
Using the Series 60 Platform Calendar APIs

Version 1.0
July 2003

P L A T F O R M
60
S E R I E S

Contents

1.	Introduction.....	6
1.1	Purpose and Scope.....	6
2.	Using the Agenda APIs	7
2.1	Overview	7
2.2	Agenda File API	7
2.2.1	Agenda models.....	7
2.2.2	Agenda file functionality.....	9
2.2.3	Repeat rules	14
2.2.4	File observers	15
2.2.5	Monitoring progress	16
2.3	Agenda Entries and Instances	16
2.3.1	Entry types.....	17
2.3.2	Instances	18
2.3.3	Agenda IDs.....	18
2.4	Agenda Model Utilities	19
2.4.1	AgnDateTime.....	19
2.4.2	TAgnDate.....	19
2.4.3	TAgnTime	19
2.4.4	TAgnDateTime	19
2.4.5	TAgnVersion	19
2.4.6	Agenda utility panics.....	20
2.5	Using a Client-Side Interface	20
2.6	Using the Agenda Model Engine	20
3.	Using the Calendar Application.....	23
3.1	Accessing the Calendar Data	23
3.2	Activating the Calendar View	23
4.	Calendar Conversion.....	24

Version History

Version	Date	Comment
1.0	25/06/03	Initial Document Release

Legal Notice

Copyright © Nokia Corporation 2003. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

Disclaimer

The information in this document is provided "as is," with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Furthermore, information provided in this document is preliminary, and may be changed substantially prior to final release. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

Glossary

Term	Meaning
API	Application Programming Interface. The interface by which an application accesses the operating system and other services.
DLL	Dynamic Link Library. A library that is linked to application programs when they are loaded or run rather than as the final phase of compilation.
SDK	Software Development Kit
UI	User Interface

1. Introduction

1.1 Purpose and Scope

This document explains how to utilize the calendar facilities on a Series 60 Platform device. The agenda server is the engine that provides the functionality for these features and it is examined along with its associated APIs. Code examples are provided to illustrate how these techniques are used.

2. Using the Agenda APIs

2.1 Overview

The agenda model is a calendar against which entries can be added. For the end user, the agenda model facilitates the scheduling and coordination of events and provides timely reminders of them.

For the developer, several different APIs are available for interaction with the agenda model. Combined, these APIs provide the full range of functionality required to use all of the model's features.

The agenda file API is of primary importance because it provides a generic means of accessing entries in an agenda file. Other APIs are then used to interpret and manipulate the different types of returned entries. Additional features, such as date/time format conversion, are provided by a utilities API. This section covers the functionality and common usage of each API, and demonstrates how to deploy them in conjunction with one another.

2.2 Agenda File API

The agenda file API is the primary API for accessing the agenda model. Through this API, the developer can retrieve and manipulate data stored in an agenda file and be informed of changes to an agenda file. The API also provides support for importing and exporting vCalendar objects.

2.2.1 Agenda models

To access the data in an agenda document, an application must open the agenda model at run time. The agenda model then routes calls to the appropriate server functions. There are three different types of model that can be opened:

2.2.1.1 *Entry model*

The entry model, `CAgnEntryModel`, is the most basic model. It is a suitable choice for an application with no UI and where an indexed search would not be helpful, such as a format conversion application, for example.

If the entry model is used, the agenda data can be accessed as entries. There are four types of entry:

- Appointments (`CAgnAppt`)
- To-dos (`CAgnToDo`)
- Events (`CAgnEvent`)
- Anniversaries (`CAgnAnniv`)

(A further discussion about entry types can be found in Section 2.3, "Agenda Entries and Instances.")

2.2.1.2 Indexed model

The indexed model, `CAgnIndexedModel`, extends the entry model, `CAgnEntryModel`, and provides a more efficient way of accessing the agenda data when only a subset of the data is required. Using `TAgnFilter`, or one of its derivatives, criteria can be specified for filtering data before entries are read. Filters identify which entry types should be involved when tidying or searching the file or when populating a list of instances or entry symbols. The filter is usually used in combination with a date or date range.

As with the entry model, the indexed model should only be used for an application with no UI.

2.2.1.3 Instance model

The instance model, `CAgnModel`, extends the indexed model. The key feature of this model is that it provides a separate object for each instance of a repeating entry; a repeating entry is represented by a number of instances, one for each date and time on which a repeat occurs. This model is therefore suitable for use by applications that have a user interface and wish to display the agenda data graphically — where a user is to be presented with an individual event for each date that the repeat occurs.

Instances are implemented using the same classes as entries, namely `CAgnAppt`, `CAgnTodo`, `CAgnEvent` and `CAgnAnniv`, but when they are accessed using the instance model, a separate object (an instance) is created for each repeat of the entry, with its date and time information set for one of the individual repeats. A non-repeating entry or a to-do entry is represented by one single instance in the instance model.

Note that while `CAgnModel::AddEntryL()` should be used by a view to add a new entry to the agenda model, other data manipulation should generally be performed through use of the indexed model methods. For example, `UpdateInstanceL()`, `DeleteInstanceL()` and `FetchInstanceL()` should be used in preference to `UpdateEntryL()`, `DeleteEntryL()` and `FetchEntryL()`.

The example in Figure 2.1 shows how the instance model can be used to find events on a particular day. A full explanation of the example will not be given here; reading the remaining sections should give the reader a clear understanding of the steps involved.

```
// Create and open model and set up as a client to the agenda
server
TTime today(TDateTime(2003,EMay,27,0,0,0,0));
CAgnModel* model;
RFs theFs;
RAgendaServ* theServer = RAgendaServ::NewL();
theServer->Connect();
theFs.Connect();
CView* view=CView::NewL();
model = view->Model();
model->SetServer(theServer);
```

```

model->OpenL(_L("C:\\agendafile"),660,900,1000,view,EFalse);

// retrieve list of agenda instances for next day that has entries
// ignore to-dos
TAgnDayFilter filter(model);
filter.SetIncludeTodos(EFalse);
TTime startDate(TDateTime(2003,EMarch,25,0,0,0,0));

TTime nextDate = model->NextDayWithInstance(today,
filter,startDate);
CAgnDayList<TAgnInstanceId>* list =
CAgnDayList<TAgnInstanceId>::NewL(nextDate);
model->PopulateDayInstanceListL(list,filter,today);

//cycle through each instance and retrieve text from them
for(TInt j=0; j < list->Count(); j++)
    {
    TAgnInstanceId instanceId = (*list)[j];
    CAgnEntry* entry=model->FetchInstanceL(instanceId);

    TBuf<256> title;
    entry->RichTextL()->Extract(title,0);
    // do something with this text
    ...
    delete entry;
    }

// clean up
delete view;
theFs.Close();
theServer->Close();
delete theServer;

```

Figure 2.1: Use of the instance model to find events

2.2.2 Agenda file functionality

2.2.2.1 Iteration (accessing database entries)

Functions are provided that allow easy movement through all the entries of an agenda file. Essentially, an entry point into the file is established and then movement through the other entries can be achieved, as shown in Figure 2.2.

```

// Set up starting point (first entry)
TBool available = iAgenServ->CreateEntryIterator();

```

```

while (available)
{
    // Get id of entry
    TAgnEntryId id = iAgnServ->EntryIteratorPosition();
    iEntry = iModel->FetchEntryL(id);
    // get text associated with entry
    iEntry->RichTextL()->Extract(iText,0,iEntry->RichTextL()->
DocumentLength());
    console->Printf(KFormat1, &iText);
    // move forward to next entry
    available = iAgnServ->EntryIteratorNext();
    delete iEntry;
}

```

Figure 2.2: Iteration through an agenda file

Date iteration functions are also provided; these are used to move through the entries in date order and can be used for merging and tidying files.

2.2.2.2 *Merging and tidying agenda files*

Merging is the process of adding the entries and to-do lists from one agenda file into another, while tidying means deleting a subset of entries or moving or copying a subset of a file's entries or to-do lists into another agenda file. The functions `CAgnIndexedModel::TidyByDateL()`, `CAgnIndexedModel::MergeL()`, `CAgnIndexedModel::TidyByToDoListL()` are provided for this purpose.

An example of the use of `CAgnIndexedModel::TidyByToDoListL()` is provided in Figure 2.3. The crossed-out entries in a specified to-do list are copied to a stream and then deleted from the original list. (It should be noted that the example in Figure 2.3 is just a code extract, not a full implementation.)

```

// create an array of to-do list ids
CArrayFixFlat<TAgnToDoListId>* todoList = new (ELeave)
CArrayFixFlat<TAgnToDoListId>(1);

//add one id to the list - this specifies the list we are tidying
todoList->AppendL(todoListId);

// Carry out tidying - copy crossed out entries to specified
stream and delete from the original list

viewModel->TidyByToDoListL(todoList, view,
CAgnModel::ECopyAndDelete, CAgnModel::ECrossedOut, targetStore,
targetModelStreamId, CAgnModel::ETidyByToDoListCallbackHow);

```

Figure 2.3: Use of the TidyByToDoListL() function

2.2.2.3 vCalendar support

The vCalendar specification defines an industry-standard format for the exchange of calendar and scheduling information. It is widely supported and facilitates the easy transfer of events and to-do items between applications on different platforms. Within Symbian OS, support for exchanging vCalendars is achieved through `CAgnEntryModel::ImportVCalL()` and `CAgnEntryModel::ExportVCalL()`.

The example in Figure 2.4 demonstrates the use of `ExportVCalL()` to write out all the information in an agenda file in vCalendar format.

```
// Export all agenda entries as a vCalendar file

RFile calFile;
calFile.Replace(fileSession, aVCalFile, EFileWrite);
RFileWriteStream writeStream(calFile);

// Create array of entry Id's
CArrayFixFlat<TAgnEntryId>* entryIdList =
new(ELeave) CArrayFixFlat<TAgnEntryId>(1);
// Iterate through adding each id to the array
TBool available = agnServ->CreateEntryIterator();
while (available)
{
    TAgnEntryId id = agnServ->EntryIteratorPosition();
    entryIdList->AppendL(id);
    available = agnServ->EntryIteratorNext();
}

// Export as vCalendar
model->ExportVCalL(writeStream, entryIdList);
writeStream.CommitL();
writeStream.Close();
calFile.Close();
delete entryIdList;
```

Figure 2.4: Writing out information in vCalendar format using `ExportVCalL()`

`ImportVCalL()` can be used to read in vCalendar entries from a file. The example in Figure 2.5 shows the use of this function and the subsequent adding of the entries to an agenda file.

```
// Imports all agenda entries from a vCalendar file

RFile infile;
infile.Open(theFs, aVCalFile, EFileRead);
RFileReadStream readStream(infile);
```

```

// Create ptr array for new entries
CArrayPtrFlat<CAgnEntry>* entryArray =
new (ELeave) CArrayPtrFlat<CAgnEntry>(1);
model->ImportVCall(readStream, entryArray);
readStream.Close();
infile.Close();

// add entries to the model
TInt size = entryArray->Count();
for (TInt count=0; count<size; count++)
{
    CAgnEntry* entry = (*entryArray)[count];
    model->AddEntryL(entry);
}
entryArray->ResetAndDestroy();
delete entryArray;
}

```

Figure 2.5 Reading in vCalendar information using ImportVCall()

2.2.2.4 To-do lists

A to-do entry, `CAgnToDo`, represents a task to be carried out. A to-do list, `CAgnToDoList`, is, predictably, a list of to-do entries. The properties of a to-do list include name, ID, and display information. To-do lists are held in a list of to-do lists, `CAgnToDoListList`, which is owned by the agenda model. Because there could be more than one to-do list available, it is necessary to call `CAgnToDo::SetToDoListId()` before a new to-do entry can be added to the model. This means it is possible to separate tasks according to their type. For example, individual lists could be compiled for shopping, bill payment, DIY, thus enabling information to be given to the user in a more helpful format. `CAgnEntryModel::PopulateToDoListNamesL()` can be used to retrieve information about the to-do lists currently held by the model.

An example showing the creation of a to-do list and then adding that list to the model is shown in Figure 2.6.

```

CAgnToDoList* todoList=CAgnToDoList::NewL();
todoList->SetName(_L("Bills"));
TAgnToDoListId list1 = model->AddToDoListL(todoList);
delete todoList;
// retrieval of to-do list
todoList = model->FetchToDoListL(list1);
TAgnUniqueId uniqueId1 = todoList->UniqueId();
delete todoList;
// retrieval of to-do list by unique id

```

```

todoList = model->FetchToDoListL(uniqueId1);
delete todoList;

```

Figure 2.6: Creating a to-do list

2.2.2.5 Using the alarm server

An obvious requirement of an agenda-type application is the facility to set alarms. Entries that require an alarm can therefore call the appropriate `SetAlarm()` function, for example, `CAgnAppt::SetAlarm()`, with details of when the alarm should be triggered. The example code in Figure 2.7 shows how to create an alarm.

```

_LIT(KTtxtApptWithAlarm, "An appt with an alarm");
LOCAL_D TTime startDateTime(TDateTime(2003,EMay,6,12,20,0,0));
LOCAL_D TTime endDateTime(TDateTime(2003,EMay,6,12,35,0,0));

//Create an appointment
CAgnAppt* appt=CAgnAppt::NewL(iParaFormatLayer,iCharFormatLayer);
appt->RichTextL()->InsertL(0,KTtxtApptWithAlarm);
appt->SetStartAndEndDateTime(startDateTime,endDateTime);

// set alarm to sound two days before appt date and 615 mins after
// midnight
appt->SetAlarm(TTimeIntervalDays(2),TTimeIntervalMinutes(615));
iModel->AddEntryL(appt);
delete appt

```

Figure 2.7: Creating an alarmed appointment

Activation of alarms is performed through the alarm server, so the next outstanding alarm (or next few outstanding alarms) from an agenda file must be added to the alarm server queue. The class `CAgnAlarm` is provided as an interface between the agenda model and the alarm server.

The procedure for adding an alarm to the alarm server queue is:

- Create a session with the alarm server using `CAgnAlarm::NewL()`.
- Register the alarm with the model using `CAgnEntryModel::RegisterAlarm()`.
- Retrieve the agenda model's next due alarm and set as the session alarm using `CAgnAlarm::FindAndQueueNextAlarmL()` (or `CAgnAlarm::FindAndQueueNextFewAlarmsL()` for queuing more than one alarm).

Addition or deletion of entries from the model results in a call to `CAgnAlarm::FindAndQueueNextAlarmL()`; this ensures that the alarms queued are always valid. Also, the model guarantees that any particular alarm can only feature once in the alarm server queue.

If the session with the alarm server is terminated, then any outstanding alarms are canceled. For a session alarm to outlive the session, the alarm must be orphaned (`CAgnAlarm::OrphanAlarm()`). This allows the alarm to be serviced after the session is closed.

Notification of session termination is possible through implementation of `MAgnAlarmServerTerminationCallBack`. If a `MAgnAlarmServerTerminationCallBack` object is passed as a parameter during construction of `CAgnAlarm`, then a request to be informed when the session has ended can be made through `CAgnAlarm::RequestServerTerminationNotification()`. The notification will be performed through a call to `MAgnAlarmServerTerminationCallBack::AlarmServerTerminated()`.

After disconnection occurs, an attempt may be made to reconnect to the server using `Connect()`. Once a connection has been re-established, a call to `RestartL()` will requeue the agenda model's next outstanding alarm with the alarm server.

2.2.3 Repeat rules

A recurring event can be set as a repeat event. In the entry model this can be seen as one event with information about the repetition, but when displayed to the user, each occurrence should be viewed as an individual event.

A `CAgnRptDef` object is used to define the rules for the repetition interval and these are set using `CAgnEntry::SetRptDefL()`. The repeat definition has a type and stores a set of repeat details (for example, start and end date, interval between repeats). The repeat details are stored using an instance of a `TAgnRpt` derived class and are set using methods such as `SetDaily()` and `SetWeekly()`.

The example in Figure 2.8 shows how to set up an event to recur daily.

```

TTime today;
today.HomeTime();
TTime startTime
(TDateTime(today.DateTime().Year(), today.DateTime().Month(),
today.DateTime().Day(), 10, 0, 0, 0));
CAgnEntry* entry =
CAgnEvent::NewL(paraFormatLayer, charFormatLayer);
CAgnEvent* event = entry->CastToEvent();
event->SetStartAndEndDate(startTime);
entry->RichTextL()->InsertL(0, _L("Repeat Event entry"));
CAgnRptDef* repeat = CAgnRptDef::NewL();
CleanupStack::PushL(repeat);
TAgnDailyRpt daily; // Default daily repeat
repeat->SetDaily(daily);
repeat->SetStartDate(startTime);
repeat->SetRepeatForever(ETTrue);
repeat->SetInterval(1);
entry->SetRptDefL(repeat);

```

```

model->AddEntryL(entry);
CleanupStack::PopAndDestroy();      // repeat
Delete entry;

```

Figure 2.8: Creating a recurring event

2.2.4 File observers

Observers can be created and registered with the observer controller to observe an agenda file. They are then notified whenever an entry or a to-do list is added, updated, or deleted. This is important, for example, to ensure that any view displayed to the user is accurate.

An abstract base class, `CAgnObserver`, is defined and all agenda file observers must derive from this class and implement its pure virtual functions. An example of such a class is defined in Figure 2.9.

```

class CAgnTestObserver : public CAgnObserver
{
    IMPORT_C static CAgnTestObserver* NewL();
    ~CAgnTestObserver();
public: // FROM CAgnObserver
    void StartObserving();
    void StopObserving();
    CAgnObserver* CloneL();
    void Send(TInt aFunction, CAgnEntry* aEntry);
    void Send(TInt aFunction, CAgnToDoList* aToDoList);
    void ExternalizeL(RWriteStream& aStream) const;
    void InternalizeL(RReadStream& aStream);

public: // NEW METHODS

private: // INTERNAL CONSTRUCT
    CAgnTestObserver();
    void ConstructL();
private: // INTERNAL METHODS
};

```

Figure 2.9: Definition of an agenda file observer class

Notification of events is through a call to the appropriate `Send()` function. The parameter `aFunction` that is passed relates to the type of event (defined in `TFunction`), for example, `ENotifyAddEntry`, if an entry was added. So, if an entry is added to the model, `CAgnEntryModel::AddEntry()` calls `ObserverController()->Send(CAgnObserver::ENotifyAddEntry, aEntry)`. This in turn calls the `Send()` function of every registered observer, passing on the same parameters.

The code extract in Figure 2.10 demonstrates how a `CAgnTestObserver` object could be registered as an observer of the agenda file `AgendaTestFile`. (Note that `TestOb` defines the filename of the DLL where the observer's definition can be found.)

```
RAgendaServ* server = RAgendaServ::NewL();
CleanupStack::PushL(server);
User::LeaveIfError(server->Connect());
CleanupClosePushL(*server);

// AgendaModel
CAgnEntryModel* model = CAgnEntryModel::NewL();
CleanupStack::PushL(model);
model->SetServer(server);
model->OpenL(_L("C:\\AgendaTestFile"));

// Observer
CAgnTestObserver* observer = CAgnTestObserver::NewL();
CleanupStack::PushL(observer);
model->RegisterObserverL(_L("TestOb"), observer);
CleanupStack::PopAndDestroy(4); // observer, model, server,
cleanup close
```

Figure 2.10: Registration of an agenda file observer

2.2.5 Monitoring progress

Some of the operations provided by the agenda file API might take a significant amount of time to complete. In applications with a user interface, it is desirable to inform the user if this is the case. This can be accomplished through the use, where available, of asynchronous function calls and the implementation of `MAgnProgressCallback`.

For example, opening the indexed model involves building indexes and could be time-consuming. An asynchronous implementation of `CAgnIndexedModel::OpenL()` is therefore available that takes a pointer to a `MAgnProgressCallback` object to allow progress to be monitored.

Other operations that allow progress monitoring through use of `MAgnProgressCallback` include merging, tidying, and saving.

2.3 Agenda Entries and Instances

In this section, the ideas of agenda *entry* and *instance* will be discussed and the differences between them explained in relation to the different agenda model types. The types of entry supported will also be detailed.

2.3.1 Entry types

`CAgnEntry` defines an abstract base class from which all specific entry types are derived. It holds common entry information, including entry IDs, persistence, and synchronization information.

The available concrete agenda entry types and their purpose are described in the following sections.

2.3.1.1 Appointments, `CAgnAppt`

Appointments are events that begin and end on a particular time during a day. They have a beginning and an ending date as well as a beginning and an ending time (specified using the `TAgnDateTime` class). Appointments differ from events in that they do not have an assigned display time; the display time is inherent in the start date/time.

An appointment whose end date/time is the same as its start date/time is known as a day note. To define a day note, `CAgnAppt::SetStartAndEndTime(const TTime& aStartTime, const TTime& aEndTime)` should be called with a null value for `aEndTime`. A day note has a date (the date component of its date/time) and a display time (the time component of the date/time). The display time can be used to tell the user interface where to display the day note in a view that contains times of day, such as in day or week views.

2.3.1.2 To-dos, `CAgnTodo`

To-dos are task reminders. They are different from appointments and events because they are reminders and therefore do not require begin and end dates. They can have either a due date (to indicate the date by which the action should be carried out) or a crossed-out date (the date the action was carried out). For display purposes, they have a display time and an optional start date (the first date on which it can be displayed).

2.3.1.3 Events, `CAgnEvent`

Events are all-day entries that have a start date and a display time and, optionally, an end date. Events differ from appointments in that they do not have start and end times.

2.3.1.4 Anniversaries, `CAgnAnniv`

An anniversary is an event that can occur only once a year, for example, a birthday. The anniversary entry, by default, is accompanied by the yearly-by-day repeat rule. In addition, a "base year" parameter can be set, thus allowing the number of years from the first anniversary to be calculated and anniversaries to be displayed as "nth year anniversary." If the base year specified is earlier than the start date specified in a repeat rule, the start date would be amended to the value of the base-year date. An exception to this is if the new base year is before the agenda model's minimum year (1980). In this case, the repeat is set to start from the minimum year.

2.3.2 Instances

When creating an entry in the agenda model, information can be set regarding at what intervals the event should be repeated. For example, an appointment could be entered as one event, with the added information that this event should be displayed to the user every week for six weeks. Conceptually, each separate appointment can be viewed as an instance, so in this example, while there is just one entry, there would be six instances.

Instances are implemented using the same classes as entries, but when they are accessed through the instance model, a separate object (an instance) is created for each repeat of the entry, with the date and time information set for each individual repeat. A non-repeating entry or a to-do entry is represented by a single instance in the instance model. The different model types and their uses are explained in Section 2.2.1, "Agenda Models."

2.3.3 Agenda IDs

Several different types of ID exist. This section explains the meaning of each and highlights their differences in terms of common usage.

2.3.3.1 ID types

Entry ID

An Entry ID identifies an entry with respect to its storage location in a database. Therefore, the Entry ID does not uniquely represent an entry. It is used as an indexing mechanism. This type of ID is represented by the `TAgnEntryID` class.

Instance ID

An Instance ID identifies an instance of an entry in an agenda model. It comprises an Entry ID and an agenda date. This type of ID is represented by the `TAgnInstanceID` class.

Unique ID

A Unique ID is assigned when an entry or to-do list is created in an agenda file and is preserved during updates. It can therefore be used to identify an entry during synchronization. The `TAgnUniqueID` class is used to represent this type of ID.

2.3.3.2 Example usage of IDs

IDs are obviously a means of identifying a particular entry or instance. The functions `CAgnEntryModel::FetchEntryL()` and `CAgnModel::FetchInstanceL()` can be used to retrieve a given entry from the agenda file. The example in Figure 2.11 shows how to obtain the Unique ID of an entry and then use this for retrieval of the entry at a later date.

```
CAgnEntry* entry =
CAgnAppt::NewL(paraFormatLayer, charFormatLayer);
CAgnAppt* appt = entry->CastToAppt();
appt->SetStartAndEndDateTime(startTime, endTime);
entry->RichTextL()->InsertL(0, L("Entry 1"));
```

```

model->AddEntryL(entry);
TAgnUniqueId uniqueId1 = entry->UniqueId();
delete entry;
...
entry = model->FetchEntryL(uniqueId1);
...

```

Figure 2.11: Use of TAgnUniqueId

2.4 Agenda Model Utilities

Symbian OS is very mindful of memory constraints. For this reason, within the agenda model, dates and times are stored as TUint16 integers rather than as instances of the standard TTime class, thus saving space. A consequence of this design choice is the restriction of the valid date range, namely January 1, 1980 –December 31, 2100. Dates are represented as a number of days from January 1, 1980.

The agenda model utilities API aids the conversion between agenda date and time formats and standard representations of dates and times, and offers other useful functionality. The main classes are described in this section.

2.4.1 AgnDateTime

The `AgnDateTime` class provides the developer with a number of static functions useful when dealing with dates and times. Translation between different time formats can be accomplished, information about the minimum and maximum date of the agenda model retrieved, and time intervals between dates calculated.

2.4.2 TAgnDate

`TAgnDate` represents an agenda model date, that is, a number of days from the agenda model's minimum valid date (January 1, 1980). `TAgnDate` is defined as type `TInt16`.

2.4.3 TAgnTime

`TAgnTime` represents an agenda model time, that is, the number of minutes from midnight. (Values must be in the range 0 – 1439.) `TAgnTime` is defined as type `TInt16`.

2.4.4 TAgnDateTime

`TAgnDateTime` is a container class for an agenda date and time. It contains `TAgnDate` and `TAgnTime` member data and can be used to manipulate this data.

2.4.5 TAgnVersion

`TAgnVersion` represents the version number of the agenda model (consisting of major, minor, and build numbers). An object of this class is returned by the `CAgnEntryModel` functions `ModelVersion()` and `FileVersion()`. `ModelVersion()` returns the version of the model object being used, while

`FileVersion()` returns the version of model that was used to create the currently loaded file.

2.4.6 Agenda utility panics

In Symbian OS, an unrecoverable application error is called a panic. The developer will be alerted to such an occurrence by a pop-up note containing some details of the problem. A panic caused by an error in the agenda model will have a category of “agenda model,” a full list of reason codes can be found in the SDK (*Symbian OS v6.1 Edition for C++ >> API Reference >> Agenda Model Utilities >> Agenda Utility Panics*).

2.5 Using a Client-Side Interface

The agenda model can be used to gain direct access to agenda files (when operating in normal mode) or it can route calls through the agenda server (client mode). Some earlier releases of Symbian OS did not support this client-server architecture, but, over time, the need arose to allow and coordinate database access between multiple applications simultaneously. The client-side interface has been provided for this purpose and is now the standard means of accessing agenda files. It is still possible to use the direct access methods, but this approach is not encouraged and should only be considered if the agenda file to be accessed is used exclusively by the application.

Using the client-server architecture is straightforward and differs from direct access only in the association of the agenda model with a client-side API; this is achieved through `CAgnEntryModel::SetServer()`. Once this association has been made, all agenda model function calls are, where necessary, routed through the client-side API to the server. It should be noted that because only one file can be open at any one time in a given server session, a separate instance of the `RAgendaServ` class will be required for each agenda file that is to be opened. The example code in Figure 2.12 demonstrates how to use the agenda model in client mode.

```
RAgendaServ* agnServer = RAgendaServ::NewL(); // allocate and construct
server
CleanupStack::PushL(agnServer);
agnServer->Connect(); // connect to the agenda server
CleanupClosePushL(*agnServer); // guarantees that the server's Close()
method gets called
CAgnEntryModel* model = CAgnEntryModel::NewL(); // allocate and
construct model
CleanupStack::PushL(model);
model->SetServer(agnServer); // set server pointer for model
model->OpenL(fileName) // Open file using server
// ...Use agenda model API as normal - invokes corresponding server
functions
CleanupStack::PopAndDestroy(3); // model, close session with server,
agnServer
```

Figure 2.12: Using the agenda model in client mode

2.6 Using the Agenda Model Engine

The use of many of the APIs discussed in this document is demonstrated in the *UpdateEntry* example found in the SDK (`\EpoC32Ex\AppEngine\AgendaModel\UpdateEntry`). The purpose of the application is to amend the first entry for a particular day so that if it is

currently crossed out, it is changed to not crossed out and vice versa. This section presents the main elements of that example. (Note that when running this application, the agenda file `agendata2` should be moved to the directory `\Epoc32\Wins\c.`)

Since the *UpdateEntry* example is console-based, it does not have the usual view architecture. A dummy view class has therefore been written so that the use of the mix-in classes `MAgnModelStateCallBack` and `MAgnProgressCallBack` can be shown.

The first step carried out by the application is the creation of a `CAgendaFileReader` object. It should be noted that during construction, the date of the entries that we will be searching for is set to March 14, 2000.

```
CAgendaFileReader* reader=new (ELeave) CAgendaFileReader
(TDateTime(2000,EMarch,14,0,0,0));
```

A `RAgendaServ` instance is then created and connection is made of this client-side object to the agenda server.

```
iAgenServ = RAgendaServ::NewL();
iAgenServ->Connect();
```

The dummy view is then created and this in turn creates a `CAgnModel` object. The model is created by the view so that the state of the model can be monitored (the view implements `MAgnModelStateCallBack`); the model created is then assigned to `CAgendaFileReader::iModel`. `CAgnModel::SetServer()` is then called so that we can operate in client-server mode. (Note that the call to `CAgnModel::SetMode()` is superfluous; `SetServer()` automatically sets the mode to `EClient`.)

```
CView* view = CView::NewL();
iModel = view->Model();
iModel->SetServer(iAgenServ);
iModel->SetMode(CAgnEntryModel::EClient);
```

The model is opened asynchronously. The view implements `MAgnProgressCallBack` and can be used to monitor the progress of the operation. The agenda file `agendata2` will be opened. A call is made to `RAgendServ::WaitTillLoaded()` to ensure that the file has been loaded before a request is made to access it.

```
TRAPD(error, (iModel->OpenL(_L("C:\\agendata2.dat"), KDefaultTimeForEvents,
KDefaultTimeForAnnivs, KDefaultTimeForDayNote, view, EFalse)));
iAgenServ->WaitUntilLoaded();//wait until the file is loaded
```

At this point, all the entries for the date March 14, 2000 are retrieved from the file. It is possible to filter the type of entries retrieved by calling functions from `TAgnFilter`, but in this example, we consider all entry types.

A `CAgnEntryDayList` object is created with the date March 14, 2000 passed as a parameter. This list is then populated with the Instance IDs for all entries on the given date using `CAgnModel::PopulateDayInstanceListL()`.

```
// create a day list
iDayList=CAgnDayList<TAgnInstanceId>::NewL(iDay.DateTime());
```

```
// fill the day list
// filter allows particular types of instance to be excluded/included
// Default is to include all types of entry.
TAgnFilter filter;
iModel->PopulateDayInstanceListL(iDayList,filter,iDay.DateTime());
```

The `UpdateEntry()` function is then called to alter the crossed-out property of the first instance in the list; instances are sorted by display time. The instance is retrieved using `CAgnModel::FetchInstanceL()` and then the appropriate function, dependent upon instance type, is called to amend the crossed-out status.

```
CAgnEntry* entry=iModel->FetchInstanceL((*iDayList)[0]);
if (entry->Type() == CAgnEntry::ETodo)
{
    CAgnTodo* todo=entry->CastToTodo();
    if (todo->IsCrossedOut())
        todo->UnCrossOut();
    else
        todo->CrossOut(iDay);
}
else
    entry->SetIsCrossedOut(!entry->IsCrossedOut());
```

The file is then closed with the updated instance information saved.

```
iAgenServ->CloseAgenda();
```

3. Using the Calendar Application

The Calendar application is a standard Series 60 Platform application available to users to organize their appointments and meetings. In addition, a To-Do application is also present for users to make a list of things to do. Developers are able to take advantage of these applications within their own applications.

3.1 Accessing the Calendar Data

Developers are able to create their own agenda files or share information across applications. Both the Calendar and To-Do applications share the file `C:\System\Data\Calendar` to store events. Developers are able to access this file, use or modify the data contained in it, or observe its entries. However, it is imperative that the file is not corrupted because this would result in all data contained within being lost. Care should therefore be taken if this file is to be used.

3.2 Activating the Calendar View

It is possible for the developer to activate a view from an external application from within the code. In particular, a view from either the Calendar or To-Do application could be displayed. The `ActivateViewL()` method of the `AppUI` class can be used to produce the desired result. This function takes an application view ID, `TVwsViewId`, as a parameter. The example code in Figure 3.1 shows how to activate the day view of the Calendar application.

```
// Open calendar application in day view
const TUid KCalendarUid = { 0x10005901 }; // Application Uid
const TUid KCalendarViewUid = { 3 }; // View Uid of the external
application
AppUi() ->ActivateViewL(TVwsViewId(KCalendarUid, KCalendarViewUid));
```

Figure 3.1: Activating the Calendar application day view

The application Uid of the Calendar application is 0x10005901.

The view Uids are:

- 0x01 – month view
- 0x02 – week view
- 0x03 – day view

The application Uid of the To-Do application is 0x10005900.

(Further explanation of this technique can be found in the document *Utilizing External Application Views*, available from [Forum Nokia](#).)

4. Calendar Conversion

In the SDK documentation, functionality to provide conversions between different calendar types is described. Unfortunately, many of the classes, including `TChineseCalendar`, `TGregorianCalendar`, `TArithmeticalCalendar` and `TAstronomicalCalendar` have been deprecated and are not supported by the Series 60 Platform. The header files `calconv.h`, `calconvcalendar.h`, `calconvarithmeticalCal.h`, and `calconvastronomicalCal.h` are not supplied with Series 60 SDK 1.0 for Symbian OS, Nokia Edition (www.forum.nokia.com/).

It is anticipated that a future release of the Platform, Series 60 Platform 2.0, will provide an API to convert between a Chinese date and `TDateTime`.