continued)

```c
getParent ()
Returns the parent of this component.

void setParent (TcGfxComponent *)
Sets the parent of this component.

void setDspLibComponent (TcGfxDspLibComponent *)
Sets the display library component.

TcGfxDspLibComponent*
getDspLibComponent () const
Returns the display library component.
```

### B.0.0.0.36 Protected Members

20.5 TcUtString geometryFileName Attribute: geometryFileName Description: This attribute stores the file location of the corresponding geometry file

Abstract Base Class TcGfxComponent.
This is the root class for all Tandem Graphics Components.

Role: This abstract base class forms the composite pattern by letting clients treat TcGfxObjects and TcGfxGroups uniformly.

Responsibilities: It provides an interface to TcGfxObjects and TcGfxGroups

```
 enum LoadType

LoadType enumeration
```
Appendix B (Continued)

Parameters:
- ASYNC — Asynchronous load type
- SYNC — Synchronous load type

20.2

```cpp
enum LoadState
```

*LoadState enumeration*

Parameters:
- LOADED — Loads the component during instantiation if the LoadType is set to SYNC otherwise it is ignored.
- UNLOADED — The component is not loaded. Can be used in combination with SYNC or ASYNC

20.3

```cpp
virtual void loadGeometry (const LoadType &ldType = ASYNC)
```

*Loads the geometry*
Parameters:  

ldType — Determines whether the load will be asynchronous or synchronous.

```c
void addUserComponent (const TcUtString &name, TcUserComponent *component)
```

Add a user component with the given name to the UserClass table. This is one way clients can add additional features/attributes to a component without subclassing. It is especially useful for adding application specific properties and behavior. WARNING - Because it uses SGI's STL hash_map, `name` must be preallocated and exist for the lifetime of the user component. The preferred way of dealing with `getName()` method as argument to `name`.

```
TcUtString geometryFileName
```

Attribute: `geometryFileName` Description: This attribute stores the file location of the corresponding geometry file. For instance it could be a performer binary (.pfb) or an inventor file (.iv)
class TcGfxDisplayLibrary

Class TcGfxDisplayLibrary

B.0.0.0.37 Inheritance

TcGfxDisplayLibrary -> TcGfxPfLibAdapter

B.0.0.0.38 Public Members

TcGfxDisplayLibrary ()
Constructor.

TcGfxDisplayLibrary (TcLoader*tanLoader)
Constructor.

TcGfxDisplayLibrary (TcGfxComponent *scene,
TcGfxComponent *nav,
TcGfxComponent *Av)
Constructor.

virtual ~TcGfxDisplayLibrary ()
Destructor.

TcGfxComponent*
getSceneComponent ()
Returns Scene component.

TcGfxComponent*
Appendix B (Continued)

getNavComponent ()

Returns Navigation component.

TcGfxComponent*

getAvComponent ()

Returns Avatar component.

virtual void setViewPoint (const TcUtVector3f *rot,
const TcUtVector3f *trans)

Sets the view point.

virtual void setIntersectManager (TcGfxIntersectionManager *)

Sets Intersection manager.

Class TcGfxDisplayLibrary. This is the base class for all core graphics library classes. The current implementation provides support for IRIS Performer through the derived TcGfxPfLibAdapter. Future support for other libraries will be implemented by deriving from this class.
Appendix B (Continued)

class TcGfxDspLibComponent

Abstract base class for Display Library Component Adapters

B.0.0.39 Inheritance

22

TcGfxDspLibComponent

30

TcGfxPfComponentAdapter

B.0.0.40 Public Members

virtual const TcUtString&
    GetComponentId () const
    Returns the component id associated with
    this component adapter.

virtual const char*
    getCompId_cstr () const
    Returns a C style string pointer to the
    component id.

virtual void hideGeometry ()
    Virtual function used to hide geometry
    from the scene.

virtual void unhideGeometry ()
    Virtual function used to unhide geometry
    from the scene.
Appendix B (Continued)

virtual void loadGeometry() virtual function used to load a geometry component into the scene.

virtual void unloadGeometry() virtual function used to unload a geometry component from the scene.

virtual bool isActive() Returns true if the geometry component is loaded and false otherwise.

virtual ~TcGfxDspLibComponent Destructor.

void setComponentId(const TcUtString &value) Set the name of the component id.

B.0.0.0.41 Protected Members

TcGfxDspLibComponent() Default constructor.

22.1 TcGfxDspLibComponent (const TcUtString &name) Constructor

Abstract base class for Display Library Component Adapters. This class is used to adapt underlying display library components to the TcGfxComponent hierarchy. Each component adapter inherits from this base class and can then be associated with a TcGfxComponent.

22.1

TcGfxDspLibComponent (const TcUtString &name)

Constructor
Appendix B (Continued)

Parameters:

name — Component id to be set in the component.
Appendix B (Continued)

class TcGfxGLObjecl : public TcGfxObject

Class is used to hold GL Gfx objects that implement the draw and update methods

B.0.0.42 Inheritance

TcGfxGLObjecl

\[\text{V}\]

TcGfxObjec1

\[\text{V}\]

B.0.0.43 Public Members

TcGfxGLObjecl () Default constructor.
TcGfxGLObjecl (const TcUtString &name) Allows the compName to be set by the constructor.

virtual ~TcGfxGLObjecl () Destructor.

23.1 void loadGeometry (const LoadType& lType) Loads the openGL object into the scene

void unloadGeometry () Unloads the openGL from the scene.
virtual void hideGeometry () \textit{Hide the geometry so that it is still loaded, but not drawn.}

virtual void unHideGeometry () \textit{Make geometry that has been hidden visible.}

virtual void draw () \textit{Implemented by descendents and called once per iteration by Tandem Execution.}

virtual void update () \textit{Implemented by descendents and called once per iteration by Tandem Execution.}

virtual TcUtVector3f getPosition () \textit{Returns the position of the object.}

virtual int setPosition (TcUtVector3f) \textit{Sets the position of the object.}

virtual TcUtVector3f getOrientation () \textit{Returns the orientation of the object.}

virtual int setOrientation (TcUtVector3f) \textit{Sets the orientation of the object.}

virtual void accept (TcGfxVisitor &v) \textit{Used to accept TcGfxVisitors.}

\textbf{B.0.0.0.44  Protected Members}

bool* draw_gl \textit{Boolean flag that must be allocated from shared memory to be seen by the draw process in multi-processor and multi-processed systems}
Class is used to hold GL Gfx objects that implement the draw and update methods. Shared memory allocation must occur in constructor initialization.

23.1

```cpp
void loadGeometry (const LoadType& lType)
```

*Loads the openGL object into the scene*

**Parameters:**

- `lType` — Generally not used by GLObjects, but can be ASYNC or SYNC
Appendix B (Continued)

```cpp
class TcGfxGroup : public TcGfxComponent
```

Abstract base class for GfxGroups

B.0.0.0.45 Inheritance

```
<table>
<thead>
<tr>
<th>TcGfxComponent</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>TcGfxGroup</td>
<td></td>
</tr>
<tr>
<td>TdGfxPfGroup</td>
<td>36</td>
</tr>
<tr>
<td>TdGfxGLGroup</td>
<td>33</td>
</tr>
</tbody>
</table>
```

B.0.0.0.46 Public Members

```cpp
TcGfxGroup () Constructor.
TcGfxGroup (TcUtString name) Constructor.
TcGfxGroup (const TcUtString &name,
TcGfxPfComponentAdapter *dspLibComp)
```
Appendix B (Continued)

24.2

TcGfxGroup (const TcGfxGroup &right)  
Constructor .............................................. 171

Copy Constructor ................................. 172

virtual ~TcGfxGroup ()  
Destructor

const TcGfxGroup & 
operator= (const TcGfxGroup &right)  
Assignment Operation Not yet implemented

virtual int addChild (TcGfxComponent *gfxNode)  
Add a child to the child list.

virtual int removeChild (TcGfxComponent *gfxNode)  
Remove a child from the child list.

24.3

TcIterator <TcGfxComponent *> * 
getChildIterator ()  
Returns a dynamically instantiated iterator to a child component .............. 172

virtual const TcGfxComponent * 
getCompByName (const TcUtString &name)  
Returns a pointer to component in the sub-tree rooted at this node and NULL if not found.

24.1

TcGfxGroup (const TcUtString &name, TcGfxPfComponentAdapter *dspLibComp)  
Constructor
Appendix B (Continued)

Parameters:

name — Name given to the component.

dspLibComp — Display Library Component set in the base class.

24.2

TcGfxGroup (const TcGfxGroup &right)

Copy Constructor

Copy Constructor. Not yet implemented.

24.3

TcIterator <TcGfxComponent *> * getChildIterator ()

Returns a dynamically instantiated iterator to a child component

Returns a dynamically instantiated iterator to a child component. WARNING - The client is responsible for deleting this iterator.
class TcGfxIntersectionManager

Intersection Manager Class

B.0.0.0.47 Public Members

TcGfxIntersectionManager ()
Construct.

TcGfxIntersectionManager ()
Destructor.

virtual void tdUpdateIntersections ()
Update intersections. Can be overridden by descendents if necessary.

void addIntersectionTest (TcGfxIntersectionTest *test)
Add an intersection test to the manager.

doTcIntersectionTest will be called each iteration.

void deleteIntersectionTest (const TcUtString &id)
Remove an intersection with the given id.

TcGfxIntersectionTest*
getIntersectionTest (const TcUtString &id)
Returns an intersection test with the given id, and NULL otherwise.

Intersection Manager Class. This class handles the automatic update of intersectionTests that are added to it.
class TcGfxIntersectionTest

Abstract base class for Intersection Tests

B.0.0.0.48 Inheritance

B.0.0.0.49 Public Members

26.1 enum IsectState
26.2 enum CoordType
26.3 TcGfxIntersectionTest (const TcUtString &isectId)

virtual ~TcGfxIntersectionTest ()
    Destructor.

virtual IsectState
doTcIntersectionTest ()
    This method must be overridden by clients
to determine how an intersection is done.

const TcUtString&
Appendix B (Continued)

getIsectId () const  
This method returns the string identification of the intersection test.

void setIsectId (const TcUtString &)  
Sets the string identification for the intersection test.

const IsectState&
getLastUpdateState () const  
Determines if an intersection was found in the last call to doTcIntersection test.

const TcUtString&
getLastIsectName () const  
Returns the graphics component name of the last intersection.

B.0.0.0.50  Protected Members

void setIsectState (const IsectState &T)  
Sets the intersection state.

void setIsectName (const TcUtString &T)  
Sets the name of the graphics component intersected.

Abstract base class for Intersection Tests. This class specifies the interface for all Tandem core intersections.

26.1

enum IsectState

enum IsectState
Appendix B (Continued)

Parameters:

ISECT_NOTFOUND — No intersection was found.

ISECT_FOUND — An intersection was found.

ERROR — An error occurred.

26.2

```cpp
enum CoordType
```

Parameters:

WORLD — World coordinate system.

LOCAL — Local coordinate system.

26.3

```cpp
TcGfxIntersectionTest (const TcUtString &isectId)
```

Constructor
Appendix B (Continued)

**Parameters:**

- `isectId` — String identification given to the intersection test.
Smart pointer class TcGfxLoadRequestPtr This class is instantiated on the stack in order to send a load request.

B.0.0.0.51 Public Members

TcGfxLoadRequestPtr ()
Constructor.

~TcGfxLoadRequestPtr ()
Destructor.

TcGfxPfBackgroundLoader::TcGfxLoadRequest*
operator-> () Forwards requests to the underlying LoadRequest.

TcGfxPfBackgroundLoader::TcGfxLoadRequest&
operator* () Forwards requests to the underlying LoadRequest.

Smart pointer class TcGfxLoadRequestPtr This class is instantiated on the stack in order to send a load request. It encapsulates a TcGfxLoadRequest.
Appendix B (Continued)

28

class TcGfxManager

Class: TcGfxManager Role: This class manages Tandem’s graphics subsystem

B.0.0.52 Public Members

28.1 TcGfxManager (int argc, char *argv[], TcLoader *clientLoader)

Constructor  .........................  180

virtual ~TcGfxManager () Destructor.

bool drawScene () Updates draw routines handled by the TcGfxLibrary.

void updateScene () Updates GL components and interactive events.

bool updateTravel () Updates the travel scheme.

TcGfxComponent*

initTcGfxScene () Initialize the graphics scene.

void removeTcGfxComponent (TcGfxComponent *) Remove node from the TcGfxComponent hierarchy.

TcGfxComponent*

getAvComponent () Returns the Avatar Group.

28.2 TcGfxComponent*

getGfxCompByName (const TcUtString &name) const

Returns the TcGfxComponent with the given name in the tree and NULL if it does not exist  .........................  181

28.3 TcGfxComponent*
getGLCompByName (const TcUtString &name) const

Returns the TcGfxComponent with the

given name in the GL tree and NULL if

it does not exist .............................. 181

TcTraveler* getTcTraveler () const

Returns a pointer to the TcTraveler.

setIntersectionManager (TcGfxIntersectionManager &)

Sets the TcGfxIntersectionManager ...

setViewPoint (const TcUtVector3f *rot,

const TcUtVector3f *trans)

Sets the position and orientation of the

viewpoint in the world ........................ 182

Class: TcGfxManager Role: This class manages Tandem’s graphics subsystem. This includes drawing, searching, updating and maintaining its various objects.

Responsibilities: It provides an interface for searching TcGfxComponents. This class also updates the TcGfxLibrary’s draw routines. In addition it traverses the TcGfxComponent hierarchy each frame and updates iterative events.

28.1

TcGfxManager (int argc, char *argv[], TcLoader *clientLoader)

Constructor

Constructor. This class is instantiated internally by TcApplication.
Appendix B (Continued)

**Parameters:**
clientLoader — Class returned by createLoader in TcApplication.

---

### 28.2

TcGfxComponent* `getGfxCompByName` (const TcUtString &name) const

*Returns the TcGfxComponent with the given name in the tree and NULL if it does not exist.*

**Parameters:**
name — This can be a TcUtString or a string literal.

---

### 28.3

TcGfxComponent* `getGLCompByName` (const TcUtString &name) const

*Returns the TcGfxComponent with the given name in the GL tree and NULL if it does not exist.*

**Parameters:**
name — This can be a TcUtString or a string literal.
28.4

void setIntersectionManager (TcGfxIntersectionManager &)

Sets the TcGfxIntersectionManager

Sets the TcGfxIntersectionManager. This should not be done at runtime unless the client wants
to change the manager.

28.5

void setViewPoint (const TcUtVector3f *rot, const TcUtVector3f *trans)

Sets the position and orientation of the viewpoint in the world

Sets the position and orientation of the viewpoint in the world. Usually called by the interaction
manager at startup.
class TcGfxObject : public TcGfxComponent

Abstract Class TcGfxObject

B.0.0.0.53 Inheritance

TcGfxComponent

TcGfxObject

TdGfxPfObject

TcGfxGLObject

B.0.0.0.54 Public Members

TcGfxObject () Constructor.
TcGfxObject (TcUtString name) Constructor.
TcGfxObject (TcUtString nodeName, TcUtString fileName) Constructor.
TcGfxObject (TcUtString name,

    TcGfxDspLibComponent *dspLibComp)
Abstract Class TcGfxObject. This is the abstract class for all GfxObjects. It takes care of methods that involve traversing the component tree as well as issuing warnings for clients that try to add or remove children everywhere else remains abstract.

**Constructors.**

29.1 \[\text{TcGfxObject} \text{ (const TcGfxObject \\&right)} \]

*Copy constructor*

29.2 \[\text{const TcGfxObject \\& operator=} \text{ (const TcGfxObject \\&right)} \]

*Assignment*

Copy constructor. Not yet implemented.

Assignment. Not yet implemented.
class TcGfxPfComponentAdapter : public TcGfxDspLibComponent

*Base class for all performer component adapters*

B.0.0.0.55  **Inheritance**

```
TcGfxDspLibComponent

\downarrow

TcGfxPfComponentAdapter

\downarrow

TdGfxPfNodeAdapter

\downarrow

TdGfxPfGroupAdapter
```

B.0.0.0.56  **Public Members**

virtual ~TcGfxPfComponentAdapter ()

*Destructor.*

pfNode* getPerformerNode () const

*Returns the performer node.*

void setPerformerNode (pfNode *node)

*Sets the performer node.*

virtual void hideGeometry ()

*Hides the geometry from the scene.*
Appendix B (Continued)

virtual void unhidGeometry() Restores the geometry from the hidden state.

virtual void loadGeometry() Load geometry.

virtual void saveMasks() Saves the current APP, DRAW, ISECT masks.

virtual void unloadGeometry() Unloads the geometry from the scene.

virtual const char* getCompId_cstr() const
Return the char name pointer contained by the performer node.

void deletePerformerNode() Performs a pfDelete on the performer node.

void setCompName(const TcUtString &)
Sets the name of the pfNode.

void setTravAttributes(const TdGfxPfTravAttributes &atts)
Sets a traversal mask for a given mask type.

B.0.0.0.57 Protected Members

TcGfxPfComponentAdapter() Constructor.

TcGfxPfComponentAdapter(pfNode *node) Constructor .......................... 187

TcGfxPfComponentAdapter(const TcUtString &name) Constructor

void setPfName(const TcUtString &name)
Set the name of the performer node.

pfNode* perfNode Performer node.
30.1

**TcGfxPfComponentAdapter** (pfNode *node)

*Constructor*

**Parameters:**

node — A pre-instantiated performer node.
class TcGfxPfLibAdapter : public TcGfxDisplayLibrary

Core Graphics Performer Library Adapter

B.0.0.0.58 Inheritance

TcGfxDisplayLibrary

∨

TcGfxPfLibAdapter

B.0.0.0.59 Public Members

31.1 TcGfxPfLibAdapter (int argc, char *argv[], TcLoader *ldr,

TcGfxPfBackgroundLoader *bgldr)

Constructor ....................... 189

~TcGfxPfLibAdapter ()

Destructor.

void doOneGfxCycle () Perform one performer draw cycle.

TcGfxComponent*

getGLComponent () Returns the GL component initialized in

this class.

virtual void setViewPoint (const TcUtVector3f *rot,

const TcUtVector3f *trans)
Appendix B (Continued)

Sets the viewpoint orientation and translation.

Core Graphics Performer Library Adapter. This class adapts the Performer CAVE library. It takes care of initialization and draw updates.

31.1

TcGfxPfLibAdapter (int argc, char *argv[], TcLoader*ldr, TcGfxPfBackgroundLoader *bgldr)

Constructor

Parameters:  argc — argument passed from main() procedure.

argc — argument passed from main() procedure.

ldr — Pointer to the core loader.

bgldr — Pointer to the background loader.
Appendix B (Continued)

class TcGfxVisitor

Abstract base class for visitors and controllers

B.0.0.0.60 Inheritance

B.0.0.0.61 Public Members

TcGfxVisitor () Default constructor
virtual ~TcGfxVisitor () Destructor

32.1 virtual void visitGfxComponent (TcGfxComponent *c) Root base class visitor method ........... 191

32.2 virtual void visitGfxObject (TcGfxObject *o) Common base class TcGfxObject visit

32.3 virtual void visitGfxGroup (TcGfxGroup *g) Common base class TcGfxGroup method

virtual void visitGfxPfObject (TdGfxPfObject *o) TdGfxPfObject visit method.

virtual void visitGfxPfGroup (TdGfxPfGroup *g)
Appendix B (Continued)

TdGfxPfGroup visit method.

virtual void visitGfxGLObject (TcGfxGLObject *o)
TcGfxGLObject visit method.

virtual void visitGfxGLGroup (TdGfxGLGroup *g)
TdGfxGLGroup visit method.

Abstract base class for visitors and controllers. Description: This abstract base class provides an interface for all actions that must distinguish the type of TcGfxComponent in order to carry out their behavior. It also allows clients to extend the behavior of the TcGfxComponent hierarchy.

Collaborators: ConcreteVisitors inherit from this class and then implement each of these interface methods. A client who holds a TcGfxComponent instance calls accept(visitor v) and passes a visitor instance as argument. See visitGfxComponent() for further detail.

32.1

virtual void visitGfxComponent (TcGfxComponent *c)

Root base class visitor method

Root base class visitor method. Description: This method provides an interface to specify default behavior for a Concrete Visitor.

Comments: To specify default behavior, a concrete visitor would place it into this method: ConcreteVisitor::visitGfxComponent(TcGfxComponent c) ...

Then when implementing any of the class specific visitors the default behavior could be extended.

eg. ConcreteVisitor::visitPfGfxObject(TcGfxObject *o) visitGfxComponent(o); <...> <extended implementation code>

In addition this is a catchall for extensions to the TcGfx hierarchy. See visitGfxObject for an example.
32.2

virtual void visitGfxObject (TcGfxObject *o)

Common base class TcGfxObject visit method

Common base class TcGfxObject visit method. Description: This is an interface for default TcGfxObject visitor behavior, see visitGfxComponent.

Comments: Like visitGfxComponent, this is a catchall for extensions to the TcGfxComponent hierarchy.

For example consider a new class called MyClassObject which inherits from TcGfxObject. MyClassObject::accept(visitor v) Because the visitor interface is not able to include new classes without recoding we call default behavior v->visitGfxComponent(this) v->visitTcGfxObject(this); New visitors could implement visitTcGfxObject as follows:

void myNewVisitor::visitGfxObject(TcGfxObject *o) visitGfxComponent(o);

MyClassObject *obj = dynamic_cast<MyClassObject*>(o); if(obj) <MyClassObjectspecificvisitorcodegoeshere> else <TcGfxObjectgenericvisitorcodegoeshere>

32.3

virtual void visitGfxGroup (TcGfxGroup *g)

Common base class TcGfxGroup method

Common base class TcGfxGroup method. See TcGfxObject for an example.
class TdGfxGLGroup : public TcGfxGroup

This class is used to setup a tree of GL components

B.0.0.0.62 Inheritance

B.0.0.0.63 Public Members

TdGfxGLGroup () Constructor

TdGfxGLGroup (const TcUtString &id) Constructor .................. 195

virtual ~TdGfxGLGroup () Destructor.

virtual int addChild (TcGfxComponent *gfxNode) Adds a TcGfxComponent GL object.

virtual int removeChild (TcGfxComponent *gfxNode)
Appendix B (Continued)

Removes a TcGfxComponent GL object.

virtual void hideGeometry () // Hides the geometry of a group of GL objects.
virtual void unHideGeometry () // Unhides the geometry of a group of GL objects.
virtual void loadGeometry (const LoadType& ldType) // Loads the geometry of a group of GL objects.
virtual void unLoadGeometry () // Unloads the geometry of a group of GL objects.
virtual void draw () // Draw a group of GL objects.
virtual void update () // Update a group of GL objects.
virtual TcUtVector3f getPosition () // Return the position of GL objects.
virtual int setPosition (TcUtVector3f) // Set the position of GL objects.
virtual TcUtVector3f getOrientation () // Get the orientation of GL objects.
virtual int setOrientation (TcUtVector3f) // Set the orientation of GL objects.
virtual void accept (TcGfxVisitor &v) // Used by the TcGfxVisitor mechanism.

This class is used to setup a tree of GL components. It is similar to the role of TdGfxPfGroup for Performer components.

33.1

TdGfxGLGroup (const TcUtString &id)
Appendix B (Continued)

Constructor

Parameters: id — Sets the compName
class TdGfxPfDCSAdapter : public TdGfxPfGroupAdapter

pfDCS adapter class

B.0.0.0.64 Inheritance

TcGfxDspLibComponent

TcGfxPfComponentAdapter

TdGfxPfGroupAdapter

TdGfxPfDCSAdapter

B.0.0.0.65 Public Members

34.1 TdGfxPfDCSAdapter ()
   Default Constructor

34.2 TdGfxPfDCSAdapter (float matrix[])

[end of document]
Appendix B (Continued)

Constructor ......................... 198

34.3 TdGfxPfDCSAdapter (const TcUtVector3f &orientation,
                const TcUtVector3f &translation)
Constructor ......................... 199

34.4 TdGfxPfDCSAdapter (const TcUtVector3f &scale,
                const TcUtVector3f &orientation,
                const TcUtVector3f &translation)
Constructor ......................... 199

34.5 TdGfxPfDCSAdapter (pfDCS *dcs)
Constructor ......................... 200

34.6 TdGfxPfDCSAdapter (const TcUtString &compID)
Constructor ......................... 200

virtual ~TdGfxPfDCSAdapter ()
  Destructor.

void setRot (const float &x, const float &y, const float &z)
  Sets rotation in x, y, z float degree angles.

void setRot (const TcUtVector3f &)
  Sets the rotation in x, y, z float degree angles.

void setTrans (const float &, const float &, const float &)
  Sets the absolute translation in float x, y,
  z values

void setTrans (const TcUtVector3f &)
  Set the absolute translation in float x, y,
  z values

const pfDCS*
get_perfDCS () const
  Returns the pfDCS

pfDCS adapter class. This wrapper class constructs a pfDCS using TcUtVector3f objects. An
instance of this class is stored in the base TcGfxComponent as a TcGfxDspLibComponent. It is usually passed as an argument during the construction of a TdGfxPfGroup.

**34.1**

TdGfxPfDCSA dapter ()

*Default Constructor*

Default Constructor. scale = (1,1,1) rotation = (0,0,0) translation = (0,0,0)

**34.2**

TdGfxPfDCSA dapter (float matrix[])

*Constructor*

**Parameters:**

matrix — A matrix of 16 floats in row order a00..a03, a10..a13, a20..a23, a30, a33
34.3

\textbf{TdGfxPfDCSAdapter} ( const TcUtVector3f &orientation, const TcUtVector3f &translation)

Constructor

Constructor. These transformations are performed in the following order:

\textbf{Parameters:}

- orientation — Specified as float degree angles x,y,z NOT head,pitch,roll

- translation — Specified as x,y,z float values.

34.4

\textbf{TdGfxPfDCSAdapter} ( const TcUtVector3f &scale, const TcUtVector3f &orientation, const TcUtVector3f &translation)

Constructor

Constructor. These transformations are performed in the following order:
Appendix B (Continued)

Parameters:

scale — Specified as x,y,z and hence can be a non-uniform scale.

orientation — Specified as float degree angles x,y,z NOT head,pitch,roll

translation — Specified as x,y,z float values.

34.5

TdGfxPfDCSAdapter (pfDCS *dcs)

Constructor

Parameters: dcs — A pre-instantiated pfDCS.

34.6

TdGfxPfDCSAdapter (const TCHARString &compID)

Constructor

Constructor. scale = (1,1,1) rotation = (0,0,0) translation = (0,0,0) This saves having to make the setCompName(..) call.

Parameters: compID — a string name that the pfNode will be set to.
class TdGfxPfGrabController : public TcGfxVisitor

**Performing grab controller class**

### Inheritance

```
TdGfxVisitor
     V
TdGfxPfGrabController
```

### Public Members

- **Default constructor**
  ```cpp
  TdGfxPfGrabController ()
  ```

- **Destructor**
  ```cpp
  virtual ~TdGfxPfGrabController ()
  ```

- **Visit GfxComponent**
  ```cpp
  virtual void visitGfxComponent (TcGfxComponent *c)
  ```

- **Visit GfxObject**
  ```cpp
  virtual void visitGfxObject (TcGfxObject *o)
  ```

- **Visit GfxGroup**
  ```cpp
  virtual void visitGfxGroup (TcGfxGroup *g)
  ```

- **Visit GfxPfObject**
  ```cpp
  void visitGfxPfObject (TdGfxPfObject *o)
  ```
Appendix B (Continued)

void visitGfxPfGroup (TdGfxPfGroup *g)

    TdGfxPfObject visit method.

void visitGfxGLObject (TcGfxGLObject *o)

    TcGfxGLObject visit method.

void visitGfxGLGroup (TdGfxGLGroup *g)

    TdGfxGLGroup visit method.

35.1 void reset (const TcUtfVector3f &isectLocation)

    This method must be called with the intersection point of the grab ............... 203

void initArrow (TcGfxComponent &navGrp)

    This must be called once at instantiation.

void hideArrow ()

    This is used to hide the grab intersection arrow.

void setIsectLocation (const TcUtfVector3f &loc)

    This is used to set the position the arrow is to appear.

Performer grab controller class. This is a development interaction class. A user can grab an object and transform it in a virtual environment. This can only be done on TdGfxPfDCSAdapter classes. Error messages are shown for all other class types. This saves clients from having to switch on class types to achieve the same level of run-time type safety. First reset() must be called. Then an instance of this class must be sent as argument to a TcGfxComponent instance accept() method (the object to be grabbed). Objects in World Coordinates must be a TdGfxPfGroup containing a DCSAdapter. Objects in Local Coordinates must be TdGfxPfObject containing a DCSAdapter.
This method must be called with the intersection point of the grab

**Parameters:**

- `isectLocation` — the location the object is grabbed from in World coordinates.
Appendix B (Continued)

36

\[
\text{class } \text{TdGfxPfGroup} : \text{public TcGfxGroup}
\]

\textit{Class TdGfxPfGroup}

B.0.0.0.68 \underline{Inheritance}

20

\text{TcGfxComponent}

24

\text{TcGfxGroup}

36

\text{TdGfxPfGroup}

B.0.0.0.69 \underline{Public Members}

36.1 \textbf{TdGfxPfGroup} (\text{const TcUtString \&name,} \\
\text{const LoadState \&initialLoadState,} \\
\text{TcGfxPfComponentAdapter \*dspLibComp})

\textit{Constructor} .......................... 206

36.2 \textbf{virtual} \textbf{~TdGfxPfGroup} () \textit{Destructor} .......................... 207

36.3 \textbf{virtual} \textbf{void} \textbf{unLoadGeometry} () \textit{This function unloads all children of this} \\
\textit{node, removing them from memory} ... 207
Appendix B (Continued)

36.4 virtual void **loadGeometry** (const LoadType &ldType = ASYNC)
   *This function loads all children of this node.*

36.5 virtual int **addChild** (TcGfxComponent *gfxNode)
   *Add a TcGfxComponent to the child list.*

36.6 virtual int **removeChild** (TcGfxComponent *gfxNode)
   *Remove a TcGfxComponent from the child list.*

36.7 virtual void **accept** (TcGfxVisitor &v)
   *Visitor pattern method.*

Class TdGfxPfGroup. This class is used to encapsulate groups of performer graphics components. It provides recursive operations that traverse its children. Its TcGfxDspLibComponent is generally an instances of TdGfxPfGroupAdapter or any of its descendants.

**Role:** This is a Development Graphics Group. Clients can use this class as is or inherit from it and extend its functionality. The grouping in this class specifies the load hierarchy. The load hierarchy can be used to partition the environment (swap objects in and out of performer memory) or define priorities (load surrounding objects based on proximity). The view hierarchy specifies the structure of the scene graph.
Appenidx B (Continued)

Responsibilities: This class must implement the load and unload methods that operate on a list of children.

36.1

```
TdGfxPfGroup (const TcUtString &name, const LoadState &initialLoadState, TcGfxPfComponentAdapter *dspLibComp)
```

*Constructor*

Constructor.

**Parameters:**

- `name` — This name defines the name used for searches on the TcGfxComponent tree. It must be unique in the entire tree to which it is to be added.

- `initialLoadState` — Groups to be asynchronously loaded should be set to UNLOADED and loadGeometry(ASYNC) call should be made after all the children have been added to the group.

36.2

```
virtual ~TdGfxPfGroup ()
```

*Destructor*
Destructor. This will take care of deleting all its children.

36.3

virtual void unLoadGeometry()

*This function unloads all children of this node, removing them from memory*

This function unloads all children of this node, removing them from memory. Performer’s caching may sabotage certain unload efforts.

36.4

virtual void loadGeometry (const LoadType &ldType = ASYNC)

*This function loads all children of this node*

**Parameters:**

ldType — An argument of SYNC will perform a synchronous load causing the program to stop responding in the interim. An argument of ASYNC will perform an asynchronous load of all children.
Appendix B (Continued)

36.5

virtual int addChild (TcGfxComponent *gfxNode)

Add a TcGfxComponent to the child list

Parameters: gfxNode — If this node is loaded its pfNode will be added to the scene by attaching it to the underlying pfGroup.

36.6

virtual int removeChild (TcGfxComponent *gfxNode)

Remove a TcGfxComponent from the child list

Remove a TcGfxComponent from the child list. This method must be called to remove a child from a parent before the child is deleted.

36.7

virtual void accept (TcGfxVisitor &v)

Visitor pattern method
Visitor pattern method. This will call `v->visitGfxPfGroup` to any visitor resulting in type laundering and implements a special case of double dispatching.
class TdGfxPfGroupAdapter : public TcGfxPfComponentAdapter

pfGroup adapter class

B.0.0.0.70 Inheritance

B.0.0.0.71 Public Members

37.1 TdGfxPfGroupAdapter (const bool &instantiate = true)
37.2 TdGfxPfGroupAdapter (pfGroup *group)
(Constructor) .............................. 212

virtual ~TdGfxPfGroupAdapter ()
(Destructor).

void removeChild (TcGfxPfComponentAdapter &childNode)

Removes the pfNode contained in the
childNode from the pfGroup.

void addChild (const TcGfxPfComponentAdapter &childNode)

Adds the pfNode contained in the
childNode to the pfGroup.

int completeAsyncLoad (const TcUtString &,

TcGfxPfComponentAdapter &child)

Used by TcGfxComponent, after an
async load is completed.

virtual bool isActive ()

Returns true;

B.0.0.0.72  Protected Members

37.3 TdGfxPfGroupAdapter (const TcUtString &compID)
(Constructor) .............................. 212

pfGroup adapter class. This wrapper class constructs a pfGroup.
Appendix B (Continued)

**Parameters:**

instantiate — Determines whether a pfGroup is instantiated or not. By default a pfGroup is instantiated. Derived classes must set this to false to prevent instantiating two pfGroup nodes (or derivations from pfGroup).

<table>
<thead>
<tr>
<th>37.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TdGfxPfGroupAdapter</strong> *(pfGroup <em>group)</em></td>
</tr>
</tbody>
</table>

*Constructor*

**Parameters:**

group — Allows clients to pre-instantiate a pfGroup.

<table>
<thead>
<tr>
<th>37.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TdGfxPfGroupAdapter</strong> <em>(const TcUtString &amp;compID)</em></td>
</tr>
</tbody>
</table>

*Constructor*
Appendix B (Continued)

Parameters:  

compID — Used by derived classes to set the pfNode name.
Appendix B (Continued)

```cpp
class TdGfxPfIntersectionTest : public TcGfxIntersectionTest
```

**Performer Cavelib Intersection Test**

B.0.0.0.73 Inheritance

B.0.0.0.74 Public Members

38.1 enum Sensor

Sensor enum .........................
Appendix B (Continued)

38.2  \texttt{TdGfxPfIntersectionTest} (const TcUtString \&isectName,
const

\texttt{TdGfxPfIntersectionTest::Sensor}

sensor,

TcGfxComponent \&rootNode,
const unsigned int \&isectMask,
const unsigned int \&activeMask

= 0x01, const int \&modeMask =

PFTRAV(JS\_PATH —

PFTRAV(JS\_GSET —

PFTRAV(JS\_CULL\_BACK)

\textit{Constructor} ............................... 218

\texttt{\~ TdGfxPfIntersectionTest} ()

\textit{Destructor.}

38.3  \texttt{void setActiveSegments} (const int \&numSegs = 1)

\textit{This sets the number of active segments as consecutive bits in the active mask .} 219

\texttt{void setIsectMask} (const unsigned int \&isectMask)

\textit{Sets the mask to search for in the graphics hierarchy.}

\texttt{void setMode} (const int \&modeMask)

\textit{Determines the behavior of the intersection test.}

38.4  \texttt{void setRayLength} (const float \&rayLength,
const int \&segNum = 0)
Appendix B (Continued)

38.5 virtual IsectState

   doTcIntersectionTest ()

   updates the intersection test based on its
current state .......................... 220

38.6 const TcUTVector3f&

   getIsectNormal (const int &segNum = 0)

   returns the normal of the last successful
intersection .......................... 221

38.7 const TcUTVector3f&

   getLastIsectPoint (const CoordType &type = LOCAL,
   const int &segNum = 0)

   returns the position of the last successful
intersection .......................... 221

38.8 void setRayStartPostionFromWand (const int &segNum = 0)

   Set the start position for a given segment
number from the wand sensor .......... 222

38.9 void setRayDirectionFromWand (const int &segNum = 0)

   Set the direction for a given segment
number from the wand sensor .......... 222

38.10 void setRayStartPostionFromHead (const int &segNum = 0)

   Set the start position for a given segment
from the head sensor .................. 223

38.11 void setRayDirectionFromHead (const int &segNum = 0)

   Set the start direction for a given segment
from the head sensor .................. 223

38.12 void setRayStartPosition (const TcUTVector3f &vector,

   const CoordType &type = WORLD,
   const int &segNum = 0)
Appendix B (Continued)

Set the start position for the segment number from the given vector position

38.13  void  setRayDirection (const TcUtVector3f &vector,

const CoordType &type = WORLD,

const int &segNum = 0)

Set the start direction for the segment number from the given vector orientation

setCurSeg (const int &num)

Sets the segment number do TcIntersectionTest will operate on

Performer Cavelib Intersection Test. This class provides an interface to Performer’s pfSegSet. It can be automatically updated by passing it to the intersection manager, or the client can chose to update the intersection.

38.1

enum Sensor

Sensor enum
Appendix B (Continued)

Parameters:

**HEAD** — Set the intersection test from the head sensor.

**WAND** — Set the intersection test from the wand sensor.

**NONE** — Manually set the intersection position and direction.

```
38.2

TdGfxPfIntersectionTest (const TcUtString &isectName, const
TdGfxPfIntersectionTest::Sensor sensor, TcGfx-
Component &rootNode, const unsigned int
&isectMask, const unsigned int &active-
Mask = 0x01, const int &modeMask = PF-
TRAV_JS_PATH — PFTRAV_JS_GSET —
PFTRAV_JS_CULL_BACK)
```

*Constructor*
Appendix B (Continued)

**Parameters:**

- **isectName** — This string is used to retrieve an intersection test from the intersection manager.

- **sensor** — Determines whether doTeIntersection automatically updates the intersection test from a sensor position and direction.

- **rootNode** — Sets the subtree in the graphics hierarchy that the intersection is tested from.

- **isectMask** — Sets a bit field mask to test for.

- **activeMask** — Sets the initial number of segments that are to be tested. This is a bitfield that determines which intersection segments are active.

- **modeMask** — This is a Performer mask that determines the behavior of the intersection test. See pfSegSet.

```c
void setActiveSegments (const int &numSegs = 1)
```

*This sets the number of active segments as consecutive bits in the active mask*

This sets the number of active segments as consecutive bits in the active mask. For instance 5 active segments would cause the first 5 bits to be set to 1 in the activeMask.
Appendix B (Continued)

Parameters: numSegs — Determines the number of consecutive bits that are active.

38.4

```c
void setRayLength (const float &rayLength, const int &segNum = 0)
```

Sets the length of a segment

Parameters: rayLength — A float value determining ray length.

segNum — Determines the segment to apply this length to. This must be less than the number of active segments.

38.5

```c
virtual IsectState doTcIntersectionTest ()
```

updates the intersection test based on its current state

updates the intersection test based on its current state. Sensor (HEAD, WAND) based intersection tests automatically update intersection start position and direction. This method is called once per iteration if the intersection test is managed by the intersection manager.
Appendix B (Continued)

Parameters: 

Returns — ISECTFOUND, ISECTNOTFOUND

const TcUtVector3f& getIsectNormal (const int &segNum = 0)

returns the normal of the last successful intersection

returns the normal of the last successful intersection. This value is not cleared after an update in which no intersection is found.

Parameters: type — Returns the normal for a given line segment.

const TcUtVector3f& getLastIsectPoint (const CoordType &type = LOWCal, const int &segNum = 0)

returns the position of the last successful intersection

returns the position of the last successful intersection. This value is not cleared after an update in which no intersection is found.
Appendix B (Continued)

**Parameters:**

- **type** — Returns the point in local or world coordinates.

- **segNum** — Returns the point for the given line segment. This is a number starting from 0 and must be within the range of the active segments.

**38.8**

```c
void setRayStartPositionFromWand (const int &segNum = 0)
```

*Set the start position for a given segment number from the wand sensor*

**Parameters:**

- **segNum** — This is a number starting from 0 and must be within the range of the active segments.

**38.9**

```c
void setRayDirectionFromWand (const int &segNum = 0)
```

*Set the direction for a given segment number from the wand sensor*
Appendix B (Continued)

Parameters: \texttt{segNum} — This is a number starting from 0 and must be within the range of the active segments.

38.10

\begin{verbatim}
void setRayStartPositionFromHead (const int &segNum = 0)
\end{verbatim}

\textit{Set the start position for a given segment from the head sensor}

Parameters: \texttt{segNum} — This is a number starting from 0 and must be within the range of the active segments.

38.11

\begin{verbatim}
void setRayDirectionFromHead (const int &segNum = 0)
\end{verbatim}

\textit{Set the start direction for a given segment from the head sensor}

Parameters: \texttt{segNum} — This is a number starting from 0 and must be within the range of the active segments.
Appendix B (Continued)

38.12

```c
void setRayStartPosition (const TcUtVector3f &vector, const CoordType &type = WORLD, const int &segNum = 0)
```

*Set the start position for the segment number from the given vector position*

**Parameters:**
- `vector` — a positional vector.
- `type` — Currently if type is WORLD it will be converted to the viewpoint (local) coord system (CAVEConvertWorldToNav). This assumes the intersection starts from pfScene node.
- `segNum` — This is a number starting from 0 and must be within the range of the active segments.

38.13

```c
void setRayDirection (const TcUtVector3f &vector, const CoordType &type = WORLD, const int &segNum = 0)
```

*Set the start direction for the segment number from the given vector orientation*
Appendix B (Continued)

Parameters:

- **vector** — a directional vector.
- **type** — Currently if type is WORLD it will be converted to the viewpoint (local) coord system (CAVEConvertVectorWorld-ToNav). This assumes the intersection starts from pfScene node.
- **segNum** — This is a number starting from 0 and must be within the range of the active segments.

```c
38.14

void setCurSeg (const int &num)

Sets the segment number doTcIntersectionTest will operate on

Sets the segment number doTcIntersectionTest will operate on. This must be greater than 0 and less than the number of active segments.
Appendix B (Continued)

```cpp
class TdGfxPfNodeAdapter : public TcGfxPfComponentAdapter
```

**Class TdGfxPfNodeAdapter**

### B.0.0.0.75 Inheritance

![Inheritance Diagram]

**B.0.0.0.76 Public Members**

39.1 TdGfxPfNodeAdapter (const TcUtfString &compId)  
*Constructor*  
……………………………… 227

39.2 TdGfxPfNodeAdapter (pfNode *nd)  
*Constructor*  
……………………………… 227

virtual TdGfxPfNodeAdapter ()  
*Destructor.*

39.3 int loadFile (const TcUtfString &fileName)
Class TdGfxPfNodeAdapter. This class is a generic wrapper for pfNode. It is generally used to hold pfNode’s returned from pfdLoad(). It can also be used to write custom performer classes that contain pfObjects such as pfGeodes.

### 39.1

**TdGfxPfNodeAdapter** (const TcUtString &compId)

*Constructor*

**Parameters:**

compId — Name stored in pfNode.

### 39.2

**TdGfxPfNodeAdapter** (pfNode *nd)

*Constructor*
Appendix B (Continued)

Parameters: nd — Pre-instantiated pfNode.

39.3

int loadFile (const TcUtString &fileName)

Loads a file. Results from loadGeometry() calls made to a TcGfxComponent.

Parameters: fileName — filename as it appears in PFPATH.
Appendix B (Continued)

```cpp
class TdGfxPfObject : public TcGfxObject, public TcGfxPfBackgroundLoader
    TcGfxLoadRequest TcGfxBgLoaderObserver
```

*Class TdGfxPfObject*

**Inheritance**

```
TdGfxPfObject

TcGfxBgLoaderObserver

TcGfxLoadRequest

TcGfxPfBackgroundLoader

TcGfxComponent

TcGfxObject

```
 Appendix B (Continued)

B.0.0.0.78 Public Members

40.1 virtual ~TdGfxPfObject () Destructor.

40.2 TdGfxPfObject (const TcUtString &name,
               const LoadState &ldState,
               const LoadType &ldType,
               TcUtString fname)

               Constructor  .................. 232

40.3 TdGfxPfObject (const TcUtString &name,
               const LoadState &ldState,
               const LoadType &ldType,
               TcUtString fname,
               const TdGfxPfTravAttributes &travAtts)

               Constructor  .................. 233

40.4 TdGfxPfObject (TcUtString name)

               Constructor  .................. 235

40.5 void loadFromSceneFile ()

               Read in a model description from a text

               file  .......................... 235

40.6 virtual void clientLoadGfxObject ()

               Allows client extension of the loadFrom-

               SceneFile  ...................... 236

40.7 virtual void loadGeometry (const LoadType &ldType = ASYNC)
Appendix B (Continued)

40.8 virtual void **unLoadGeometry** () 
*Unload geometry from the scene .......... 237*

virtual void **hideGeometry** () 
*Set the APP, DRAW, ISECT traversal masks to 0.*

virtual void **unHideGeometry** () 
*Restores the APP, DRAW, ISECT traversal masks to their value prior to being hidden*

40.9 virtual **TcUtVector3f**
**getPosition** () 
*Returns a vector with the current position of the object ......................... 237*

40.10 virtual int **setPosition** (TcUtVector3f) 
*Sets the position of the object .............. 237*

40.11 virtual **TcUtVector3f**
**getOrientation** () 
*Returns a vector with the current orientation of the object ..................... 238*

40.12 virtual int **setOrientation** (TcUtVector3f) 
*Sets the orientation of the object ...... 238*

40.13 virtual int **gfxLoadUpdate** (const TcGfxPfBackground-
Loader::TcGfxLoadRequest&) 

**TcGfxBgLoaderObserver method implementation ..................... 238**

virtual void **accept** (TcGfxVisitor &) 
*Visitor pattern method.*

B.0.0.0.79 **Protected Members**
Class TdGfxPfObject. It contains information about an object that can be drawn in the scene.

Role: This is the development Graphics class for Performer leaf nodes. Clients can use it as is or extend its functionality by deriving a custom class.

Responsibilities: This class implements methods that enable clients to physically load and unload the object out of performer memory (pfdLoad) as well as the ability to hide and unhide the object.
Appendix B (Continued)

Parameters:

name — String name given to TcGfxComponent and pfNode.

ldState — Determines whether an object is loaded at instantiation.

ldType — Determines whether an object is synchronously or asynchronously loaded

fname — Specifies the file name that pfdLoad will be sent.

40.2

TdGfxPfObject (const TcUtString &name, const LoadState &ldState, const
LoadType &ldType, TcUtString fname, const TdGfxPfTvAttributes & travAtts)

Constructor

Constructor. An object should not be instantiated with ldState == LOADED and ldType
== ASYNC since asynchronous loading requires the TdGfxPfObject to be attached to a parent.
This request will be ignored.
Appendix B (Continued)

**Parameters:**

- **name** — String name given to TcGfxComponent and pfNode.
- **ldState** — Determines whether an object is loaded at instantiation.
- **ldType** — Determines whether an object is synchronously or asynchronously loaded.
- **fname** — Specifies the file name that pfdLoad will be sent.
- **travAtts** — Specifies traversal attributes for one of DRAW, ISECT or CULL. Additional masks can be set by a setTravMask call on the pfComponentAdapter.

```cpp
TdGfxPfObject (const TcUtString &name, const LoadState &ldState,
                TdGfxPfNodeAdapter *ndAdapter)

Constructor
```
Appendix B (Continued)

Parameters:

- **name** — String name given to TcGfxComponent and pfNode.
- **ldState** — Determines whether an object is loaded at instantiation.
- **ndAdapter** — A pre-instantiated pfNodeAdapter that implements loadFile()

### 40.4

**TdGfxPfObject** *(TcUtString name)*

*Constructor*

Constructor. This must have its TcGfxPfComponentAdapter explicitly set.

Parameters:

- **name** — String name given to TcGfxComponent and pfNode.

### 40.5

**void loadFromSceneFile ()**

*Read in a model description from a text file*

Read in a model description from a text file. Not yet implemented.
Appendix B (Continued)

40.6

virtual void clientLoadGfxObject ()

Allows client extension of the loadFromSceneFile

Allows client extension of the loadFromSceneFile. Not yet implemented.

40.7

virtual void loadGeometry (const LoadType& ldType = ASYNC)

Load geometry into the scene

Load geometry into the scene. This method will invoke the performer loader either in the main process or via the DBASE process.

Parameters: ldType — Determines whether a synchronous or asynchronous (using DBASE process) is performed.

40.8

virtual void unLoadGeometry ()

Unload geometry from the scene
Appendix B (Continued)

Unload geometry from the scene. This method will remove the pfNode from the parent and pfDelete it, removing it from memory. An error will be displayed to the console if pfDelete is not successful.

40.9

virtual TcUtVector3f getPosition ()

_Returns a vector with the current position of the object_

Returns a vector with the current position of the object. Not yet implemented.

40.10

virtual int setPosition (TcUtVector3f)

_Sets the position of the object_

Sets the position of the object. Not yet implemented.
virtual TcUtVector3f getOrientation ()

Returns a vector with the current orientation of the object

Returns a vector with the current orientation of the object. Not yet implemented.

virtual int setOrientation (TcUtVector3f)

Sets the orientation of the object

Sets the orientation of the object. Not yet implemented.

virtual int gfxLoadUpdate (const TcGfxPfBackgroundColor::TcGfxLoadRequest& request)

TcGfxBgLoaderObserver method implementation

TcGfxBgLoaderObserver method implementation. This method is invoked when a load request is completed.
Appendix B (Continued)

40.14

TdGfxPfObject (const TcUtString &name, const LoadState &ldState,

TcGfxPfComponentAdapter *ndAdapter)

Constructor

Constructor. This constructor enables extension to the TdGfxPfObject class. It is useful when you want to add TdGfxPfObjects that have differing load and unload behavior than a typical TdGfxPfObject which is loaded from a pfLoad() call and expects the geoFileName member to be set. The client is responsible for attaching and unattaching the pfNode to the parent as a result of calls to loadGeometry or unLoadGeometry.

Parameters:

name — String name given to TcGfxComponent and pfNode.

ldState — Determines whether an object is loaded at instantiation.

ndAdapter — A pre-instantiated pfNodeAdapter that implements loadFile()
class TdGfxPfSCSAdapter : public TdGfxPfGroupAdapter

pfSCS adapter class

B.0.0.0.80 Inheritance

TcGfxDspLibComponent

TcGfxPfComponentAdapter

TdGfxPfGroupAdapter

TdGfxPfSCSAdapter

B.0.0.0.81 Public Members

41.1 TdGfxPfSCSAdapter (const TcUti::Vector3f &orientation,
const TcUti::Vector3f &translation)
pfSCS adapter class. This wrapper class constructs a pfSCS using TcUtVector3f objects. An instance of this class is stored in the base TcGfxComponent as a TcGfxDispLibComponent. It is usually passed as an argument during the construction of a TdGfxPfGroup.

**Constructor**

These transformations are performed in the following order:

**Parameters:**

- **orientation** — Specified as float angles x,y,z NOT head,pitch,roll
- **translation** — Specified as x,y,z float values.
Appendix B (Continued)

Constructor. These transformations are performed in the following order:

**Parameters:**

- **scale** — Specified as x,y,z and hence can be a non-uniform scale.
- **orientation** — Specified as float angles x,y,z NOT head,pitch,roll
- **translation** — Specified as x,y,z float values.
class TdGfxPfTravAttributes

*Performe traversal attributes class*

B.0.0.0.82 Public Members

**TdGfxPfTravAttributes ()**

*Constructor.*

**TdGfxPfTravAttributes (const int &maskType,**

const unsigned int& mask,

const int &mode,

const int &bitop)***

*Constructor    .................. 244*

**~TdGfxPfTravAttributes ()**

*Destructor.*

const TdGfxPfTravAttributes&

*operator* = (const TdGfxPfTravAttributes &right)*

*Assignement operator.*

const int&

getMaskType () const

*Returns the type of mask. See the constructor.*

void

setMaskType (const int &value)*

*Sets the type of mask.*

const unsigned int&

getMask () const

*Gets the value of the mask.*

void

setMask (unsigned const int &value)*

*Sets the value of the mask.*

const int&

getMode () const

*Returns the mode of the mask.*

void

setMode (const int &value)
Appendix B (Continued)

Sets the mode of the mask.

```cpp
const int& getBitop () const
Returns the bit operation to be performed.
```

```cpp
void setBitop (const int &value)
Sets the bit operation to be performed.
```

Performer traversal attributes class. This class is used to set performer traversal properties for
the APP, DRAW, ISECT traversals. It is generally used to set the TcGfxPfComponentAdapter
classes or as an argument to the constructor of a TdGfxPfObject.

```cpp
42.1

TdGfxPfTravAttributes (const int &maskType, const unsigned int &mask,
const int &mode, const int &bitop)
```

*Constructor*

**Parameters:**

- **maskType** — Determines which mask to set: either ISECT — APP
  — DRAW

- **mask** — This is the value of the mask itself, interpreted as a bit-field.

- **mode** — This determines how the mask is set and whether to pro-
pagate to its children.

- **bitOp** — either PF\_OR, PF\_SET or PF\_AND
Appendix B (Continued)

Class \texttt{TcNetCavConnectionManager} : public \texttt{TcNetConnectionManager}

\textit{Role: This class plays the role of TcNetEvent shipper and receiver}

B.0.0.0.83 Inheritance

B.0.0.0.84 Public Members
TcNetCavConnectionManager (int argc, char* argv[],

const long&

cclientLocalPort = 7000,

cconst long&

cclientRemotePort = 7001,

cconst bool&

cclientRemoteConnect =

cfalse, const TcUtString&

cclientUserName =

cTcUtString("Demo"),

cconst TcUtString&

cclientWorldServer =

cTcUtString("localhost"),

cconst TcUtString&

cclientWorldName =

cTcUtString("Demo"))

Constructor

virtual ~TcNetCavConnectionManager ()

Destructor.
Appendix B (Continued)

**createConnection** (const TcUtfString &keyName, const TdNetCavConnection::connectionType &type)

> Creates a Cavern Connection

---

void **addConnection** (TdNetCavConnection *connection)

> Add a connection to the connection table.

---

TdNetCavConnection*

**getConnection** (const TcUtfString &name)

> Retrieve a connection from the connection table.

---

void **deleteConnection** (const TcUtfString &name)

> Remove a connection from the connection table.

---

void **addKey** (TdNetCavKey *)

> Add a key to the key table.

---

TdNetCavKey*

**lookUpCavKey** (const TcUtfString &name)

> Returns a key with the given name.

---

void **closeLink** ()

> Close a link.

---

virtual int **config** (long &clientLocalPort, long &clientRemotePort,

> TcUtfString &clientUserName,

> TcUtfString &clientWorldServer,

> TcUtfString &clientWorldName)

> Must be called to initialize the class.

const TcUtfString &

**getWorldServer** () const

> Return the server name.

const TcUtfString &
Appendix B (Continued)

```cpp
getWorldName() const
    Return the world name.

const TcUtString&
getUserName() const
    Return the user name.

static const CAVERN_irb* 
getCavernIrb()  Return the CAVERN irb.

const long& getLocalPort() const
    Return the local port.

const long& getRemotePort() const
    Return the remote port.

const bool& getRemoteConnect() const
    Return the value of the remoteConnect flag.
```

Role: This class plays the role of TcNetEvent shipper and receiver. It adapts the CAVERNsoft API to Tandem.

```cpp
TcNetCavConnectionManager (int argc, char* argv[]), const long& clientLocalPort = 7000, const long& clientRemotePort = 7001, const bool& clientRemoteConnect = false, const TcUtString& clientUserName = TcUtString("Demo"), const TcUtString& clientWorldServer= TcUtString("localhost"),
const TcUtString& clientWorldName = TcUtString("Demo"))
```
Appendix B (Continued)

Constructor

**Parameters:**

- `argc` — argument from main()
- `argv` — argument from main()
- `clientLocalPort` — port number of the client
- `clientRemotePort` — port number of the server
- `clientRemoteConnect` — determines whether connection to server is needed.
- `clientUserName` — name requested for connection to server.
- `clientWorldServer` — url of the server to connect to.
- `clientWorldName` — name of the world to connect to on the server.

```cpp
virtual TdNetCavConnection* createConnection (const TcUtString &keyName, const TdNetCavConnection::connectionType &type)
```

*Creates a Cavern Connection*
Appendix B (Continued)

Parameters:

keyName — The name of the Connection. Used to retrieve from the connection table.

type — Either UDP or TCP.
Appendix B (Continued)

class TcNetCavEvent : public TcNetEvent

CAVERNsoft Network Event

B.0.0.0.85 Inheritance

B.0.0.0.86 Public Members

virtual ~TcNetCavEvent() destructor.

virtual int sendToCavernKey (CAVERN.irbKey.c &key)
Description: This virtual functions is implemented by derived classes

virtual int getFromCavernKey (CAVERN.irbKey.c &key)
Description: This virtual functions is implemented by derived classes

const TcUtString &
getDestination() Returns the destination key.

void setDestination(const TcUtString &)
Sets the destination key.

const TcUtString &
Appendix B (Continued)

getOrigin () const  
\textit{Used to get the sender.}

void setOrigin (const TcUtString &c)  
\textit{Used to set the sender.}

\textbf{B.0.0.0.87  Protected Members}

\begin{itemize}
    \item \texttt{TcNetCavEvent ()}  
        \textit{Constructor.}
    \item \texttt{CAVERN\_irbKey\_\_\_: status t &}  
        \texttt{getKeyStatus ()}  
        \textit{Returns the CAVERNsoft key status after a send or receive.}
    \item \texttt{char * eventBuffer}  
        \textit{Used for event data.}
    \item \texttt{char * metaBuffer}  
        \textit{Used for event meta data.}
    \item \texttt{CAVERN\_irbKey\_\_\_: status t}  
        \texttt{keyStatus}  
        \textit{Used for the key status.}
    \item \texttt{TcUtString destination}  
        \textit{Used to set the destination key.}
    \item \texttt{TcUtString origin}  
        \textit{Used to set the sender.}
\end{itemize}

CAVERNsoft Network Event. This class provides serialization via CAVERNsoft.

\begin{itemize}
    \item virtual int \texttt{sendToCavernKey (CAVERN\_irbKey\_\_c &key)}
\end{itemize}

\textit{Description: This virtual functions is implemented by derived classes}

Description: This virtual functions is implemented by derived classes. The intent is such that each class will implement its own CAVERN packing scheme and put the data onto the key passed to it.
Appendix B (Continued)

44.2

virtual int getFromCavernKey (CAVERN irbKey c &key)

Description: This virtual functions is implemented by derived classes.

Description: This virtual functions is implemented by derived classes. The intent is such that each class will implement its own CAVERN unpacking scheme and get the data from the key passed to it.
class TcNetConnectionManager

Abstract Base Class for Network Connection Manager

Inheritance

TcNetConnectionManager

TcNetCavConnectionManager

Public Members

virtual ~TcNetConnectionManager ()

Destructor.

int tdSendNetworkEvent (TcNetEvent& event)

Overidden by clients to determine how a network event is sent.

bool isConfigured () const

Determines if this class has been initialized.

virtual int config (long& clientLocalPort, long& clientRemotePort,

TcUtString& clientUserName,

TcUtString& clientWorldServer,

TcUtString& clientWorldName)
Appendix B (Continued)

Used to initialize the class.

B.0.0.0.90  Protected Members

TcNetConnectionManager ()

Constructor.

bool configureFlag  Used to set the configuration state.
Appendix B (Continued)

class TcNetEvent

Abstract Base Class for Network Events.

B.0.0.0.91 Inheritance

46

TcNetEvent

44

TcNetCavEvent

B.0.0.0.92 Public Members

TcNetEvent () constructor.
virtual ~TcNetEvent () destructor.
virtual void resolveOutgoingEvent () Used to store mapping information for outgoing events.

virtual void resolveIncomingEvent () Used to store mapping information for incoming events.
class TcNetEventManager

Abstract base class for Event Manager

B.0.0.0.93 Public Members

  TcNetEventManager ()
  Constructor.

  virtual ~TcNetEventManager ()
  Destructor.

  virtual void handleNetworkEvents ()
  Must be overridden by clients to handle
  received network events.
class TcNetManager

Abstract base class for Network Manager

B.0.0.0.94 Public Members

48.1 TcNetManager (int argc, char* argv[],
long& clientLocalPort,
long& clientRemotePort,
bool& clientRemoteConnect,
TcUtString& clientUserName,
TcUtString& clientWorldServer,
TcUtString& clientWorldName)

Constructor ........................................ 260
virtual ~TcNetManager () Destructor.
virtual TcNetConnectionManager*
createNetConnectionManager ()
Factory Method
virtual TcNetEventManager*
createNetEventManager ()
Factory Method
int config () Must be called prior to using the factory
method products.
bool isConfigured () const Determines if config has been called.
void handleNetworkEvents ()
Appendix B (Continued)

Called from the Tandem Execution Control loop once per iteration.

const long& getLocalPort () Returns the client port.
TcUtString getLocalHost () Returns the client host name.

B.0.0.0.95 Protected Members

const TcNetEventManager* getEventManager () Returns the Event Manager
const TcNetConnectionManager* getConnectionManager () Returns the Connection Manager

int localArgC argc argument from main()
char** localArgV argv argument from main()
long localPort port number of the client
long remotePort port number of the server
bool remoteConnect determines whether connection to server is needed.

TcUtString userName name requested for connection to server
TcUtString worldServer url of the server to connect to.
TcUtString worldName name of the world to connect to on the server.
Appendix B (Continued)

48.1

**TcNetManager** \((\text{int } \text{argc}, \text{char* argv}[], \text{long& clientLocalPort, long& clientRemotePort, bool& clientRemoteConnect, TcUtString& clientUserName, TcUtString& clientWorldServer, TcUtString& clientWorldName})\)

*Constructor*

**Parameters:**

- **argc** — argument from main()
- **argv** — argument from main()
- **clientLocalPort** — port number of the client
- **clientRemotePort** — port number of the server
- **clientRemoteConnect** — determines whether connection to server is needed.
- **clientUserName** — name requested for connection to server.
- **clientWorldServer** — url of the server to connect to.
- **clientWorldName** — name of the world to connect to on the server.
Appendix B (Continued)

class TdNetCavConnection

\textit{Class used to encapsulate a CAVERNsoft Connection}

\textbf{B.0.0.0.96 Public Members}

49.1 \textbf{enum} \quad \textit{connectionType} \quad \textit{Connection type enumeration} \quad 262

49.2 \textbf{TdNetCavConnection} (const TcUtString &connectID,

\begin{verbatim}
const TcUtString &worldServer,
const long &remotePort, TdNetCav-
Connection::connectionType type =
TCP)
\end{verbatim}

\textit{Constructor} \quad 263

\texttt{~TdNetCavConnection ()}

\textit{Destructor}.

49.3 \textbf{void} \quad \textbf{addLink} (TdNetCavKey & key)

\textit{Link this key to key on remote server} \quad 263

\textbf{void} \quad \textbf{removeLink} (TdNetCavKey & key)

\textit{Removes a link}.

\textbf{void} \quad \textbf{setCavLinkAttribute} (CAVERN\_linkAttrib & att)

\textit{Sets the attributes for the link}.

\textbf{void} \quad \textbf{setCB} (void (*call-
back))(CAVERN\_irbChannel &::CAVERN\_channelEvent &

event, CAVERN\_irbChannel *thisChannel,

void *userData), void *userData)

\textit{Sets the callback for the channel}.

CAVERN\_irbLink &*
Appendix B (Continued)

```cpp
getLink (TdNetCavKey &key)

Returns a link for a given key by reference to the key..

CAVERN_irbLink_c*

getLink (const TcUtString &key)

Returns a link for a given key by name.
```

Class used to encapsulate a CAVERNsoft Connection. This class maintains a table of CAVERNsoft links.

49.1

```cpp
enum connectionType

Connection type enumeration
```

Parameters:

- **UDP** — Unreliable connection type.
- **TCP** — Reliable connection type.

49.2

```cpp
TdNetCavConnection (const TcUtString &connectID, const TcUtString &worldServer, const long &remotePort, TdNetCavConnection::connectionType type = TCP)
```
Appendix B (Continued)

Constructor

Parameters:
- **connectionID** — Specifies the string ID of the connection.
- **worldServer** — Specifies the destination URL of the server for the connection.
- **remotePort** — Specifies the remote port for the connection.
- **type** — Specifies a reliable or unreliable connection type.

```c
void addLink (TdNetCavKey& key)
```

*Link this key to key on remote server*

Parameters:
- **key** — The name of this key must match the name on the remote server.
Appendix B (Continued)

50

class TdNetCavKey

Class used to encapsulate a CAVERNsoft key.

B.0.0.0.97 Public Members

50.1 TdNetCavKey (const TcUtString &kName,
const TcUtString &worldName)

Constructor ............................. 265

50.2 TdNetCavKey (const TcUtString &name, void (*call-
Back)(CAVERN_irbKey_c::CAVERN_irbKeyEvent_t
event, CAVERN_irbKey_c *thisKey,
void* userData))

Constructor ............................. 266

50.3 TdNetCavKey (const TcUtString &name, void (*call-
Back)(CAVERN_irbKey_c::CAVERN_irbKeyEvent_t
event, CAVERN_irbKey_c *thisKey,
void* userData),

TdNetReceivingDock<TcNetEvent *>

*rDock)

Constructor ............................. 266

virtual ~TdNetCavKey () Destructor.

const TcUtString&
getKeyName () const

Returns the key name.

virtual int sendNetCavEvent (TcNetCavEvent &event)
Appendix B (Continued)

Send a network event on the key.

```c
void setCB (void (*call-
    Back)(CAVERN_irbKeyEvent::CAVERN_irbKeyEvent,
        CAVERN_irbKey::*thisKey, void* userData))
    Set the callback for the key.

void setDock (TdNetReceivingDock<TcNetEvent *> *)
    Set the receiving dock for the key.

static void setIRB (CAVERN_irb:* irb)
    Set the IRB for all TdNetCavKey class
    instances.
```

50.1

**TdNetCavKey** (const TcUtString &kName, const TcUtString &worldName)

*Constructor*

**Parameters:**

- *kName* — Name of the key
- *worldName* — Name of the world.
Appendix B (Continued)

50.2

```cpp
TdNetCavKey (const TcUtString &name, void (*callBack)(CAVERN_irbKey::CAVERN_irbKeyEvent &event,
CAVERN_irbKey::*thisKey, void* userData))
```

Constructor

Parameters:

- `name` — Name of the key
- `callBack` — Callback to be triggered when data arrives on the key.

50.3

```cpp
TdNetCavKey (const TcUtString &name, void (*callBack)(CAVERN_irbKey::CAVERN_irbKeyEvent &event,
CAVERN_irbKey::*thisKey, void* userData), TdNetReceivingDock<TcNetEvent *> *rDock)
```

Constructor
Appendix B (Continued)

Parameters:

- **name** — Name of the key

- **callBack** — callback to be triggered when data arrives on the key.

- **rDock** — Receiving dock associated with the key.
template <class T> class TdNetReceivingDock

*Template class for queuing asynchronous network events.*

### Public Members

- `TdNetReceivingDock()`  
  *Constructor.*
- `~TdNetReceivingDock()`  
  *Destructor.*
- `void receiveItem(T item)`  
  *Put an item into the dock.*
- `T processItem()`  
  *Remove an item from the dock.*
- `void clearDock()`  
  *Clear the dock of all items.*
- `bool empty() const`  
  *Check if the dock is empty. Return true if it is empty, false otherwise.*
- `void lockDock()`  
  *Lock the dock prior to calling empty, receiving an item or process item.*
- `void unlockDock()`  
  *Unlocks the dock mutex.*
- `int size() const`  
  *Returns the number of items in the dock.*

---

51

- `void lockDock()`  
  *Lock the dock prior to calling empty, receiving an item or process item.*
Lock the dock prior to calling empty, receiving an item or process item. Locking scheme for processing an item: lockDock if !empty, processItem, unlock dock

Locking scheme for receive lockDock receive Item unlockDock
class TcUtDebug

Debug utility class used throughout Tandem

B.0.0.0.99 Public Members

static TcUtDebug debug

Public static instance of itself.

void debugMsg (const TcUtString &file,

const TcUtString &className,

const TcUtString &method)

Prints a debug message to standard out.
Appendix B (Continued)

class TcUtMutex

Base class for Tandem mutex

B.0.0.0.100 Inheritance

B.0.0.0.101 Public Members

53.1 enum lockResult_t
virtual lockResult_t lock ()
virtual lockResult_t tryLock ()
virtual lockResult_t unlock ()
virtaul ~TcUtMutex ()

enum lock Results   ................. 272

Blocking mutex lock.
Non blocking lock attempt
Unlock mutex.
Destructor.
## 53.1

```c
enum lockResult_t
```

### Parameters:
- DEADLOCK — Deadlock has been detected.
- FAILED — Lock failed.
- GOTLOCK — Lock was successful.
- BUSY — Lock returned busy.
- UNLOCK_OK — Unlock was successful.
Appendix B (Continued)

```cpp
class TcUtSemMutex : public TcUtMutex

Mutex class built on posix semaphore for single process, threaded apps.
```

### B.0.0.0.102 Inheritance

![Inheritance Diagram]

### B.0.0.0.103 Public Members

54.1  

```cpp
TcUtSemMutex (const char *name)
```

*Constructor ............... 274*

```
virtual ~TcUtSemMutex ()

Destructor.
```

```
virtual lockResult_t

lock ()

Blocking mutex lock.
```

```
virtual lockResult_t

tryLock ()

Non blocking lock attempt.
```

```
virtual lockResult_t

unlock ()

Unlock mutex.
```
Constructor

Parameters: \( \text{name} \) — Used for debugging purposes.
Appendix B (Continued)

class TcUtSemMutex_mp : public TcUtMutex

Mutex class built on posix semaphore for multi process apps.

B.0.0.0.104 Inheritance

B.0.0.0.105 Public Members

TcUtSemMutex_mp (sem_t*)
Constructor ....................... 276

TcUtSemMutex_mp ()
Constructor.

void setSemMutex_mp (sem_t*)

virtual ~TcUtSemMutex_mp ()
Destructor.

virtual lockResult_t
lock () Blocking mutex lock.

virtual lockResult_t
tryLock () Non blocking lock attempt.

virtual lockResult_t
Appendix B (Continued)

unlock ()  Unlock mutex.

55.1

TcUtSemMutex_mp (sem_t*s)

Constructor

Parameters:  s — sem structure allocated in shared memory and initialized (sem_init).
Appendix B (Continued)


template <class T>    class TcUtObserver

Abstract Base class for observers

B.0.0.0.106   Public Members

virtual ~TcUtObserver ()  Destructor.
virtual void update (T* subject)  Must be overridden by clients. Called on
                                       observer notification.

B.0.0.0.107   Protected Members

TcUtObserver ()   Constructor.
class TcUtString

Tandem String Class

B.0.0.0.108  Public Members

TcUtString ( const char *cstring = "" )
Constructor

TcUtString ( const TcUtString & str )
Copy constructor

~TcUtString ( )
Destructor

const TcUtString&
operator= ( const TcUtString & rhs )
Copy

const TcUtString&
operator+= ( const TcUtString & rhs )
Append

char* c_str ( ) const
Return C-style string

int length ( ) const
Return string length

char operator[] ( int k ) const
Accessor operator[]

char& operator[] ( int k )
Mutator operator[]

enum
Maximum length for input string
Appedix B (Continued)

ostream& operator<<( ostream & out, const TcUtString & str )
Appendix B (Continued)

```cpp
operator>>(istream & in, TcUtString & str )
```

Input
Appendix B (Continued)

```c
istream& getline ( istream & in, TcUtString & str )
```

Read line
 Appendix B (Continued)

bool operator==( const TCHAR& lhs, const TCHAR& rhs )

Compare ==
Appendix B (Continued)

bool operator!=(const TCHARString& lhs, const TCHARString& rhs)

Compare !=
bool operator< (const TcUtString & lhs, const TcUtString & rhs)

Compare <
bool operator<= ( const TcUtString & lhs, const TcUtString & rhs )
bool operator> (const TcUtString & lhs, const TcUtString & rhs)
Appendix B (Continued)

```cpp
bool operator>= (const TcUtString & lhs, const TcUtString & rhs)
```

*Compare >=*
Appendix B (Continued)

template <class Data> class TcUtStringHashTable

Template String Hash Table

B.0.0.0.109 Public Members

TcUtStringHashTable ()
Constructor.

~TcUtStringHashTable ()
Destructor.

67.1 class iterator : public TcIterator<Data>
Nested Iterator class .................. 289

67.2 int insert (const TcUtString &key, Data item)
Insert an item into the table ............ 292
TcUtStringHashTable <Data> ::iterator
first ()
Returns an iterator to the first item in
the table.

67.3 TcUtStringHashTable::iterator
find (const TcUtString& name)
Returns an iterator to an item with index
matching name ......................... 292

void erase (const TcUtString& key)
Remove an index entry from the table.

bool empty ()
Returns true if the table is empty and
false otherwise.
class iterator : public TcIterator<Data>

Nested Iterator class

Inheritance

Public Members

iterator () Constructor.

iterator (TcUtStringHashTable<Data> &t)
  Constructor  ..................  290

iterator (const TcUtStringHashTable<Data>::iterator & right)
  Copy constructor.

getItem () const
  Returns the current item  ...............  291

void first ()
  Reset the iterator to the first element in the table.

void next ()
  Forward the iterator one element.

Data
Appendix B (Continued)

```cpp
getKey () const
Return the current key index ........... 291

bool end () const
Determines if the end of the table has
been reached. Deprecated. See isDone().

bool isDone () const
Determines if the end of the table has
been reached, same as end().

67.1.4 Data

currentItem () const
Returns the current item ............... 291

const TcUtStringHashTable<Data>::iterator&
operator= (const TcUtStringHashTable<Data>::iterator&
right)
Assignment operator.
```

67.1.1

```cpp
iterator (TcUtStringHashTable<Data> &t)
```

Constructor

Parameters:

- `t` — An instance of a TcUtStringHashTable to be traversed.

67.1.2

```cpp
Data getItem () const
```
Appendix B (Continued)

Returns the current item

Returns the current item. Deprecated. See `currentItem()` It is important to check that the iterator is not at the end before trying to get the current item. Behavior is undefined if this method is called when `end()` returns true.

```
67.1.3

const TcUtfString& getKey () const
```

Return the current key index

Return the current key index. It is important to check that the iterator is not at the end before trying to get the key index. Behavior is undefined if this method is called when `end()` returns true.

```
67.1.4

Data currentItem () const
```

Returns the current item

Returns the current item. It is important to check that the iterator is not at the end before trying to get the current item. Behavior is undefined if this method is called when `end()` returns true.
67.2

int insert (const TcUtString &key, Data item)

Insert an item into the table

Insert an item into the table. The implementation is based on SGI’s hash map. Returns 1 on successful insertion and 0 on failure.

Parameters:

key — Unique string name for hash index. The key

MUST reference dynamically allocated data whose lifespan equals that of the item.

Data associated with the key.

67.3

TcUtStringHashTable::iterator find (const TcUtString & name)

Returns an iterator to an item with index matching name

Returns an iterator to an item with index matching name. If the item is not found the iterator will point to the end of the table and the isDone() method will return true.
class TcUtVector3f

Tandem 3 float utility vector

B.0.0.0.112 Public Members

TcUtVector3f () Constructor.
TcUtVector3f (const float &x, const float &y,
const float &z ) Constructor.

TcUtVector3f (float array[]) Constructor .........................

TcUtVector3f (const TcUtVector3f &right) Copy constructor.

~TcUtVector3f () Destructor.

const TcUtVector3f&
    operator= (const TcUtVector3f &right)
    Assignment operator.

void setVector (const float &x, const float &y, const float &z)
    Set operation.

const float & x () const
    X component accessor.
const float & y () const
    Y component accessor.
const float & z () const
    Z component accessor.

const float & operator[] ( int k ) const
    Accessor operator[i], i must be between 0
    and 2 otherwise an exception is thrown

float & operator[] ( int k )
    Mutator operator[i], i must be between 0
    and 2 otherwise an exception is thrown

const TcUtVector3f&
normalize ()  
\textit{Normalizes the vector}

\texttt{TcUtVector3f}

\texttt{getNormal (const TcUtVector3f \&vec2)}

\textit{Returns a vector normal as defined by this vector cross product vec2}

\texttt{void pack (CAVERNplus\_datapack \&\&datapack)}

\textit{ Serialization pack method for Cavernsoft}

\texttt{void unpack (CAVERNplus\_datapack \&\&datapack)}

\textit{ Serialization unpack method for Cavernsoft}

\begin{center}
\textbf{68.1}
\end{center}

\texttt{TcUtVector3f (float array[])}

\textit{Constructor}

\textbf{Parameters:}

\begin{itemize}
  \item \texttt{array} — An array of 3 float values.
\end{itemize}
Class Graph

1. TcAction
2. TcActionEvent
3. TcActionEventObserver
Appendix B (Continued)

4
TcApplication ........................................................................................................ 111

5
TcAudioPlayer ........................................................................................................ 118

6
TcAvatarManager .................................................................................................... 123

7
TcExControl ........................................................................................................... 124

8
TcInputManager ..................................................................................................... 125

9
TcInteractionManager ........................................................................................... 127
Appendix B (Continued)

10
TcIterator

67.1
iterator

19
TcGfxIterator

11
TcLoader

12
TrackerData

13
TcTrackerManager
Appendix B (Continued)

14
TcTravelScheme
.............................. 139

15
TcTraveler
.............................. 141

16
TcUserComponent
.............................. 144

17
TcGfxPfBackgroundLoader
.............................. 145

40
TdGfxPfObject
.............................. 229

17.1
TcGfxLoadRequest
.............................. 146
Appendix B (Continued)

17.1.1

TcGfxBgLoaderObserver

.......................................................... 148

18

TcGfxCompList

.......................................................... 151

20

TcGfxComponent

.......................................................... 156

29

TcGfxObject

.......................................................... 183

40

TdGfxPfObject

.......................................................... 229

23

TcGfxGLObject

.......................................................... 167

24

TcGfxGroup

.......................................................... 170

36

TdGfxPfGroup

.......................................................... 204
Appendix B (Continued)
Appendix B (Continued)

TcGfxVisitor

TdGfxPfGrabController

TdGfxPfTravAttributes

TcNetConnectionManager

TcNetCavConnectionManager

TcNetEvent

TcNetCavEvent
<table>
<thead>
<tr>
<th>Appendix B (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
</tr>
<tr>
<td>TcNetEventManager</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>48</td>
</tr>
<tr>
<td>TcNetManager</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>49</td>
</tr>
<tr>
<td>TdNetCavConnection</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>TdNetCavKey</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>TdNetReceivingDock</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>52</td>
</tr>
<tr>
<td>TcUtDebug</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>257</td>
</tr>
<tr>
<td>258</td>
</tr>
<tr>
<td>261</td>
</tr>
<tr>
<td>264</td>
</tr>
<tr>
<td>268</td>
</tr>
<tr>
<td>270</td>
</tr>
</tbody>
</table>
Appendix B (Continued)

53
TcUtMutex

55
TcUtSemMutex mp

54
TcUtSemMutex

271

56
TcUtObserver

277

57
TcUtString

278

67
TcUtStringHashTable

288
Appendix B (Continued)

TcUtVector3f

................................................................. 293