ChronoTrack Reference Manual

What is ChronoTrack?

ChronoTrack is an application and the development environment used to build it. It was created by the Synergex Professional Services Group (PSG) to demonstrate the capabilities of the latest released versions of Synergy/DE. It encompasses many of the modern tools and technologies available to the Synergy/DE developer today.

The ChronoTrack application enables consultant departments to manage their projects and consultants. It includes the following components:

- Windows application. The main ChronoTrack application.
- Monitor application. Enables consultants to view information about their projects.
- Web service. Used by the monitor and mobile applications to access the ChronoTrack database.
- Web site. Enables users to access the main ChronoTrack application from a Web browser.
- Mobile application. Gives users access to the ChronoTrack application from mobile devices.
- Dashboard. Displays ChronoTrack data via a dashboard.

About this Reference Manual

There are several areas within the ChronoTrack development environment. This document describes the structure of the environment and explains how to build and use the various ChronoTrack components.

The ChronoTrack Development Environment section walks through the physical structure of the ChronoTrack folders. It details what files reside in which directories so you can easily locate the example code you require. For example, if you wish to examine the code that shows how to present Synergy data to a .NET form, look in the DataEntities folder under the ChronoTrack project. To see how .NET forms are hosted and processed by Synergy Language, look at the Applications folder.

There are a number of prerequisites for building and running ChronoTrack. See the Building ChronoTrack section below for a complete list.

Getting Help

If you have any questions, comments or suggestions while using ChronoTrack, please email them to Synergy/DE Developer Support at support@synergex.com.

The ChronoTrack Development Environment

The ChronoTrack development environment must be installed onto the C: drive. It has the following folder structure:

- C:\Development\SynPSG
    - ChronoTrack
    - Core
    - ini
    - NetCommon
    - System

Naming Conventions

We developed ChronoTrack based on a coding standards document, which you can request from your Synergy/DE account manager. One area detailed in the standards document is naming conventions. ChronoTrack has introduced a new file extension, ".dbc". Any files with an extension of ".dbc" are class files; that is, source files that contain class definitions. To view these ChronoTrack source files in Synergy/DE Workbench and display the correct color coding and Synergy Language awareness, you may need to do the following:

1. In Workbench, navigate to the Tools→Options→File Extension Setup… menu entry.
2. In the Extension Options dialog, click the "New" button.
3. In the New Extension dialog:
    a. Type "dbc" into the Extension: edit control.
    b. Select the "dbl" entry in the Refer to→ combo box.
    c. Click the OK button.
4. Click the OK button.

The “System” Folder

The “System” folder contains the development area for some base classes. It contains a single file and a number of sub-folders. One of the files in this folder is the Workbench workspace file. Opening this file from within Workbench will load all the associated projects.

The structure of this environment is:

Folder Name	Description
exe	Contains the executable libraries. These libraries include:

Name	Use/Contains
SynPSGSystem.elb	Contains the Convert, DateTime, StringUtil and TimeUtil classes
SynPSGSystemIO.elb	Contains the Stream, FileStream and MemoryStream classes
SynPSGSystemNet.elb	Contains various classes that mimic the System.Net classes. These include the TcpClient, FtpClient classes and all the required classes for email management.
SynPSGSystemText.elb	Contains the text encoding routines.

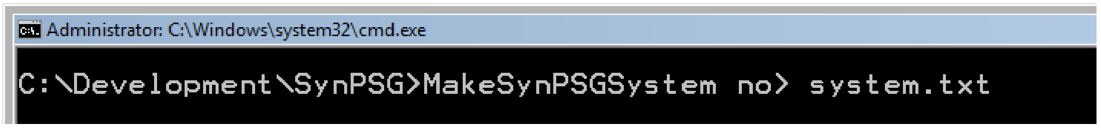
Folder Name	Description
hdr	Contains the prototyped header files. These are generated when the application is built.
lib	Contains the object libraries. No applications build against the object libraries; they are simply used to generate the executable libraries (ELBs).

📂 obj	Parent object folder that contains the child folders where the compiled object files are stored - detailed below.
📂 →Base	Contains the object files from the base source files.
📂 →IO	Contains the I/O object files.
📂 →Net	Stores the net object files.
📂 →Text	Holds the text object files.
📂 prj	Contains the Workbench project files. Do not open these project files individually; instead, open the System workspace file from within Workbench. This will ensure that all of the relevant projects are accessible.
📂 rps	This folder is empty and is defined as a placeholder.
📂 src	Parent folder that contains the child folders that store the source code files.
📂 →Base	Contains the following source files: • Convert.dbc • DateTime.dbc • StringUtil.dbc • TimeUtil.dbc This folder also contains the build files to enable you to build the source files within this folder. To build on Windows, execute the makeit.bat script; the makeit_unix file can be used to build the source files on UNIX.
📂 →Includes	Contains the include files referenced by the source files.
📂 →IO	Contains the following source files: • FileStream.dbc • MemoryStream.dbc • Stream.dbc

	This folder also contains the build files to enable you to build the source files within this folder. To build on Windows, execute the makeit.bat script; the makeit_unix file can be used to build the source files on UNIX.
→Net	Contains the class files for the email, SMTP, and socket classes. This folder also contains the build files to enable you to build the source files within this folder. To build on Windows, execute the makeit.bat script; the makeit_unix file can be used to build the source files on UNIX.
→Text	Contains the ASCII encoding class and the build scripts.

Building the Source and Class Files in the System Folders

Each source folder (detailed above) contains a makeit.bat script file that will enable you to build the source/class files within each area. Additionally, if you wish to build everything under the System folder, you can execute the MakeSynPSGSystem.bat command script located in the C:\Development\SynPSG folder. We recommend that you use this build script as opposed to executing the individual makeit.bat scripts. To capture and verify the build output, use the following command:



```
C:\Development\SynPSG>MakeSynPSGSystem no> system.txt
```

On completion you can view the system.txt file for any issues. The command accepts a single argument. If you specify “NO” for the first argument, the build script will not pause when completed.

The “ini” Folder

The ini folder is used to store the synergy.ini files, which contain various settings required for ChronoTrack. The environment variable SFWINIPATH is set to point to the file in this folder.

The “NetCommon” Folder

The NetCommon folder is the development area for building the Synergy code to wrap the .NET assemblies that are used by the ChronoTrack application.

.NET assemblies are used to provide the user interface presentation layer. With the introduction of Synergy/DE 9.1.3, we now have the ability to incorporate .NET technologies within a Synergy application. The structure of the environment is:

Folder Name	Description
bat	Contains various command scripts that build the required .NET wrappers.
exe	Contains the executable library SynPSGNetCore.elb. This library contains the generated routines that wrap the SynPSGNetCore.dll assembly. (See the Core section below.)
genhdr	Contains the prototyped header files. These are generated when the wrapped code is prototyped.
lib	Contains the object libraries. When generating code to wrap the .NET assemblies, there are many hundreds (even thousands) of Synergy routines created. These can be stored in object libraries. However there is a limit to the number of routines that can be stored in an ELB. Therefore the main applications link against these object libraries.
obj	Parent object folder that contains the child folders where the compiled object files are stored, detailed below.

📂 →ChronoTrack	Contains the object files for the routines that wrap the three user interface assemblies. There are currently three different UI assemblies. • There is a version incorporating the Infragistics controls (ChronoTrackNetUIIN). • (ChronoTrackNetUIDX) incorporates the DevExpress controls. • There is a Microsoft version (ChronoTrackNetUIMS).
📂 prj	Contains all Workbench project files. Do not open these project files individually; instead, open the NetCommon workspace file in Workbench. This will ensure that all the relevant projects are accessible. This workspace can be found in the C:\Development\SynPSG\NetCommon folder.
📂 src	Parent folder that contains the child folders that store the source code files.
📂 →ChronoTrack	Contains the files required to build and prototype the assembly wrapping routines. The files are: • ChronoTrackNetUI.bat ○ This command script will generate the wrapping code for the Infragistics UI assembly, prototype it, and compile it into the object library. • ChronoTrackNetUI.xml ○ This file is used by the GENNET utility to define the required assemblies and classes within these assemblies to wrap. There are also ChronoTrackNetUIIN.inc and ChronoTrackNetUIIN.dbl, and ChronoTrackNetUIIN1.dbl, which are generated by the GENNET utility.

	• ChronoTrackNetUIDX.bat ○ This command script will generate the wrapping code for the DevExpress UI assembly, prototype it, and compile it into the object library. • ChronoTrackNetUIDX.xml ○ This file is used by the GENNET utility to define the required assemblies and classes within these assemblies to wrap. There are also ChronoTrackNetUIDX.inc, ChronoTrackNetUIDX.dbl, and ChronoTrackNetUIDX1.dbl, which are generated by the GENNET utility. • ChronoTrackNetUIMS.bat ○ This command script will generate the wrapping code for the Microsoft UI assembly, prototype it, and compile it into the object library. • ChronoTrackNetUIMS.xml ○ This file is used by the GENNET utility to define the required assemblies and classes within these assemblies to wrap. There are also ChronoTrackNetUIMS.inc, ChronoTrackNetUIMS.dbl, and ChronoTrackNetUIMS1.dbl, which are generated by the GENNET utility.

Building the Source and Class Files in the NetCommon Folders

To build the wrapped assemblies, you can utilize the command scripts in the bat folder.

The "buildassembly.bat" command script is used to rebuild the .NET assemblies. This script will attempt to rebuild the required assemblies. The command script accepts the following arguments:

- Configuration.
    - Debug – builds the debug version
    - Release – builds the release version
- Namespace
    - ChronoTrackNetUI
        - Builds the Infragistics version
    - ChronoTrackNetUIDX
        - Builds the DevExpress version
    - ChronoTrackNetUIMS
        - Builds the Microsoft version
    - ALL
        - Builds all the above

For example, to build the Infragistics version:

```
buildAssembly Debug ChronoTrackNetUI
```

**Note: The command options are case sensitive.

The "AddToGac.bat" script will add the UI assemblies to the GAC. This is required because the –s option used for the GENNET utility requires the assemblies to be loaded in the GAC. The command script accepts the following arguments:

- Configuration.
    - Debug – builds the debug version
    - Release – builds the release version
- Namespace
    - ChronoTrackNetUI
        - Builds the Infragistics version
    - ChronoTrackNetUIDX
        - Builds the DevExpress version
    - ChronoTrackNetUIMS
        - Builds the Microsoft version

- ALL
    - Builds all the above

For example, to add all of the required assemblies to the GAC:

```
AddToGac Debug ALL
```

**Note: The command options are case sensitive.

The "build.bat" command script is used to build the Synergy code that wraps the .NET assemblies. This script uses the command files detailed above in the various subfolders within the "src" folder.

The command script accepts the following arguments:

- Configuration
    - Debug – builds the debug version
    - Release – builds the release version
- Namespace
    - ChronoTrackNetUI
        - Builds the Infragistics version
    - ChronoTrackNetUIDX
        - Builds the DevExpress version
    - ChronoTrackNetUIMS
        - Builds the Microsoft version
    - ALL
        - Builds all the above

For example, to build the Synergy code to wrap the DevExpress UI assembly and create the ChronoTrackNetUIDX.olb object library:

```
Build Debug ChronoTrackNetUIDX
```

**Note: The command options are case sensitive.

Instead of using these individual build scripts to build the individual components of ChronoTrack, we recommend that you use the MakeChronoTrack.bat command script (detailed below) to successfully build the ChronoTrack application.

The "Core" Folder

The Core folder contains Synergy code that implements a .NET assembly called SynPSGNetCore. The SynPSGNetCore assembly is a collection of classes that enable "interop" between the ChronoTrack application and the ChronoTrack UI assemblies.

The Core folder also contains the code for the base DataEntity and Utility classes. These are explored in more detail below.

The structure of the environment is:

Folder Name	Description
📂 exe	Contains the CoreDataEntities and CoreUtilities executable libraries. The main ChronoTrack application links to these libraries.
📂 hdr	Contains the generated Synergy prototype. All classes within the DataEntity and Utility classes are prototyped.
📂 lib	Contains the object libraries. These object libraries are not referenced by the ChronoTrack application; they are simply used to generate executable libraries.
📂 obj	Contains the child folders where the compiled object files are stored.
📂 →DataEntities	Contains object files for the DataEntity routines.
📂 →Utilities	Contains object files for the classes in the Utility namespace.
📂 Prj	Contains the Workbench project files. Do not open these project files individually; instead, open the Core workspace file in Workbench. This will ensure that all the relevant projects are accessible. This workspace can be found in the C:\Development\SynPSG\Core folder.
📂 src	Contains the child folders that store the source

	code files.
📁 →DataEntities	Contains the source files for the classes within the DataEntity namespace. See the DataEntities section below. It also contains the build command scripts.
📁 →Utilities	Contains the utility class files. See the Utilities section below. This folder also contains the command script that prototypes and builds the Utility classes.
📁 VSSolutions	Contains the .NET assemblies.
📁 →SynPSGNetCore	Contains the Visual Studio solution and all relative files required to build the SynPSGNetCore assembly. See the SynPSGNetCore section below for full details of the classes within this assembly.

DataEntities

The DataEntity classes are the cornerstone of ChronoTrack's ability to expose Synergy DBMS (Synergy Database Management System) data to the array of clients available within ChronoTrack.

There are a number of classes, which we shall explore below.

The "SDMSio" Class

The base class is the SDMSio class, defined in the SDMSio.dbc file. As mentioned previously, the "dbc" extension identifies a Synergy class file. The SDMSio class defines a number of methods that enable access to the physical data file. The class constructor accepts the file name you wish to open.

The class exposes a number of methods that enable direct access to the associated data file. All the methods (except DeleteAllMatching() and DataConsistency()) return a Boolean value indicating the success/failure of the call, or will raise the appropriate exception:

- **CheckExists().** Accepts a key value and a key of reference and validates that a record exists in the file.

- **Locate().** Accepts a key value, key of reference, and perform lock flag. The method will locate and optionally lock the required record.
- **ReadFirst().** Accepts a key of reference and perform lock flag. Locates the first record in the file.
- **ReadNext().** Accepts a perform lock flag. Returns the next record in the file.
- **ReadPrev().** Accepts the perform lock flag. Returns the previous record in the file.
- **ReadLast().** Accepts key of reference and perform lock flag. Returns the last record in the file.
- **Update().** Updates the currently locked record.
- **Remove().** Deletes the currently locked record from the file.
- **Create().** Creates a record in the file containing the current record data.
- **CompareAndSave().** Retrieves and locks the original record from the file, checks it is the same as the originally read record and if so, updates it with the new record details. The SDMSio class implements optimistic locking (see below).
- **FindAndDelete().** Uses the current record data to locate the record from the file and delete it.
- **DeleteAllMatching().** Accepts a primary key segment, locates each record in the file that matches, and deletes it from the file.
- **DataConsistency().** Accepts a key value and executes any bound OnDeleteEventHandler method. For more information see the section below on Event Handlers.

The DeleteAllMatching() method is useful when you have a parent/child relationship between files. If you delete the parent from one file, all matching child records from the related file need to be deleted. DeleteAllMatching() is a simple way of accomplishing this. An example can be found in the C:\Development\synPSG\ChronoTrack\src\DataEntities\CustomerOnDelete.dbc files.

This SDMSio class exposes a number of properties:

- **IncludeGRFA.** Enables the consumer of the class to indicate if the GRFA should be included in the record area. See the section on Optimistic Locking below.
- **PrimaryKeyValue.** Returns the primary key value for the current record data.
- **KeyValue.** Accepts the key of reference and returns the key value for that key of reference for the current record data.
- **GRFA.** Gets or sets the current record data's GRFA. See the section on Optimistic Locking below for a description of GRFA.
- **DataArea.** Returns the current record data. This enables you to access the data that was returned after an operation, to retrieve the data from the file, or to modify the data prior to updating the data file. The methods within this class do not return the actual data, so this property provides the access mechanism to the data.

- **HashCode.** Returns the hash code for the current record's data. See the section on Optimistic Locking below for description of a hash code.
- **OriginalHashCode.** Returns the hash code for the record that was originally read from the file. See the section on Optimistic Locking below for description of a hash code.
- **OriginalDataArea.** Returns the data that was originally read from the file.
- **RecordLockOption.** Sets the required record locking mechanism. The LockedAction enumeration has the following values:
    - **WaitOnLock:** Repeatedly attempts to read the record until the lock is released and the read is successful. There will be a sleep of one second between retries.
    - **TimeOutOnLock:** Retries for *n* number of times. The timeout count can be set using the MaxRetry property below.
    - **ReturnNoRecord:** If a record is locked, simply returns with no record and a method return value of False.
    - **ThrowExpection:** The lock will cause an exception to be thrown. The calling program should trap this using a try-catch structure.
- **MaxRetry.** Sets the maximum number of read/lock retry's that should be attempted.
- **LastErrorMessage.** Can be accessed to identify the last recorded error message when errors are trapped and the methods are set to return a True/False value.

Optimistic Locking

Optimistic locking is a method of controlling locks within Synergy DBMS files. By default, when a record is read and the channel is opened in update mode, the record will be locked. If your application then performs UI input, that lock could be left in the file for some time. This causes other users to experience "record locked" errors. Optimistic locking prevents the record that is read from being locked. Instead, the RFA of the record is recorded and a hash code value identifying the contents of the record area is stored. This creates a GRFA.

A GRFA is a 22-byte field that contains the original RFA value and the hash code. The hash code is an integer value that represents the individual bytes that are contained within the record area. The hash code is generated by using a small C++ DLL (shared object file on UNIX/shared image on OpenVMS). It accepts a string of characters (the record area) and returns a unique integer value. Different string values return different integer values; matching string values return matching integer values.

This GRFA is stored in each record that is read from the Synergy DBMS file. When an operation to update the file with the record data is performed, the following steps are performed:

1. Using the GRFA, extract the RFA from the current record data area and use this to locate the original record.
2. Given a successful location of the record, calculate the hash code for this original record.
3. Extract the hash code from the GRFA for the current record area and compare it to the calculated hash code for the record that has just been read.
4. If the hash codes match, allow the update to be performed.
5. If the hash codes do not match, execute any registered UpdateConflict() event handler. For details of the UpdateConflict() event handler, see the Event Handlers section below.

Event Handlers

The event handlers enable additional processing when certain events happen. The SDMSio enables the following event handlers to be registered.

- **PreStoreEventHandler**. If registered, this event will be fired prior to a store operation on the file. Inside the event you can validate and update the record area prior to the store being performed. For example, you can ensure that key fields are correctly cased.
- **PostStoreEventHandler**. If registered, this event will fire once the store operation on the file is successful. The event can be used to perform post-store operations. For example, you may want to send an email when new records are stored in a particular file.
- **PreUpdateEventHandler**. If registered, this event will fire prior to an update on the file. Like with the PreStoreEventHandler, you can validate data and ensure correct casing of data elements/fields.
- **UpdateConflictEventHander**. When registered and an update conflict happens (see Optimistic Locking above), this event method can be used to see what the update conflict is and decide if the update can continue or not.
- **OnDeleteEventHandler**. Prior to a record being deleted from a file, this event, if registered, will be executed. Here you can perform additional logic. For example, you may want to delete related records from other files.
- **FormatTransViewEventHandler**. You can register a method to be called prior to the details of the transaction being processed and added to the TransactionViewer class.

The event handlers all have similar interfaces. Each one is defined in its own class file. Each class is defined as abstract, meaning the class cannot be instantiated but must be extended. Each class has a single method, called EventMethod, defined as an abstract method. This ensures that any extending class must implement a method called EventMethod. These base classes are designed to be extended by the application DataEntity classes. (See ChronoTrack DataEntities below.) The signature of the EventMethod method defines two arguments. The first is an object which, when executed, will be the calling object. The second is the event argument object. The event argument object enables communication between the SDMSio object and the EventMethod.

Each event has an associated event argument class, with the naming convention of *EventName*EventArgs. These event argument classes enable communication between the SDMSio object and the event method.

To assign event handlers, the SDMSio class provides the following methods:

- SetPreStoreEventHandler()
- SetPostStoreEventHandler()
- SetPreUpdateEventHandler()
- SetUpdateConflictEventHandler()
- SetFormatTransViewEventHandler()
- SetOnDeleteEventHandler()

Data Replication

The Core SDMSio class enables the replication of data maintained within the data file. When data modifications (stores, updates, deletes) are performed on the file, the details of these changes are stored in a replication file. This file is then used by the SQL Replicator application (see below) to replicate the changes into a relational database. To enable replication, the SDMSio class needs to know where the primary or replication key is within the record area. To facilitate this, the SDMSio class exposes a method that enables the consumer of the class to define the location within the record where the unique key resides. The method is called SetReplicationDetails(). It accepts the structure name, the actual record area, and the actual key fields. These details are then recorded and are used when replication details are recorded in the replication control file. For more details see the ChronoTrack SQL Replication section below.

The "SDMSCollection" Class

The SDMSCollection class is used to expose the underlying Synergy DBMS data as an ArrayList. An Array List is a dynamic array of elements with a zero based index.

The constructor of the SDMSCollection class accepts an SDMSio object. This SDMSio object enables direct access to the underlying Synergy DBMS data file.

The class has the following methods:

- **Count().** Accepts a partial key and a key of reference. The method returns the number of records that match the partial key. In doing so it also builds up an internal ArrayList collection.
- **Count().** Overridden method that returns the total number of records in the file. The internal ArrayList collection is also populated.
- **Count()**. Overridden method that accepts a ViewType and an alternative structure. The alternative structure is used to allow correct population of the ArrayList collection. This method will return the total number of records in the file. The internal ArrayList collection will be populated. The ViewType enumeration has the following members:
    - Full. Return the full record area.
    - Brief. Return a limited number of fields from the structure.
    - Combined. Return fields from more than a single structure.
- **Count().** Overridden. Accepts the ViewType and alternative structure. Also accepts a partial key and key of reference. Returns a count of the matching records. The internal ArrayList collection will be populated.
- **Collection().** Accepts a partial key and a key of reference. The method returns the internal ArrayList collection containing the matching records.
- **Collection().** Overridden method that returns an ArrayList collection containing all the records in the file.
- **Collection()**. Overridden method that accepts a ViewType and an alternative structure. The alternative structure is used to enable correct population of the ArrayList collection. This method will return an ArrayList collection containing all the records in the file.
- **Collection().** Overridden. Accepts the ViewType and alternative structure. Also accepts a partial key and key of reference. This method will return an ArrayList collection containing all the matching records in the file.

Event Handlers

There is a single event handler available for the SDMSCollection class. The CollectionLoadEventHandler class enables the loading of the collection to be performed by code you write. By default the SDMSCollection class loads the required records into the ArrayList collection based on the complete structure of the records being processed. If you specify the ViewType argument in the above methods you must provide a CollectionLoadEventHandler object. This object must populate the ArrayList collection as required. This enables you to customize the view of the fields within the ArrayList collection.

The "SDMSTable" Class

The SDMSTable class is designed to expose the Synergy DBMS data in the form of a .NET System.Data.DataTable. A DataTable is an in-memory representation of a database table, and comprises DataColumns and DataRows. By exposing the data in this format we can easily bind to data aware controls with the .NET Framework. The user interface within ChronoTrack is built using .NET controls. These controls are data aware and so can be bound directly to the columns defined within these DataTables.

The constructor of the SDMSCollection class accepts an SDMSio object. This SDMSio object enables direct access to the underlying Synergy DBMS data file.

The SDMSTable class provides the following methods:

- **AddColumn().** Accepts the column name, the type of the column, column caption, unique flag and allow null flag. The AddColumn() method is used to add columns to the DataTable. Each column is added to the DataTable in the order specified. This method is generally called from within the DefineColumnsEventHandler (see Event Handlers below).
- **Clear().** Clears the contents of the DataTable.
- **Commit().** Traverses through the DataTable and performs the required SDMSio operations for any inserted/changed/deleted rows within the table.
- **Cancel().** Rejects any changes to the DataTable, reverting the table back to its original state.

The SDMSTable class provides access to the underlying DataTable through the Table property.

Event Handlers

There are two required event handlers that must be written and assigned to the SDMSTable object. The base classes from which to extend your classes are:

- **DefineTableEventHandler.** Enables the definition of the columns within the DataTable. It will be called once, when the DataTable is first accessed/created. You cannot modify the structure of the DataTable once defined.
- **UpdateFieldsEventHandler.** Will be fired for every created, modified, and deleted row when the Commit() method (defined above) is executed. This event is used to move the individual column data elements from the DataRow to the Synergy record area.

Both of these event handlers are required.

The "SDMSBase" Class

The SDMSBase class implements the SDMSio, SDMSCollection, and SDMSTable classes. By implementing these classes SDMSBase enables complete management of the Synergy DBMS data either directly through SDMSio, via a collection (SDMSCollection) or as a DataTable (SDMSTable). The ChronoTrack application extends this base SDMSBase class for each data file within the system.

The SDMSBase constructor is used to assign the file name, event handlers, etc. to the object. The constructor's interface has the following signature:

- **filename.** Pass the physical name of the file you wish to manage.
- **preStore.** Pass in the pre-store object. A value of ^null is valid.
- **postStore.** Assign the post-store event object. A value of ^null is valid.
- **preUpdate.** Pass in the pre-update handler object. A value of ^null is valid.
- **updateConflict.** Assign the object that will handle update conflicts. A value of ^null is valid.
- **formatTransView.** Pass in the object that extends the FormatTransEventHandler class that handles formatting of transaction text. A value of ^null is valid.
- **onDelete.** Assign the event handler to be executed when deletes from a file are performed. A value of ^null is valid.
- **viewType.** TableView enumeration value.
- **tableName.** The name to assign to the System.Data.DataTable.
- **defineTable.** The event handler object that is used to define the columns within the DataTable. Must be a reference to an object that extends DefineTableEventHandler. A value of ^null is not valid.
- **updateFields.** The event handler object that is used to extract the fields from the DataRow. Must be a reference to an object that extends UpdateFieldsEventHandler. A value of ^null is not valid.
- **collectionName.** Name of the collection. Not used but is required.
- **collectionObject.** A STRUCTFIELD of the record being used.
- **noDotNet.** Pass true to indicate not to use any .NET capabilities.

The SDMSBase class implements the base SDMS classes detailed above. It also exposes them:

- **IO.** Exposes the underlying SDMSio object that points to the SDMS data file.
- **DataTable.** Exposes the underlying SDMSTable object.
- **DataCollection.** Exposes the underlying SDMSCollection object.

See the ChronoTrack Application section below to see how this SDMSBase class is extended and used.

Utilities

The ChronoTrack application uses a number of utility classes. These utility classes facilitate various capabilities to the application.

The "Conversion" Class

The conversion class contains a number of routines that convert between Synergy types and the DotNetObject type that is used by the gennet-created Synergy source. The following methods are available in the conversion class:

- **ToNetBoolean().** Converts the inbound Synergy Boolean value into an object.
- **ToSynBoolean().** Converts the inbound DotNetObject value to a Synergy Boolean type.
- **ToNetString().** Converts the inbound Synergy string or alpha value into an object.
- **ToSynString().** Converts the inbound DotNetObject value to a Synergy alpha type.
- **ToNetInteger().** Converts the inbound Synergy integer value into an object.
- **ToSynInteger().** Converts the inbound DotNetObject value to a Synergy integer type.
- **ToNetDecimal().** Converts the inbound Synergy implied decimal value into an object.
- **ToSynDecimal().** Converts the inbound DotNetObject value to a Synergy implied decimal type.
- **ToNetLong().** Converts the inbound Synergy Boolean value into an object.
- **ToSynLong().** Converts the inbound DotNetObject value to a Synergy decimal type.
- **ToNetDateTime().** Converts the inbound decimal reversed date value into a DataTime object. Optionally pass in the time to include in the resulting DataTime object.
- **ToSynDate().** Converts the inbound DotNetObject value to a DateTime value and return the decimal date value as a revered date.
- **ToSynTime().** Converts the inbound DotNetObject value to a DateTime value and return the decimal time value.
- **ToNetObject().** Converts the inbound Synergy alpha value into an object.
- **ToGRFA().** Enables the conversion from the passed-in RFA value and record area into a GRFA value. This GRFA value is used when managing optimistic locking.
- **ToRFA().** Extracts and returns the RFA portion of the passed in GRFA.

- **ToNetGRFA().** Converts the GRFA value to an object.
- **ToSynGRFA().** Converts the inbound DotNetObject value to a GRFA.
- **ToHashCode().**Generates the unique hash code using the passed-in record area.

The "Debugger" Class

The Debugger class is used to display debug messages interactively in the debug output window as the ChronoTrack application is running. Code is added to the application to use the methods within the debugger class to display messages to the debug window. There are a number of macros defined within the C:\Development\SynPSG\Core\src\Includes\Core.def that wrap the methods within the debugger class. For example,

```
.define debugInfoIO(msg) if (Debugger.InDebugMode == true)
      Debugger.InfoMessage("SDMSio " + %atrim(msg))
```

In the SDMSio class we have debug information imbedded throughout the code:

```
debugInfoIO("CheckExists() : key value" + keyValue + " KOR : "
+ %string(kor))
```

To activate the debug window, set the environment variable:

```
set CHRONOTRACK_DEBUG_MODE=ON
```

All debug information will then be displayed within the debug window. All members within the Debugger class are static and thus you do not create an instance of this class. The Debugger class has the following members:

- **Init().** The Init() method should be called to initialize the debugger class. You can optionally pass in a Boolean flag indicating that the application is being executed in an *xf*ServerPlus environment. When executing in this environment all messages are logged to the XFPL log file. (See the XFPL_LOG section in the *Developing Distributed Synergy Applications* manual.)
- **InfoMessage().** Displays the passed message to the debug window. It will be assigned the info icon.
- **ErrorMessage().** Accepts a message and a Synergy exception object. The message and exception details are displayed to the debug window and assigned the error icon.
- **UIMessage().** An event handler method that enables the .NET assemblies to call back into Synergy and display messages to the debug window.

The "TransactionViewer" Class

The ChronoTrack application processes changes to the data in a batch type mode. When one of the maintenance programs is run, the data in the file being maintained is loaded into a DataTable. This DataTable representation of the data is then bound to the edit controls within the .NET forms. The user can then make the required modifications, additions, and deletions from the DataTable. When the user then commits the changes the DataTable is interrogated to identify the changes made. All changes are then applied to the data in the physical file. Because of this "batch" approach, the TransactionViewer class is provided so that a log of the changes can be recorded and displayed to the user.

The TransactionViewer class has the following members:

- **ClearView().** Clears the contents of the transaction view window.
- **Text().** Adds text to the transaction view window.
- **ShowView().** Displays the transaction view window and enable the user to scroll through and review the entries listed.
- **IsVisible.** Enables the user to decide if he wishes to see the transaction viewer. Setting the false will not show the window when the ShowView() method is called.

The "WindowManager" Class

The WindowManager class has been designed to wrap some UI Toolkit routines with simple methods. The WindowManager class contains the following members:

- **WindowManager().** Class constructor. Accepts the UI Toolkit window library and the application title. The constructor initializes the UI Toolkit and applies the title string to the application window drag bar.
- **Footer().** Sets the application footer bar text.
- **ClearFooter().** Clears the application footer text.
- **Environment().** Enters or exits an environment.
- **TKWindow().** Assigns a .NET form to a UI Toolkit window and return the window ID. The method accepts the .NET form, required window size, window title, and if the window should be centered within the containing application window. Passing screen size values as zero will cause the window to be maximized to fill the application window.
- **LoadMenu().** Loads the required menu from the window library and place on the menu bar.
- **DisableMenu().** Disables the identified menu.
- **HideMenu().** Removes the menu bar from the application window.

- **PlaceForm().** Shows the identified form/window.
- **ProcessForm().** Processes the form.
- **HideWindow().** Removes the identified form from the application window.
- **SynInteropInstance.** This property is the instance of the SynPSGNetCore.Interop.SynInterop class.
- **FormResize.** This property returns the System.Drawing.Size object that enables you to correctly size the .NET form after it has been assigned to a UI Toolkit window.
- **ScreenRows.** Enables the program to force the screen size rows.
- **ScreenCols.** Sets the application screen column value.

The Working class is designed to simply place a working message into the application's footer bar to indicate that the program is processing and is busy. The class has a single static method called **ShowForm()**. It accepts a single argument which is the text to display in the footer section.

SynPSGNetCore

The SynPSGNetCore assembly is a collection of classes that assist with the interoperability between .NET and Synergy. The assembly is written in C# and is developed with Visual Studio. The SynPSGCNetCore solution contains the following:

- **Forms.TransactionView.cs.** This form is used by the TransactionViewer above.
- **Forms.Working.cs.** This is a redundant form. It was originally used to display a working message and included an animated image. The Working class above no longer uses this form as it places the working message in the footer bar of the application window.
- **Interop.SynInterop.cs.** This class enables the .NET forms and the Synergy Language code to handle UI Toolkit style menu processing. See the "How do I signal a menu entry from within a .NET form?" section below for details on how this class is used.
- **Logging.DebugOutput.cs.** This form is the debug output window used by the Debugger class above.

To build the SynPSGNetCore assembly, see the Building ChronoTrack section below.

The "ChronoTrack" Folder

The ChronoTrack folder and its subfolders contain the main application's classes and routines, and the various .NET assemblies containing UI components, the Web site, mobile solution, and the dashboard.

The structure of the ChronoTrack folder is:

Folder Name	Description
📂 Cat	Contains the *xf*ODBC catalog files and the connect files. It also contains the SQL script required to add the required views for the ChronoTrack Dashboard application.
📂 Dat	Contains the application's DBMS data files.
📂 Def	Contains various application definition header files.
📂 Exe	Contains all the application executable libraries and executable programs. Also contains the Synergy/DE UI Toolkit window library.
📂 hdr	Contains the header files resulting from the prototyping of the application classes and routines.
📂 inc	Contains the ChronoTrack include file.
📂 lib	Contains the object libraries. These object libraries are not referenced by the ChronoTrack application; they are simply used to generate executable libraries.
📂 obj	Contains the child folders where the compiled object files are stored.(detailed below)
📂 →Applications	Contains the object files created when the application routines and main line programs are compiled.

📂 →DataEntities	Contains the DataEntity object files.
📂 →Server	Contains the object files of the server routines that enable access via *xf*ServerPlus.
📂 →SqlApi	Contains the object files for the various SQL routines that enable replication from the Synergy DBMS data to the chosen relational database.
📂 →Subroutines	Contains the object files for the core application subroutines implemented by ChronoTrack.
📂 →Toolkit	This is an empty place holder folder.
📂 Prj	Contains the Synergy/DE Workbench project files. Do not open these project files individually; instead, open the ChronoTrack workspace file in Workbench. This will ensure that all the relevant projects are accessible. This workspace can be found in the C:\Development\SynPSG\ChronoTrack folder.
📂 rps	Contains the application repository files. Includes scripts to rebuild the repository from the schema file.
📂 rpssch	Contains stub schema files for the various structures in the repository. It is used by the PSG Build subsystem and is not required by the distributed ChronoTrack development environment.
📂 smc	Contains the Synergy Method Catalog files. These are maintained by the Method Definition Utility.
📂 src	Parent folder that contains the child folders that store the source code files.

🗁 →Applications	Contains the main application routines and main line programs. This folder contains the main application functionality demonstrated at SPC 2009. These routines manipulate the UI components defined in the various .NET assemblies. Contains the build command scripts.
🗁 →Assembly	Contains the XML file that defines the methods we are exposing through the SMC.
🗁 →→ChronoTrack	Contains the generated code defined by the routines in the SMC. This is the main folder for the Synergy .NET assembly project. It also contains the build assembly that we reference within our Visual Studio Web and Web service projects.
🗁 →DataEntities	Contains the application DataEntities. Each Synergy DBMS data file within the ChronoTrack environment has its own DataEntity class that extends the base SDMSBase class. This folder also contains all the individual event handler and event argument classes for the required event handlers that have been implemented. All these event handler and argument classes extend the base classes defined in the Core development area. Also contains the build command scripts.
🗁 →Server	Contains the routines that ChronoTrack uses to expose logic and data to external applications like the ChronoTrack Web site (ChronoWebDX) and the ChronoTrack Web service (ChronoTrackWS).
🗁 SqlApi	Contains the code-generated routines to enable the replication server to replicate the

	application DBMS data to the chosen relational database. The folder also contains the SQL replication server source file. Build command scripts are also located here.
🗁 Subroutines	Contains the source files for the generic ChronoTrack application routines to manage application settings and validating user login details.
🗁 VSSolutions	Contains the .NET assemblies.
🗁 →ChronoMobile	Contains the Visual Studio solution and all associated files to build the ChronoTrack mobile application. See the ChronoMobile section below.
🗁 ChronoTrackMonitor	Contains the Visual Studio solution and all associated files to build the ChronoTrack monitor application. See the ChronoTrackMonitor section below.
🗁 ChronoTrackNetUI	Contains the Visual Studio solution and all associated files to build the Infragistics version of the ChronoTrack .NET user interface assembly. See the ChronoTrackNetUI section below.
🗁 ChronoTrackNetUICommon	Contains the Visual Studio solution and all associated files to build the common user interface assembly. See the ChronoTrackNetUICommon section below.
🗁 ChronoTrackNetUIDX	Contains the Visual Studio solution and all associated files to build the DevExpress version of the ChronoTrack .NET user interface assembly. See the ChronoTrackNetUIDX section below.

📂 ChronoTrackNetUIMS	Contains the Visual Studio solution and all associated files to build the Microsoft version of the ChronoTrack .NET user interface assembly. See the ChronoTrackNetUIMS section below.
📂 ChronoTrackReports	Contains the Visual Studio solution and all associated files to build the sample reports. These reports are hosted within the Microsoft SQL Server Reporting Services and can be viewed through the ChronoTrack application of the ChronoWebDX sample Web site.
📂 ChronoTrackWS	Contains the Visual Studio solution and all associated files to build the ChronoTrack Web services Web site. See the ChronoTrackWS section below.
📂 ChronoWebDX	Contains the Visual Studio solution and all associated files to build the ChronoTrack Web site. See the ChronoWebDX section below.
📂 LicensingDashboard	Contains the Visual Studio solution and all associated files to build the example Licensing dashboard applications. See the LicensingDashboard section below.
📂 ODBCDashboard	Contains the Visual Studio solution and all associated files to build the example ODBC dashboard applications. See the ODBCDashboard section below.

Applications

This folder contains the mainline programs and main subroutines that make up the ChronoTrack application. The folder also contains the required build script "makeit.bat". These are the main files in this folder:

- **ChronoTrack.dbl** is the mainline program. The ChronoTrack program uses all of the available assemblies to present the user interface. For example, the login form is loaded from the ChronoTrackNetUIDX assembly, which utilizes the DevExpress suite of UI controls. The menu system is from the same assembly. The project management routines use the forms within the ChronoTrackNetUI assembly, which is built around the Infragistics control set. Some of the standard maintenance routines like State Maintenance load their forms from the ChronoTrackNetUIMS assembly, which uses standard Microsoft controls shipped with Visual Studio.
- **The xfpl_log.dbl** source file is a dummy routine that enables ChronoTrack to build. When running under *xf*ServerPlus, the real xfpl_log routine defined in XFPL.DBR will be used. This enables the Debugger class to log messages to the xfpl log file. This version of the routine is never executed and is only made available here so that ChronoTrack can be linked.

DataEntities

The ChronoTrack application utilizes the base DBMS classes defined in the Core area. All data file access is performed through these DataEntity classes. The folder also contains the "makeit.bat" build command script.

Each DataEntity class is named based on the file being accessed. For example, the state.ism file is accessed by using the StateDataEntity.dbc class. This class defines all that is required to access the state.ism data file. In addition, it also demonstrates how to implement the event handlers. The StatePreStore.dbc class file defines the event handler for the pre-store event. When a record is about to be stored in the file, any registered pre-store event handler will be fired. This enables additional processing prior to the creation of the record. The event is called by the SDMSio base class. In the StatePreStore.dbc file we have the following code:

```
public override method EventMethod   ,void
    in       req sender ,@SynPSG.Core.DataEntities.SDMSio
    inout    req e ,@SynPSG.Core.DataEntities.PreStoreEventArgs
    endparams

.include "state" repository, record = "state" ,end

proc
    state = e.NewData

    ;;data integrity validations
    upcase state.Code

    ;;set unique key for SQL replication
    state.replication_key = %datetime()

    e.NewData = state

    mreturn

endmethod
```

As you can see, the code ensures that the state code is uppercased and that the replication key value is correctly populated.

Singleton Pattern

The singleton pattern is designed to ensure a class only has one instance, and provide a global point of access to it. It also prevents the need to instantiate the class, because the class instantiates itself. To enable this inside each DataEntity class we define a static instance of it. For example, in the StateDataEntity class we have:

```
private static mInternalData    ,@StateDataEntity
```

We then expose a static instance of the call by means of a static property:

```
public static property Instance ,@StateDataEntity
    method get
    proc
        if (mInternalData == ^null)
            mInternalData = new StateDataEntity(true, TableView.Full)
        mreturn mInternalData
    endmethod
endproperty
```

Within the property we check the static mInternalData member to ensure we have an instance of the class by checking if the instance is not null, and creating an instance of the class if it is null (default state). We then return this instance.

Server

The Server folder contains all the required code to expose the ChronoTrack logic and data to the *xf*ServerPlus/*xf*NetLink .NET clients. Each source file contains the related routines for a data file. For example, the StateRotuines.dbl file contains several functions that expose the various elements of the state data. The “get_state” routine accepts a state ID and returns the state record area containing the located state details. The function returns a Boolean status to indicate the success/failure of the call. A true return value indicates a successful operation.

To locate a single state record, the remote client makes a call to the “get_state” function. Inside this function the code makes a call to the underlying DataEntity:

```
result = StateDataEntity.Instance.IO.Locate(stateID,Q_PRIMARY,false)
```

Notice the use of the actual class and not an instance of it. This is making use of the singleton pattern detailed above. We then return the state data record:

```
stateData = StateDataEntity.Instance.IO.DataArea
```

See the StateRoutines.dbl source file for a complete listing.

SqlApi

Built into the SDMSio class is the ability to replicate the changes to the underlying Synergy DBMS data to a relational database. This is achieved by recording every file modification into a central replication control file. This replication control file is then interrogated by the replication server. Each record in the control file identifies the actual data file containing the modified data, the type of modification made (created, modified, deleted), and the unique key to the record in the data file. The replication server then utilizes file-based routines to execute the required SQL commands to replicate the data to the relational database.

All of the file-based routines are code-generated and the Synergex PSG team can assist you with generating these source files for your data files.

For example, the SQL commands required to manage data within the state table stored within the relational database are all contained within the state_sqlio.dbl source file. The replication server is called ChronoTrackReplicator.dbl. This program interrogates the control file and uses the XSUBR() routine to call the required file based SQL routines.

Data Source Name

The ChronoTrack replicator is hard coded to use a DSN called “ChronoTrack” to connect to the SQL Server relational database. It also defines the username as “ChronoTrack” with a password of “ChronoTrack”. You must configure the DSN correctly on your machine. Your DSN should have the following properties:

- **Name.** ChronoTrack
- **Description.** ChronoTrack Replicator
- **SQL Server to connect to.** This should be the instance of SQL running on your machine.
- **Select to authenticate with SQL Server authentication.** Enter a username of ChronoTrack and a password of ChronoTrack.
- **Accept all other defaults.**

These settings will enable the replicator to connect to the required database. The ChronoTrack replicator, on Windows, is run as a service. To install the service, you can execute the C:\Development\SynPSG\ChronoTrack\exe\replication_server_create.bat command script. This will create the server process. To start the replicator, use the C:\Development\SynPSG\ChronoTrack\exe\replication_server_start.bat command script. When this is running, any changes to your ChronoTrack data will be replicated to the SQL Server database.

ChronoTrackNetUI

This folder contains the Visual Studio solution for the Infragistics version of the user interface assembly. The Form subfolder contains all of the WinForms used by ChronoTrack. Open the solution within Visual Studio to view the ChronoTrackNetUI project files.

ChronoTrackNetUIDX

This folder contains the Visual Studio solution for the DevExpress version of the user interface assembly. The Form subfolder contains all the WinForms used by ChronoTrack. Open the solution within Visual Studio to view the ChronoTrackNetUIDX project files.

ChronoTrackNetUIMS

This folder contains the Visual Studio solution for the Microsoft version of the user interface assembly. The Form subfolder contains all the WinForms used by ChronoTrack. Open the solution within Visual Studio to view the ChronoTrackNetUIMS project files.

ChronoTrackNetUICommon

This assembly is utilized by all three of the UI assemblies detailed above. It is designed to provide a common interface when signaling back to the Synergy code that a WinForm has been completed. The majority of the WinForms in the UI assemblies have “Save” and “Cancel” buttons. The assembly contains the following methods.

- **Handles_Save().** Called when the Save button is clicked. This method validates that all the forms controls all have valid data, and if they do, it signals a UI Toolkit menu entry of “MB_COMMT”.
- **Handles_Cancel().** Assigned to the Cancel button click event. Cancels the edit within the binding source and signals an “MB_CANCEL” UI Toolkit menu entry.

ChronoMobile

Within this folder, you’ll find the Visual Studio solution for developing the Windows Mobile Pocket PC application. This solution relies on the Windows Mobile 5.0 SDK R2 for Pocket PC, although it should also work with the Windows Mobile 6.0 SDK (not tested).

The ChronoMobile solution and application connects to the ChronoTrackWS (Web service) application (detailed below), so it is also necessary to have the Web service installed and running either on the local machine or on a network-accessible PC. Open the solution in Visual Studio to view or work with the ChronoMobile project files.

ChronoTrackDashboards

There are two dashboards distributed with the ChronoTrack application: a dashboard to show information from the ChronoTrack projects (ODBCdashboard) and a dashboard to show licensing information (LicensingDashboard). Both dashboards use DevExpress controls to display data.

ODBCdashboard

This folder contains the Visual Studio solution for the ODBCdashboard. Open the solution within Visual Studio to view the ODBCdashboard project files.

LicensingDashboard

This folder contains the Visual Studio solution for the LicensingDashboard. Open the solution within Visual Studio to view the LicensingDashboard project files.

ChronoTrackMonitor

The ChronoTrack Monitor is a simple C# program that utilizes the ChronoTrackWS Web service and presents the projects assigned to the logged-in user in the form of a DevExpress carousel grid. The monitor program displays an icon on the system tray. Right-click the icon to open the context menu, and select the required option.

ChronoTrackReports

This folder contains the source code for the various SQL Server Reporting Services reports that are used by several of the ChronoTrack client applications.

ChronoTrack uses SQL Server 2008 Reporting Services, so in order to work with the source code for the ChronoTrack reports, you must have SQL Server 2008, including the "Reporting Services" and "Business Intelligence Development Studio" options.

The ChronoTrack reports execute against the replicated copy of the ChronoTrack data. So in order to execute the reports, you must already have configured data replication.

Use "SQL Server Business Intelligence Development Studio" to open the main solution file which is called ChronoTrackReports.sln.

In the solution you will find a data source file called ChronoTrackSqlData.rds. This file "points" the reports to the replicated data. The file is pre-configured to point to a SQL Server database called "ChronoTrack" on the local system. If your replication database is elsewhere, you can double-click on the data source file to specify the name and location of the database to use.

The solution also contains several example reports (each has a .rdl file extension). To view the designer for a report, simply double-click on the report file.

The ChronoTrack client applications that consume these various reports expect the reports to be deployed in a folder called ChronoTrackReports, on a default instance of the SQL Server 2008 Report Server on the local machine (http://localhost/reportserver). To deploy the reports to your report server, select "Build > Deploy ChronoTrackReports" from the menu.

If you wish to deploy the reports to a different report server, select "Project > Properties" from the menu, then change the TargetServerURL and TargetReportFolder settings. Also modify any client application that displays the reports to use the new report server location.

ChronoTrackWS

ChronoTrack logic and data is exposed through *xf*ServerPlus and the *xf*NetLink .NET client. The ChronoTrackWS Web service provides access to the exposed logic and data. The main class that extends the System.Web.Services.WebService class is called RemoteService. Within this class the following Web methods are available:

- **ValidateUser().** Accepts the username and password and validates the user credentials against the ChronoTrack database.
- **UserSchedule().** Returns information for the currently logged-in user.
- **Projects().** Returns a DataTable of projects assigned to the currently logged-in user.
- **ProjectSummary().** Returns the project summary details for the assigned user.
- **CreateProjectNote().** Accepts a project ID, description text, and note text and creates a note in the ChronoTrack database for the project.
- **ProjectNotes().** Returns the notes associated with the identified project ID.
- **Disconnect().** Disconnects the client from the Web service. Logs out the user.

The ChronoTrackWS Web service is used by the ChronoTrackMonitor and the ChronoMobile applications.

ChronoWebDX

The ChronoWebDX folder contains the Visual Studio solution for the ChronoTrack Web client. The Web client uses various UI controls from DevExpress, so in order to work with the Web client you must have DevExpress installed. You also need to have the latest version of Synergy/DE *xf*NetLink .NET installed.

The main solution file for the Web client project is called ChronoWebDX.sln. This file is located in the parent folder (VSSolutions) to prevent the solution file from being deployed to a Web server when the client solution is published. Use Visual Studio 2008 to open this solution file.

The Web client consists of a large number of Web pages, user controls, themes, etc.--far too many to mention in detail here. The main components of the Web client are:

- The **App_Code** folder.
  This special folder contains various utility classes that are used by various pages and controls throughout the Web site. There is a "BLL" (Business Logic Layer) class for each of the main data structures, and these classes contain all of the business logic which calls *xf*NetLink .NET methods in order to interact with the ChronoTrack Server environment (*xf*ServerPlus). These BLL classes are used in conjunction with various UI controls, which "bind" to these classes via the ASP.NET's "ObjectDataSource" control. In some cases there is code in various pages and controls which also calls the various methods that these classes expose.

  Also of note in this folder is the SynUtils class, which contains various "utility" methods that are used throughout the site, the SynLct class which contains wrapper functions for the Synergy Licensing Toolkit functions, and the ThemedPage class which provides the foundation for applying consistent UI themes to pages throughout the site.

- The **App_Themes** folder.
  Contains the various files (images, style sheets, etc.) which are used to apply themes to pages throughout the site. These files are added to the Web application using the DevExpress ASPxThemeDeployer utility.

- The **Controls** folder.
  The Web client makes use of various "user controls" in order to define frequently used code in one place for reuse, or in order to simplify the development of complex pages. This folder contains these various user controls.

- The **Scripts** folder.
  This folder contains a JavaScript source file which contains various functions that are used when applying themes to pages.

- Various other folders.
  The actual Web pages that make up the application are organized by function in various other subfolders (admin, consultant, customer, project, reports, tools, etc.). ASP.NET Web pages all have an .aspx file extension.

- The **LoggedInMaster.master** master page.
  This is the master page that defines the overall layout of most pages in the Web site. Most of the pages in the site inherit from this master page.

- The **Web.sitemap** file.
  This special XML file defines the menu system structure for the entire site. The file defines which menu columns are displayed once a user has logged in and which Web pages are displayed when the user selects options from the menu.

- The **Default.aspx** page.
  This is the "home page" for the Web client application. The home page contains the login form, and the code to validate user logins.

To build the Web client application, select "Build > Rebuild Web Site" from the menu.

Before you can run the Web client you must have *xf*ServerPlus configured. (See Configuring *xf*ServerPlus section for more details.)

Various pages in the Web application display SQL Reporting Services reports. To use these pages you must have deployed the ChronoTrackReports project to your SQL Report Server. (See ChronoTrackReports.)

To run the Web client, select "Debug > Start without Debugging" from the menu. This will run the Web client using Visual Studio's built-in development Web server (Cassini).

If you wish to deploy the Web client application to an IIS server, install *xf*NetLink .NET on that server. Also deploy runtime-only copies of all of the DevExpress ASP.NET assemblies to the server. (Refer to the DevExpress documentation.)

Building ChronoTrack

To successfully build the ChronoTrack application you will need the following:

- Windows XP SP2 or higher
- The ChronoTrack zip file available from the Synergex Code Exchange.
    - Note, file names are case-sensitive.
- Synergy/DE 9.1.5b or a higher version
- Infragistics NetAdvanatage for .NET 2009.1
    - With this required hotfix: NetAdvantage_20091_CLR2X_WIN_SR_2023
- DevExpress DXperience 2009 volume 1
    - Version should be 9.1.5.
- Visual Studio 2008
    - We recommend that you install all available Microsoft service patches.
- Microsoft Internet Information Server installed and running
    - Required for the ChronoWebDX Web site and the ChronoTrackWS Web service
- Microsoft SQL Server
    - Required for the replication process
        - See the Data Source Name section for details on how to configure the server connection.
    - Provides the SQL reporting

We recommend that you install the required products to their default locations as using alternative locations may cause the build scripts to fail.

Once the required pre-requisites are installed, you can rebuild the ChronoTrack environment by using the MakeChronoTrack.bat command script. Follow these steps:

1. Open a command prompt from the start menu.
2. Change the directory to the C:\Development\SynPSG folder.
3. Execute the MakeChronoTrack.bat command script. The command script accepts the following parameters:
    - Configuration
        - Debug – builds the debug version
        - Released – builds the released version
    - Environment
        - IN – builds the Infragistics version
        - DX – builds the DevExpress version

        - MS – builds the Microsoft version
        - TK – builds a Synergy/DE-only version using UI Toolkit windows
            - There is very limited functionality available in this version.
        - ALL – Builds a version of ChronoTrack that demonstrates the full capabilities of all the available controls
            - We recommend this build as it is the most functionally complete.
    - Wait
        - NO – will not pause at the end of the build process
        - YES (default) –will pause at the end of the build process

We recommend that you pipe the output from the MakeChronoTrack.bat command script to a text file. You can then review the text file to identify any build issues. For example:

```
MakeChronoTrack Debug ALL NO > build.txt
```

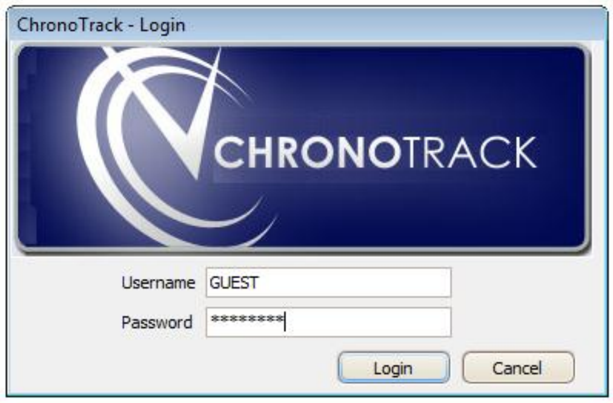
You can then view the build.txt log file. If you are experiencing build issues and would like assistance from Synergy/DE Developer Support, they will need to see this build.txt log file to effectively assist you.

Executing the ChronoTrack Windows Application

The easiest way to execute the ChronoTrack Windows application is to perform the following tasks:

1. Open a command window from the Start menu.
2. Change the directory to the C:\Development\SynPSG folder.
3. Execute the setenv.bat command script.
    - This sets up the required environment variables.
4. Execute the application.
    - dbr EXE:ChronoTrack

You are initially prompted for account details:



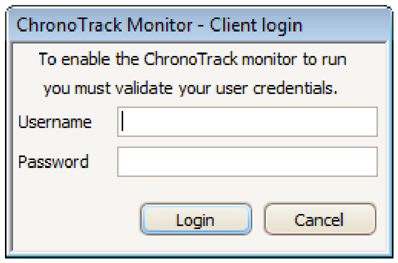
Username: GUEST
Password: ********

Use the username of “GUEST” with a password of “p@ssw0rd”.

You now have access to all elements of ChronoTrack.

Executing the ChronoTrack Monitor

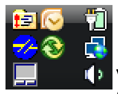
The ChronoTrack monitor is a simple C# application that runs in the system tray. When you execute the ChronoTrack monitor for the first time, you will be asked for your user credentials:

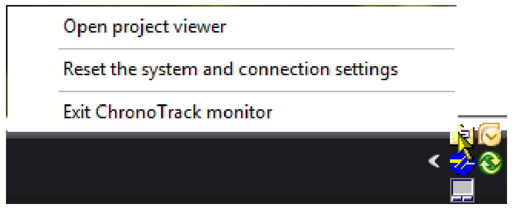


To enable the ChronoTrack monitor to run you must validate your user credentials.

Username:
Password:

Use the same “GUEST/p@ssw0rd” login, unless you have created a new user within the ChronoTrack database.

When executing, you will see a small icon in the system tray (). If you have created some projects using ChronoTrack (Windows/Web) and assigned them to you, the ChronoTrack monitor will display a notification balloon. To open the ChronoTrack monitor to view assigned projects, right-click to display the context menu:



The ChronoTrack monitor uses the ChronoTrackWS Web service to access the ChronoTrack database. Ensure that the ChronoTrackWS Web service is running on your machine. The ChronoTrack monitor expects to find the Web service at http://localhost/ChronoTrackWS/RemoteService.asmx.

Executing the ChronoTrackWS Web Service

The ChronoTrackWS is a Web service running in the Microsoft IIS environment. The MakeChronoTrack.bat command script will build the Web service, but it is not possible to publish it. The Web service needs to be published and running before the ChronoTrackMonitor and the ChronoMobile applications will run.

To publish the ChronoTrackWS Web service, complete the following steps:

1. Run Microsoft Visual Studio.
2. Locate the C:\Development\SynPSG\ChronoTrack\VSSolutions\ChronoTrackWS\ChronoTrackWS.sln solution file.
    - This will load the Web service solution.
3. Build the solution (Build→Build Solution F6).
4. Publish the Web service (Build→Publish ChronoTrackWS).
    - Select the target location http://localhost/ChronoTrackWS/.
    - Click the Publish button.

The ChronoTrackWS Web service will now be available for the ChronoTrack monitor and ChronoMobile. Once these steps have been completed, you will not be required to complete them again unless you upgrade to a new version of ChronoTrack.

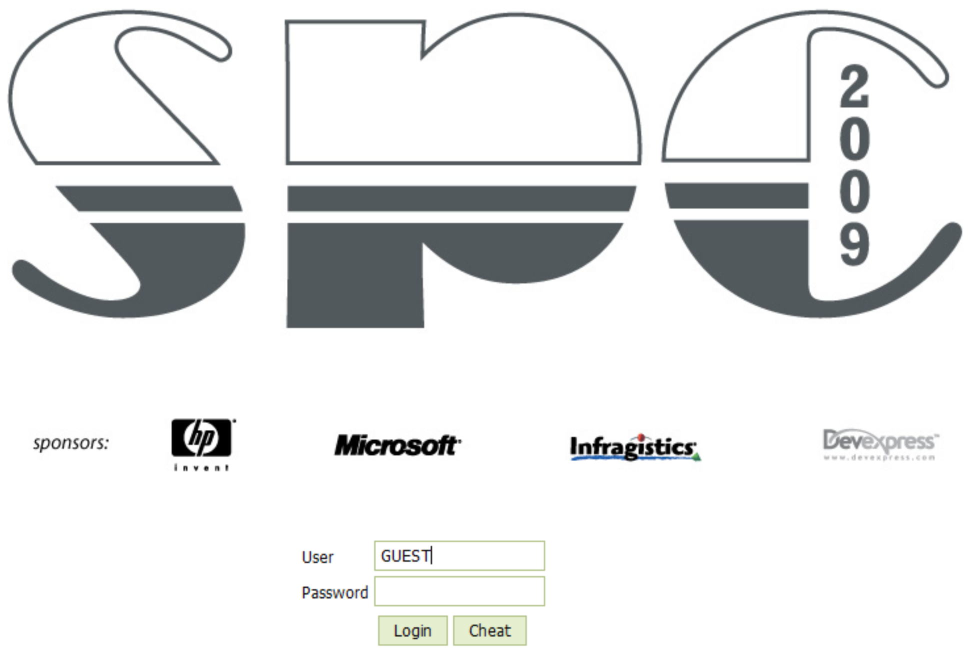
Executing the ChronoWebDX Web Site

The ChronoTrack Web site provides the ability to navigate through the ChronoTrack database and examine projects, etc. You can also manage much of the data within the application from here. And you can configure the look and feel of the Web site.

To execute the ChronoTrack ChronoWebDX Web site, complete the following steps:

1. Run Microsoft Visual Studio.
2. Locate the C:\Development\SynPSG\ChronoTrack\VSSolutions\ChronoWebDX.sln solution file.
   - This will load the Web site solution.
3. Build the solution (Build→Build Solution F6).
4. Publish the Web service (Build→Publish ChronoWebDX).
   - Select the target location http://localhost/ChronoWebDX.

The Web site will now be active. You can navigate to the ChronoTrack Web site by entering the following URL into your chosen browser: http://localhost/ChronoWebDX. When the Web site is successfully loaded, you will see the login page:



User GUEST
Password

Use the GUEST/p@ssw0rd login, unless you have created an alternate user through ChronoTrack.

Executing the ChronoMobile Mobile Device Solution

The ChronoMobile application requires several different software elements before it can be executed. These include:

- ChronoTrackWS Web service
- Windows Mobile 5.0 SDK
- Microsoft .NET Compact Framework 2.0 SP2
- Microsoft Virtual PC 2007 (possible requirement – see below)

ChronoMobile utilizes the ChronoTrackWS Web service, so it is important that the steps for Executing the ChronoTrackWS Web Service have been successfully followed prior to testing and/or launching this project.

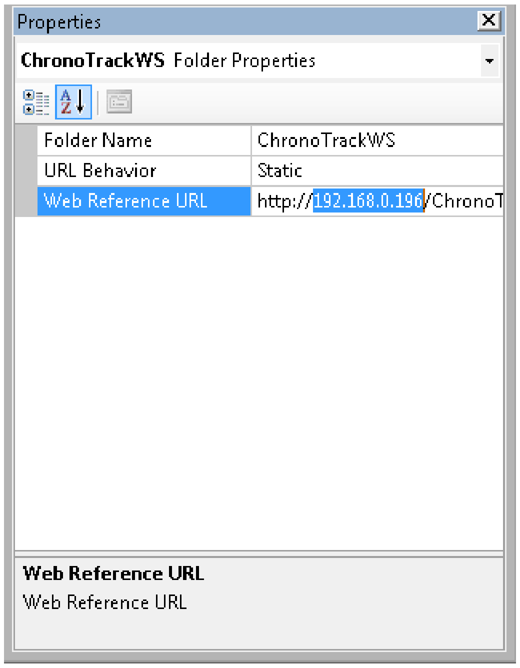
Once the ChronoTrackWS Web service has been published, make note of the network IP address on which it runs. To test it, try typing the following address into Internet Explorer:

http://*ipAddress*/ChronoTrackWS/RemoteService.asmx

where *ipAddress* is the location of the published Web service. You should see a page that details the different services available for RemoteService.

Designating the location of the ChronoTrackWS Web service

Before you can run the ChronoMobile application, you'll need to modify the project's referenced location of the ChronoTrackWS Web service. Open the ChronoMobile solution, and make sure the "Solution Explorer" pane is visible (usually on the righthand side of the window). Under "Web References," right-click **ChronoTrackWS** and select **Properties**.



Folder Name	ChronoTrackWS
URL Behavior	Static
Web Reference URL	http://192.168.0.196/ChronoT

Web Reference URL
Web Reference URL

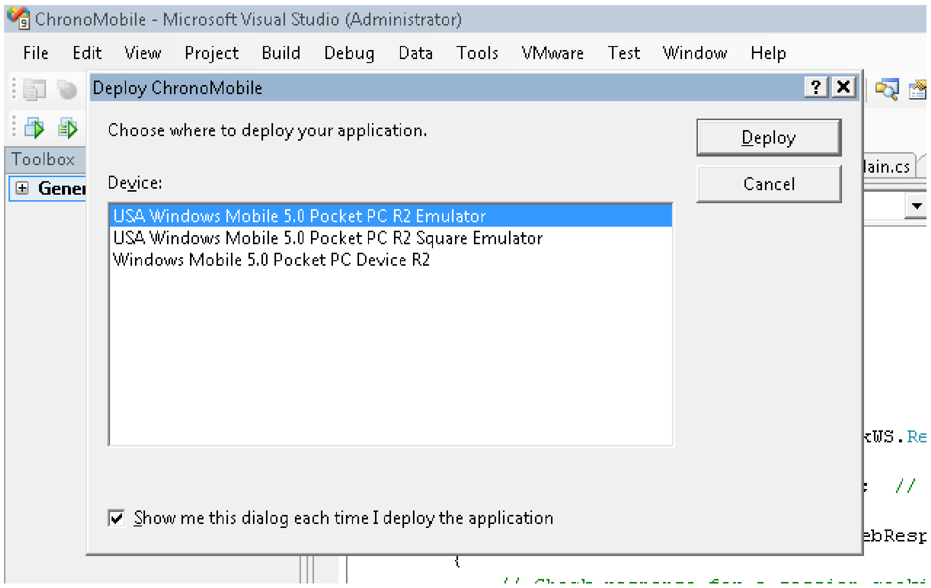
You will see an entry named "Web Service URL" which points to a local IP address and the path of the RemoteService.asmx file. Since the application will be deployed to an emulator – which has its own localhost entry – you'll need to modify this entry's value with the actual IP address of the machine on which the ChronoTrackWS Web service has been published. The property value will be exactly as it was for testing the Web service's availability from

Internet Explorer in the last step. (If you've published the Web service on the same machine with which you are developing, this will be the local IP address of the development PC.)

Configuring the Emulator

In order for the Mobile emulator to connect to the Web service, it will need network connectivity. In general, Microsoft Virtual PC 2007 is used by the Mobile emulator to connect to and share the local network connection. It is possible that other software already installed on a development machine will make this unnecessary. Therefore, it is recommended that the Mobile emulator be launched and configured for network & Internet access prior to testing ChronoMobile. This can be accomplished immediately upon opening the ChronoMobile solution by simply debugging the solution:

1. Press **F5** to begin debugging.
2. Select **USA Windows Mobile 5.0 Pocket PC R2 Emulator** for the deployment location.
3. Click **Deploy**.


ChronoMobile - Microsoft Visual Studio (Administrator)

Deploy ChronoMobile

Choose where to deploy your application.

Device:

USA Windows Mobile 5.0 Pocket PC R2 Emulator
USA Windows Mobile 5.0 Pocket PC R2 Square Emulator
Windows Mobile 5.0 Pocket PC Device R2

☑ Show me this dialog each time I deploy the application

4. Once the emulator has opened, locate the File menu and select the **Configure…** entry. Note that this menu is located on the main emulator *application* window, and not on the mobile emulator itself.
5. In the Emulator Properties window, select the Network tab.
6. Ensure that **Enable NE2000 PCMCIA network adapter and bind to** is checked, and select the appropriate device from the drop-down. If you encounter an error upon enabling the NE2000 PCMCIA network adapter, then it is likely you will

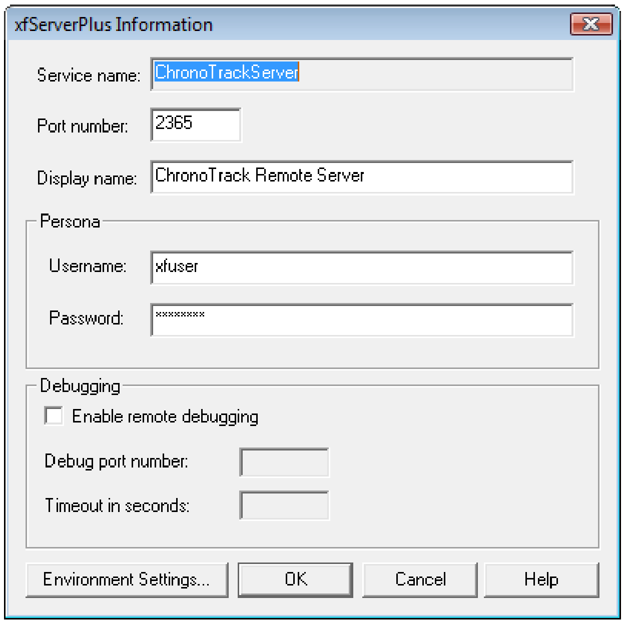
need to install Microsoft Virtual PC 2007, which is available for free download from the Microsoft Web site.

7. After the adapter is enabled, test for Internet connectivity on the emulator. Close the ChronoMobile application by clicking **Exit**, and then try browsing the Internet using the emulator's Internet Explorer. If all is working, then verify that the ChronoTrackWS Web service is also reachable by typing in its address as detailed above. Note that "localhost" will not work in this instance.
8. Close the emulator and save its settings when prompted. You may now try debugging the application again (press F5), and you should be able to login to ChronoMobile using the standard GUEST/p@ssw0rd credentials.

Configuring *xf*ServerPlus

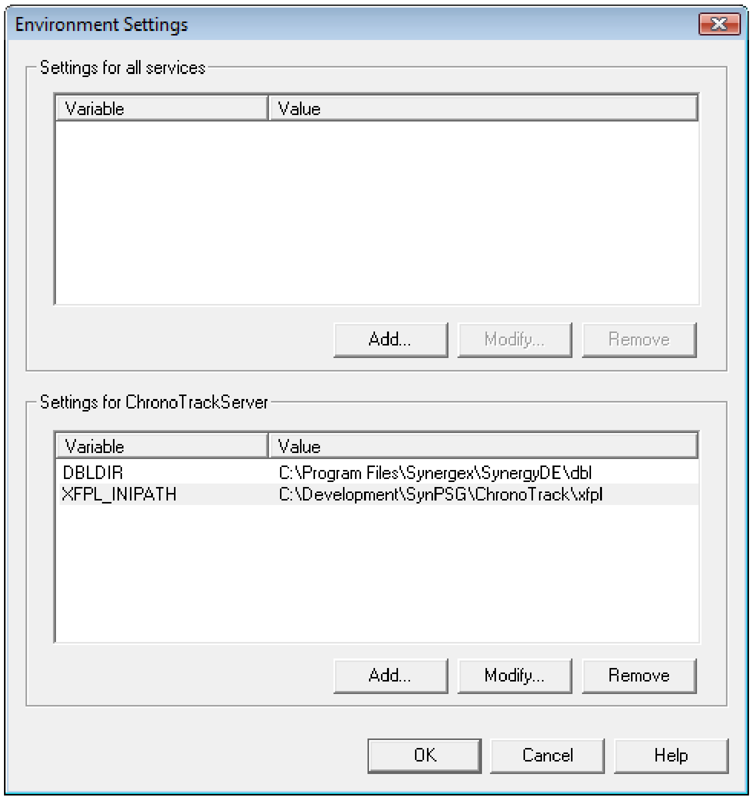
The ChronoTrack Web site (ChronoWebDX) and the ChronoTrack Web service (ChronoTrackWS), which is used by the ChronoTrack monitor and the ChronoMobile applications, must be configured correctly. All *xf*ServerPlus access is coded to port number 2365 on your local machine (localhost). To enable the Web site and Web service to function, configure an *xf*ServerPlus connection:

1. From the SynergyDE→Utilities Start menu select the Synergy Configuration Program.
2. Click on the *xf*Server/*xf*ServerPlus tab.
3. Click on the Add *xf*ServerPlus Service… button:



Service name: ChronoTrackServer
Port number: 2365
Display name: ChronoTrack Remote Server
Username: xfuser
Password: ********
☐ Enable remote debugging
Debug port number:
Timeout in seconds:

4. Ensure that the port number is 2365.
5. There is a single environment setting required. Click the Environment Settings… button:



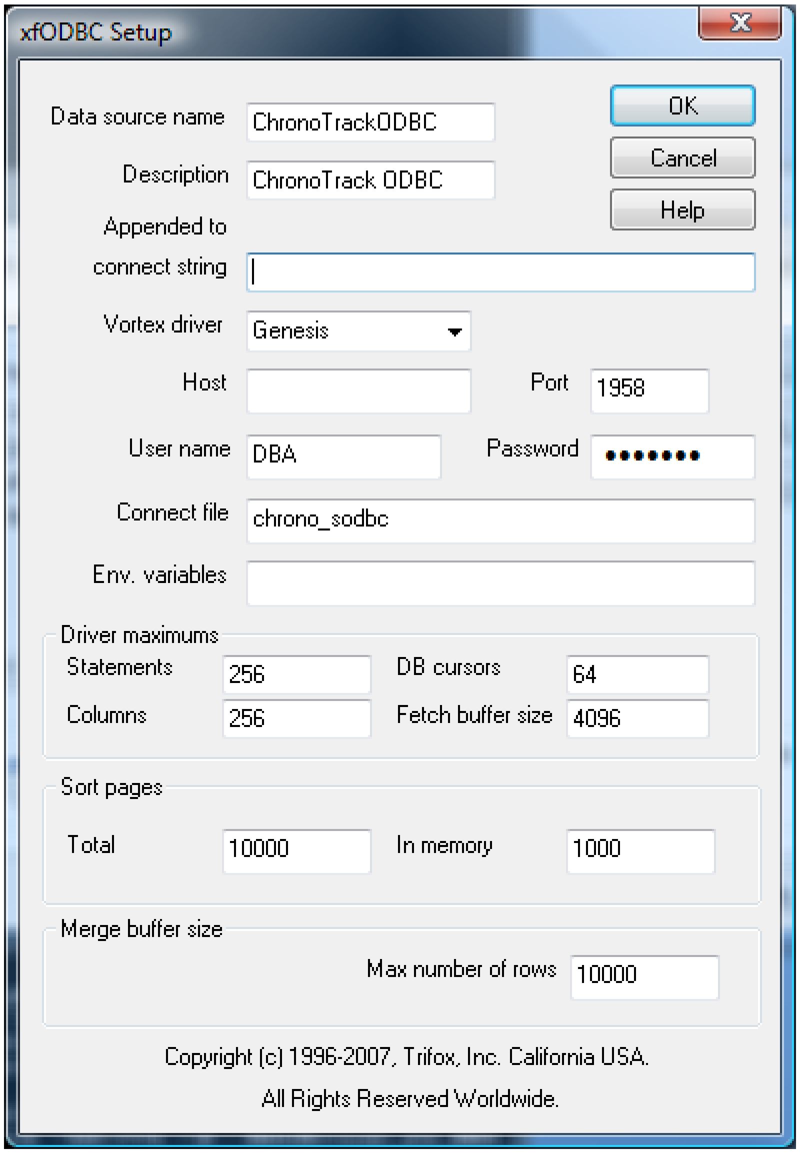
Variable	Value
DBLDIR	C:\Program Files\Synergex\SynergyDE\dbl
XFPL_INIPATH	C:\Development\SynPSG\ChronoTrack\xfpl

6. Add XFPL_INIPATH with a value of C:\Development\SynPSG\ChronoTrack\xfpl.

Configuring *xf*ODBC and the ChronoTrack Dashboard

The dashboards use *xf*ODBC as the mechanism to retrieve data, and they use the system DSN called ChronoTrackODBC. The connect file used by the DSN is called chrono_sodbc, and it is distributed in the C:\Development\SynPSG\ChronoTrack\cat folder. Copy this file to the %GENESIS_HOME% folder on your system. To set up the *xf*ODBC system catalogs, run the batch file C:\Development\SynPSG\ChronoTrack\cat\makeCatalog.bat. This batch file will create the system catalogs in the C:\Development\SynPSG\ChronoTrack\cat folder, and it will copy the connect file to %GENESIS_HOME%. Note: Microsoft Windows may not actually perform the copy command from the batch file, because of UACC or other security issues. You may have to copy this file manually to overcome this issue.
The DSN (as documented) is using the default DBA/MANAGER account. You can use any user account that has write access to the ChronoTrack data files. By default, the DBA

account does not have write access, so you may have to run the *xf*ODBC DBA utility (xfdba.dbr) to modify the DBA account.



xfODBC Setup

Data source name	ChronoTrackODBC
Description	ChronoTrack ODBC
Appended to connect string	
Vortex driver	Genesis
Host	
Port	1958
User name	DBA
Password	●●●●●●●
Connect file	chrono_sodbc
Env. variables	

Driver maximums

Statements	256
Columns	256
DB cursors	64
Fetch buffer size	4096

Sort pages

Total	10000
In memory	1000

Merge buffer size

Max number of rows	10000

Copyright (c) 1996-2007, Trifox, Inc. California USA.
All Rights Reserved Worldwide.

ODBCdashboard

The "ODBCdashboard" dashboard makes extensive use of ODBC views. These must be set up before the dashboard will function correctly. Once you have created the DSN and system catalogs, open your favorite tool for executing SQL queries (if you don't have one, please contact Synergy/DE Developer Support and have them send you a copy of the Microsoft utility -ODBCTE32), and execute the SQL statements found in the file C:\Development\SynPSG\ChronoTrack\cat\chronotrack.sql.

LicensingDashboard

Once *xf*ODBC is set up correctly, there is no further configuration required for the LicensingDashboard.

How Does ChronoTrack….

This section discusses how the ChronoTrack application implements certain features.

How do I signal a UI Toolkit menu entry from within my .NET form?

Because ChronoTrack is built as a Synergy/DE UI Toolkit application, it still relies on UI Toolkit menu processing to control navigation. Within the Core folder there is an assembly called SynPSGNetCore. Within this assembly is a simple class called SynInterop. This class enables menu signals to be raised from within your .NET assembly and processed by the UI Toolkit application. To achieve this functionality:

Create an instance of the Interop class in the Synergy code:

```
 mInterop = new SynPSGNetCore.Interop.SynInterop()
 ;;assign the available event handlers
addhandler(mInterop.MenuSignal, WindowManager_MenuSignal)
```

Where "WindowManager_MenuSignal" is a Synergy method/subroutine that has the following code (for example):

```
public static method WindowManager_MenuSignal  ,void
  in  req menuName            ,string
     endparams
  proc
     m_signal(menuName)
     mreturn
 endmethod
```

When you create your .NET window in Synergy, pass the SynInterop object you created:

```
frmCustomerMaint = new ChronoTrackNetUI.CustomerMaintenance(mInterop)
```

Assuming it's C#, add the following to the top of the code:

```
using SynPSGNetCore.Interop;
```

Create a private reference to the Interop object (that will come from Synergy):

```
//need a copy of the interop object
private SynInterop interop;
```

For the constructor of the window, add a new parameter so we can pass in the Syninterop object from Synergy:

```
public CustomerMaintenance(SynInterop i)
{
      InitializeComponent();
      interop = i;
}
```

When you wish to raise a menu entry in the C# code, use:

```
interop.SendMenuSignal("CNTCTMNT");
```

where "CNTCTMNT" is the required menu name. When you process the window with dotnet_tkinput(), it will return with the menu entry.

Does the ChronoTrack application use xf*ServerPlus* and xf*NetLink .NET?*

No. The ChronoTrack Windows application does not use *xf*ServerPlus/*xf*NetLink; it uses the Synergy .NET assembly API and the gennet utility to wrap the .NET assemblies so the WinForms within the ChronoTrackNetUI assemblies can be hosted within the Synergy code.

However, the ChronoTrackNETUI assemblies reference the xfNLNet assembly (*xf*NetLink .NET client) and the ChronoTrack assembly used by the ChronoWebDX Web site and the ChronoTrackWS Web service. The UI projects reference these assemblies purely for design time purposes. With the ability to reference the data classes exposed by the ChronoTrack assembly (generated from the SMC), we can create Object Data Sources. These Object Data Sources can then be assigned to the UI controls at design time and thus have references to the individual fields within the classes. We can then data bind at design time to these class fields.

The Future of ChronoTrack

ChronoTrack is designed to be an example of how to implement the many capabilities available to your mission-critical business applications. The PSG team will continue to implement new technologies and capabilities as they become available with Synergy/DE. ChronoTrack will continue to showcase new features at future Synergex Success Partner Conferences. The ChronoTrack Windows application is being enhanced to incorporate the latest Windows Presentation Foundation (WPF) capabilities to deliver the latest user interface capabilities. The ChronoWebDX Web site is being enhanced with the latest AJAX and SilverLight technologies to improve performance and usability. The ChronoTrack environment is also being utilized as a testing bed for our Synergy/DE for .NET development, with the aim to build ChronoTrack within Visual Studio. If you have any questions, comments, or suggestions while using ChronoTrack, please email them to Synergy/DE Developer Support at support@synergex.com