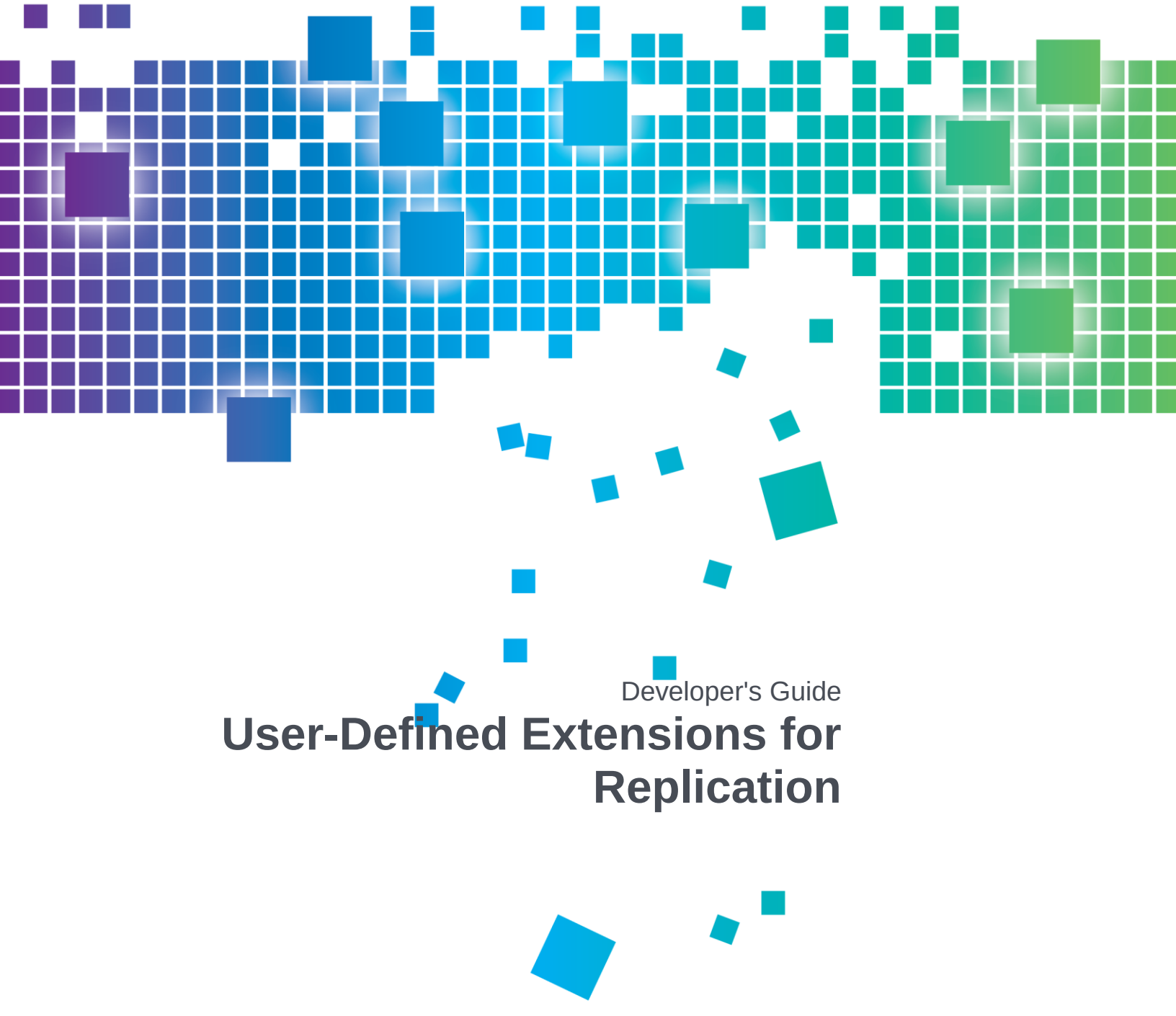
c-treeACE®
Advanced Core Engine



Developer's Guide

# User-Defined Extensions for Replication

# Contents

# 1. Replication Agent Extension Library Support

The Replication Agent now supports loading a user-defined external library that can perform customized actions when processing replication operations. The library contains functions that are called when certain predefined events occur. This feature allows a software developer to customize the Replication Agent's behavior to support capabilities such as:

- Conflict detection and resolution
- Data transformation
- Filtering or redirecting data updates
- Replicating to a third-party database

### Conflict Detection and Resolution

At their core, replication solutions need to copy data from a source server to a target server for the purpose of providing a failover server, maintaining backup copies, or creating an environment for testing or reporting. A foremost consideration is detecting and resolving conflicts between the data on the source and target servers. The Replication Agent is able to provide basic default handling of conflicts; however, many situations require adherence to individual corporate procedures based on proprietary business logic. The Extension Library allows developers to provide specialized logic to handle these situations according to your corporate policies.

### Data Transformation

Closely related to conflict detection, some replication environments must accommodate replication between non-identical datastores. For example, it may sometimes be necessary to replicate between databases with different schemas. The Extension Library can be used to provide functions that transform the source data to comply with the schema of the target data.

For example, suppose a replicated server is taken offline and a schema upgrade is performed on it. Because it now has a newer schema than the other replicated servers, the Extension Library could transform the data structure from the original server to the upgraded server. Note, be sure to also consider the use of FairCom's new Hot Alter Table logic for helping to solve this scenario. See the Hot Alter Table documentation for specifics.

### Filtering / Redirecting Data Updates

In the example above, an alternative strategy could be to selectively replicate to the updated server. In this case, the Extension Library could be used to filter out the data that should not be replicated.

### Replicating to a Third-Party Database

When replicating to a third-party database, the Extension Library could be used to translate replication operations into inserts and updates to be applied to the third-party database.

### The Replication Agent Extension Library SDK

The Replication Agent extensions are provided by an application developer within an SDK framework in an external library loaded by the Replication Agent at runtime. When a user-defined extension library is used, the Replication Agent calls a function in the external library for each operation that it reads from the source server. The function can modify the operation values, or take custom actions, and it can indicate to the agent what action to take for that operation.

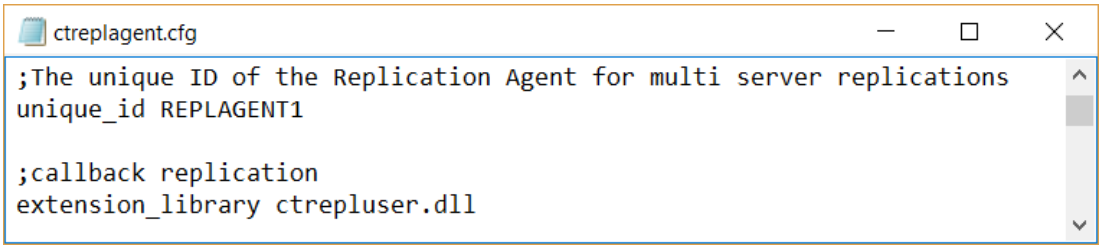
To use this feature, the developer needs access to c-treeACE Professional and to a development environment that supports writing and compiling C code into a DLL or shared library.

## 1.1 Configuring the Replication Agent to Use a User-Defined Extension Library

To register an external library to be used by the Replication Agent, add the option *extension_library* to *ctreplagent.cfg*, optionally including the directory name in which it is located if necessary:

```
extension_library ctrepluser.dll
```

As a suggestion, you can add the setting after `unique_id`, which will result in the *ctreplagent.cfg* file appearing as shown below:




This configuration will take effect upon the initialization of the Replication Agent.

You must have *ctrepluser.dll* present. This example assumes it is in the same path as *ctreplagent.cfg*.

## 1.2 Building a User-Defined Extension Library

To build a Replication Agent extension library, you will need to use the c-treeACE Professional package.

Create a client-side multi-threaded DLL or shared library makefile or project file by running **mtmake** with the *repldll* command-line option as shown below:

```
c:\c-treeACE_Pro_path>mtmake repdll
```

The makefile or project file will contain rules to compile the source file ctree/samples/special/utils/ctrepluser.c which contains your user-defined callback functions, and will link that module into a DLL named ctrepluser.dll or a shared library named libctrepluser.so (or other naming convention depending on the operating system).

The external library must provide a function named rxOnStartAgent() for the Start agent (see details on page 5) event. All other callbacks are optional: they do not need to exist in your external library.

## 1.3 Replication Logging

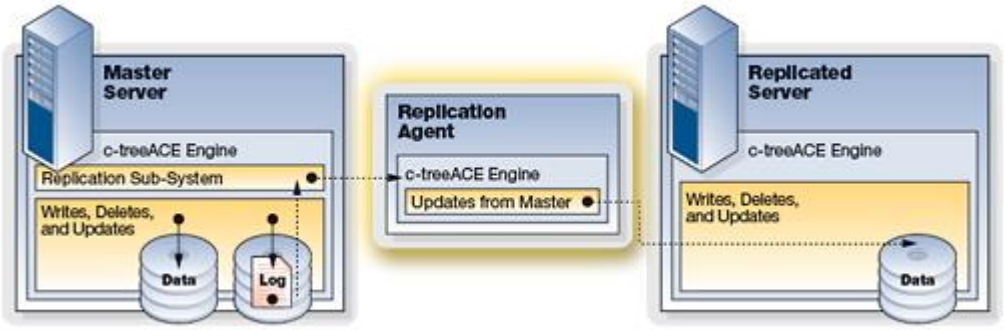
When compiling *ctrepluser.c*, you can uncomment `#define REPLUSER_LOGGING` if you want to enable code that calls a function to log a message to the file *ctrepluser.log* each time one of the callback functions is called. The sample code also tracks the number of calls made to each callback function.

# 2. Tutorial

In this tutorial, we will run an example that demonstrates one possible use of the User-Defined Extensions for replication by providing detailed logging of the replication subsystem.

To exercise replication allowing us to monitor its behavior, we will compile *ctrepluser.c* with `#define REPLUSER_TUTORIAL` uncommented. This will enable code that calls a function to log a message to the file *ctrepluser.log* each time one of the callback functions is called. The sample code also tracks the number of calls made to each callback function using the `#define REPLUSER_LOGGING` logic discussed in *Replication Logging* (page 3). Note this is only one simple use of the callback extension. By reviewing the code in *ctrepluser.c*, you will be able to explore many alternate uses for this technology.

To begin the tutorial, we will execute some scripts to build the data on Source Server and Target Server. With data on both servers, we can insert data and delete data in the table we built. We will connect to the Source and watch the Replication Agent replicating data to the Target Server.




## 2.1 Enable Replication

To enable replication, set the following keyword in the *ctsrvr.cfg* file on the Source Server:

```
REPLICATE *.*
```

This tells the source server to replicate all its files. This configuration only takes effect during server initialization; if the Source Server is running, you must restart.

If the Target server is not running, start it now. Both servers must be running.

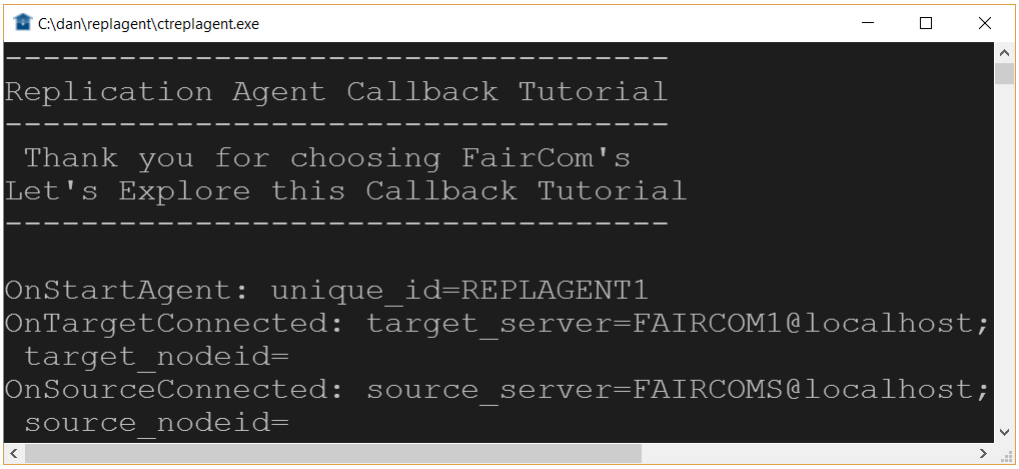
> **Note**: See the Replication Agent documentation for more details about the command to turn on replication.

## 2.2 Start the Replication Agent

Execute the `ctreplagent` command to start replication, for example:

```
# ./ctreplagent
```

With the Replication Agent started, we are now able to run the scripts to insert and delete data. Please check if your screen appears like the one below, if not please recheck your compilation.

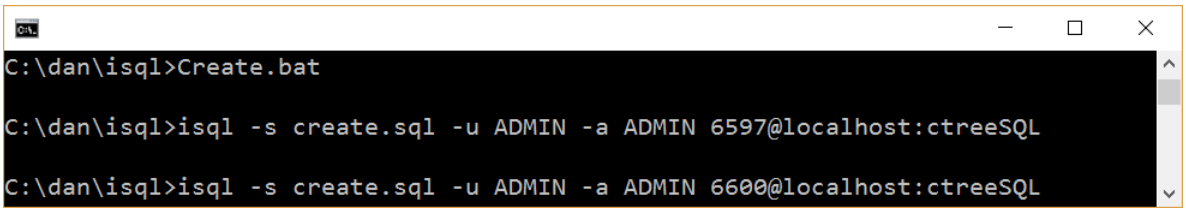



## 2.3 Create Data

Now run the following SQL command using c-treeACE SQL Explorer, or the ISQL command-line utility to create data on the Source and Target:

```
CREATE TABLE custmast (cm_custnumb CHAR(4), cm_custzipc CHAR(9), cm_custstat CHAR(2), cm_custrtng
CHAR(1), cm_custname VARCHAR(47), cm_custaddr VARCHAR(47), cm_custcity VARCHAR(47));
COMMIT;
```

The image below shows a batch file, *Create.bat*, that uses ISQL to execute a script, *create.sql*, that contains the commands shown above:

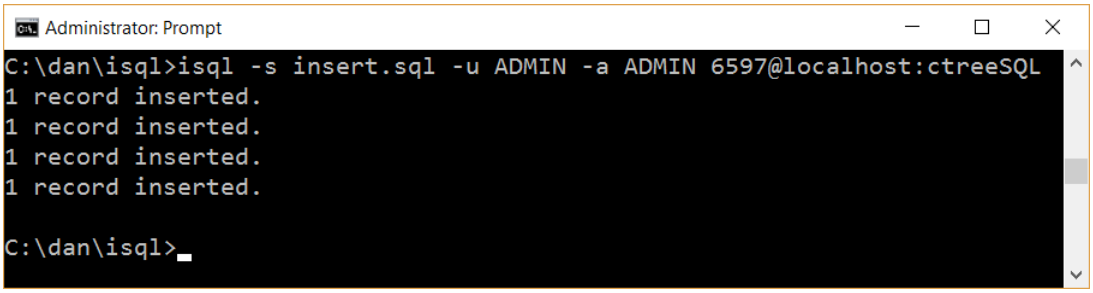



## 2.4 Insert Data

For this tutorial, we will insert some sample data into the database to be replicated:

```
INSERT INTO custmast VALUES ('1000','92867','CA','1','Bryan Williams','2999 Regency','Orange');
INSERT INTO custmast VALUES ('1001','61434','CT','1','Michael Jordan','13 Main','Harford');
INSERT INTO custmast VALUES ('1002','73677','GA','1','Joshua Brown','4356 Cambridge','Atlanta');
INSERT INTO custmast VALUES ('1003','10034','MO','1','Keyon Dooling','19771 Park
Avenue','Columbia');
COMMIT;
```

The image below shows ISQL running a SQL script that executes the commands shown above:

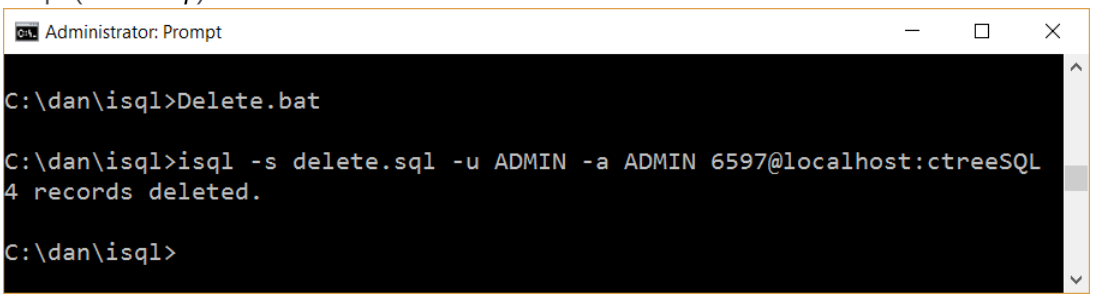



## 2.5 Delete Data

Next, we will delete the data to test replication:

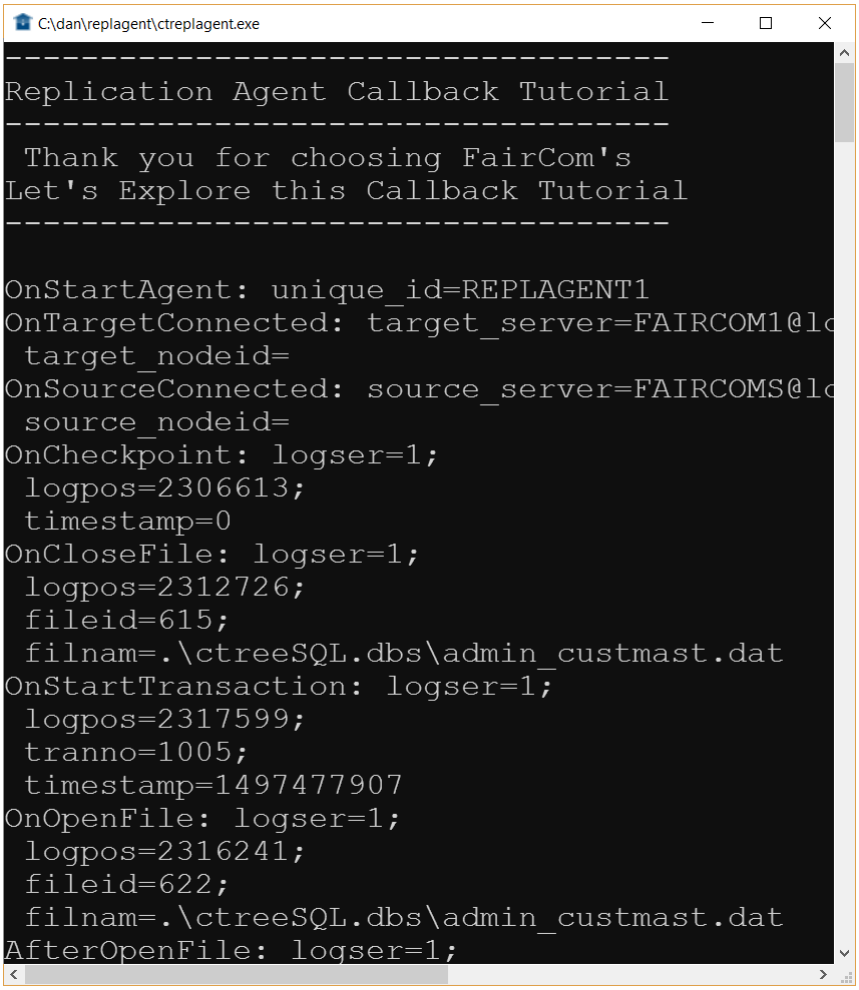
```
DELETE FROM custmast;
COMMIT;
```

The image below shows a batch file (*Delete.bat*) that does the deletion using ISQL by executing a script (*delete.sql*) that contains the commands shown above:



## 2.6 Results

Now we can check the results of the callback functions executed. The image below shows a sequence of events seen while inserting data:



# 3. Replication Events

The Replication Agent calls the corresponding callback function in the external library when the following events occur:

**Replication Agent State Change Events:**

- **Start agent**: The Replication Agent is starting.
- **Connected to source**: The Replication Agent successfully connected to the source server.
- **Connected to target**: The Replication Agent successfully connected to the target server.
- **Lost connection to source**: The Replication Agent has lost its connection to the source server.
- **Lost connection to target**: The Replication Agent has lost its connection to the target server.
- **Disconnecting from source**: The Replication Agent is disconnecting from the source server.
- **Disconnecting from target**: The Replication Agent is disconnecting from the target server.
- **Terminating agent**: The Replication Agent is terminating due to an unhandled error.
- **Pause agent**: The Replication Agent has been requested to pause its operation.
- **Resume agent**: The Replication Agent has been requested to resume its operation.
- **Stop agent**: The Replication Agent is shutting down.

**File Events:**

- **Open file**: The Replication Agent is opening a replicated file on the target server.
- **After file open**: Replication Agent has successfully opened a replicated file on target server.
- **Close file**: The Replication Agent is closing a replicated file on the target server.
- **Alter schema**: The Replication Agent is performing an alter schema operation on a replicated file the target server.

**Transaction Events:**

- **Start transaction**: The Replication Agent is starting a transaction on the target server.
- **Commit transaction**: The Replication Agent is committing a transaction on the target server.
- **Abort transaction**: The Replication Agent is aborting a transaction on the target server.
- **User-defined log entry**: Replication Agent is processing a user-defined log entry operation.
- **Checkpoint**: The Replication Agent is processing a checkpoint log entry.

**Data Events:**

- **Add record**: The Replication Agent is adding a record to a replicated file on the target server.
- **Update record**: Replication Agent is updating a record in a replicated file on the target server.
- **Delete record**: Replication Agent is deleting a record from a replicated file on target server.
- **Deferred key**: The Replication Agent is processing a deferred key log entry.

**Exception Events:**

- **Exception**: An action performed by the Replication Agent has failed.

The next sections describe the events available in the library.

**Note**: The external library must provide a function named **rxOnStartAgent()** for the **Start agent** (page 9) event. All other callbacks are all optional: they do not need to exist in your external library.

# 3.1 Replication Agent State Change Event Details

### Start agent

The Replication Agent is starting. The external library *must* provide a function named **rxOnStartAgent()** for the **Start agent** event to handle any extension library initialization that may be needed.

```
VOID rxOnStartAgent(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxVEROP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues startup.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Connected to source

The Replication Agent successfully connected to the source server.

```
VOID rxOnSourceConnected(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Connected to target

The Replication Agent successfully connected to the target server.

```
VOID rxOnTargetConnected(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: rxEVENT action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.

- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Lost connection to source

The Replication Agent has lost its connection to the source server.

```
VOID rxOnSourceLostConnection(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Lost connection to target

The Replication Agent has lost its connection to the target server.

```
VOID rxOnTargetLostConnection(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Disconnecting from source

The Replication Agent is disconnecting from the source server.

```
VOID rxOnSourceDisconnected(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Disconnecting from target

The Replication Agent is disconnecting from the target server.

```
VOID rxOnTargetDisconnected(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Terminating agent

The Replication Agent is terminating due to an unhandled error.

```
VOID rxOnTerminateAgent(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Pause agent

The Replication Agent has been requested to pause its operation.

```
VOID rxOnPauseAgent(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent pauses its operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Resume agent

The Replication Agent has been requested to resume its operation.

```
VOID rxOnResumeAgent(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as RXA_DEFAULT).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Stop agent

The Replication Agent is shutting down.

```
VOID rxOnStopAgent(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

## 3.2 File Event Details

### Open file

The Replication Agent is opening a replicated file on the target server.

```
VOID rxOnOpenFile(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxFILOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent opens the file on the target server.
- *RXA_SKIP* - The Replication Agent does not open this file on the target server.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### After file open

The Replication Agent has successfully opened a replicated file on the target server.

```
VOID rxAfterOpenFile(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxFILOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent continues normal operation.
- *RXA_SKIP* - Returning this action for this event has no effect on the Replication Agent (same as *RXA_DEFAULT*).
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Close file

The Replication Agent is closing a replicated file on the target server.

```
VOID rxOnCloseFile(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxFILOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent closes the file on the target server.
- *RXA_SKIP* - The Replication Agent skips closing the file.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Alter schema

The Replication Agent is performing an alter schema operation on a replicated file the target server.

```
VOID rxOnAlterSchema(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxFILOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent performs the alter schema operation on the target data file.
- *RXA_SKIP* - The Replication Agent skips the alter schema operation.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Resync file

The Replication Agent is resyncing a replicated file with the target server.

```
VOID rxOnResyncFile(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxFILOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent performs the resync file operation on the target data file.
- *RXA_SKIP* - The Replication Agent skips the resync file operation.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

## 3.3 Transaction Event Details

### Start transaction

The Replication Agent is starting a transaction on the target server.

```
VOID rxOnStartTransaction(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxTRNOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent starts a transaction on the target server.
- *RXA_SKIP* - The Replication Agent does not start a transaction on the target server.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Commit transaction

The Replication Agent is committing a transaction on the target server.

```
VOID rxOnCommitTransaction(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxTRNOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent commits the transaction on the target server.
- *RXA_SKIP* - The Replication Agent does not commit the transaction on the target server.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Abort transaction

The Replication Agent is aborting a transaction on the target server.

```
VOID rxOnAbortTransaction(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxTRNOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent aborts the transaction on the target server.
- *RXA_SKIP* - The Replication Agent does not abort the transaction on the target server.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### User-defined log entry

The Replication Agent is processing a user-defined log entry operation.

```
VOID rxOnUserTransaction(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxTRNOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent processes the user defined log entry.
- *RXA_SKIP* - The Replication Agent ignores the user defined log entry.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Checkpoint

The Replication Agent is processing a checkpoint log entry.

```
VOID rxOnCheckpoint(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxTRNOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent processes the checkpoint entry.
- *RXA_SKIP* - The Replication Agent ignores the checkpoint entry. Be aware that if you skip the Replication Agent's processing of checkpoint entries, you must manually inform the source server of the Replication Agent's current minimum transaction log requirement.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

# 3.4 Data Event Details

### Add record

The Replication Agent is adding a record to a replicated file on the target server.

```
VOID rxOnAddRecord(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxDATOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent adds the specified record to the target file.
- *RXA_SKIP* - The Replication Agent ignores the add record entry.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Update record

The Replication Agent is updating a record in a replicated file on the target server.

```
VOID rxOnUpdateRecord(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxDATOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent updates the specified record in the target file.
- *RXA_SKIP* - The Replication Agent ignores the update record entry.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Delete record

The Replication Agent is deleting a record from a replicated file on the target server.

```
VOID rxOnDeleteRecord(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxDATOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent deletes the specified record from the target file.
- *RXA_SKIP* - The Replication Agent ignores the delete record entry.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

### Deferred key

The Replication Agent is processing a deferred key log entry.

```
VOID rxOnDeferredKey(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxDATOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent processes the deferred key entry.
- *RXA_SKIP* - The Replication Agent ignores the deferred key entry.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

## 3.5 Exception Event Details

**Exception**

An action performed by the Replication Agent has failed.

```
VOID rxOnException(prxEVENT prxagentev);
```

Input Parameters: *rxEVENT* with *rxEXCOP* (see *Callback Function Input Parameters* (page 17))

Output Parameters: *rxEVENT* action field:

- *RXA_DEFAULT* - The Replication Agent handles the specified exception using its normal exception handling procedure.
- *RXA_SKIP* - The Replication Agent ignores the exception.
- *RXA_SHUTDOWN* - The Replication Agent shuts down.

# 4. Callback Function Input Parameters

The structures described in this section hold input parameters that are passed to the callback functions.

## 4.1 rxEVENT

Replication Agent event structure (*rxEVENT*):

```
typedef struct _rxevent {
    LONG    versn;            /* [IN] version of this structure      */
    LONG    errcod;           /* [IN] Error code                     */
    pctCNXH psrccnxhnd;       /* [IN] Source server connection handle   */
    pctCNXH ptgtcnxhnd;       /* [IN] Target server connection handle   */
    pVOID   pusrctx;          /* [OUT] User context pointer          */
    RXACT   action;           /* [OUT] Set to action to take         */
    union {
           rxVEROP verop;     /* [IN] Parameters for version check          */
           rxFILOP filop;      /* [IN] Parameters for file events           */
           rxTRNOP trnop;      /* [IN] Parameters for transaction events */
           rxDATOP datop;      /* [IN] Parameters for data events           */
           rxEXCOP excop;      /* [IN] Parameters for exceptions     */
    } ev;
```

- Input: Structure version
- Input: Error code for the current event
- Input: Source server connection handle - Can be used to call Replication API functions such as **ctReplOpenFile()** on the source server.
- Input: Target server connection handle - Can be used to call Replication API functions such as **ctReplOpenFile()** on the target server.
- Input: Replication Agent unique ID.
- Input: Source server name.
- Input: Target server name.
- Input: Source server node ID.
- Input: Target server node ID.
- Input: One of the following structures (described following this section), depending on the type of event:
  - *rxVEROP* - For the Start agent callback
  - *rxFILOP* - For file operation callbacks
  - *rxTRNOP* - For transaction operation callbacks
  - *rxDATOP* - For data operation callbacks
  - *rxEXCOP* - For exception callbacks
- Output: Optional user-defined context pointer - Can be used to store a pointer to memory allocated by the external library.

- Output: Action to take for this event. Supported actions are:
  - *RXA_DEFAULT* - Perform the default action for this event.
  - *RXA_SKIP* - Skip the default action for this event. For some events this action has no effect; that is, it is treated as *RXA_DEFAULT*. In the table of events below.
  - *RXA_SHUTDOWN* - Shut down the Replication Agent.

## 4.2 rxVEROP

The Replication Agent version structure (*rxVEROP*) is only used for the start agent event. The **Start agent** callback function must set the fields of this structure to the structure versions in use by the external library so that the agent can verify that the external library is compatible. See the sample start agent callback function for an example of setting these values.

```
typedef struct _rxverop {
    LONG    verop_ver;      /* [OUT] Set to version of rxVEROP      */
    LONG    event_ver;      /* [OUT] Set to version of rxEVENT      */
    LONG    filop_ver;      /* [OUT] Set to version of rxFILOP      */
    LONG    trnop_ver;      /* [OUT] Set to version of rxTRNOP      */
    LONG    datop_ver;      /* [OUT] Set to version of rxDATOP      */
    LONG    excop_ver;      /* [OUT] Set to version of rxEXCOP      */
    LONG    filh_ver;       /* [OUT] Set to version of ctFILH       */
} rxVEROP, *prxVEROP;
```

## 4.3 rxFILOP

The file operation structure (*rxFILOP*) is passed to file operation callbacks:

```
typedef struct _rxfilop {
    pctCHGB pchgbuf;       /* [IN] File operation details                */
    pctFILH pfilhnd;       /* [IN] File information               */
} rxFILOP, *prxFILOP;
```

- Input: File operation details for this event - Includes log position.
- Input: File information for this event - Includes data file ID, data file name, and data file number if open.

## 4.4 rxTRNOP

The transaction operation structure (*rxTRNOP*) is passed to transaction operation callbacks:

```
typedef struct _rxtrnop {
    pctCHGB pchgbuf;       /* [IN] Transaction operation details   */
} rxTRNOP, *prxTRNOP;
```

- Input: Transaction operation details for this event - Includes log position and transaction number.

## 4.5 rxDATOP

The data operation structure (*rxDATOP*) is passed to data operation callbacks:

```
typedef struct _rxdatop {
    pctCHGB pchgbuf;       /* [IN] Data operation details                */
    pctFILH pfilhnd;       /* [IN] File information               */
} rxDATOP, *prxDATOP;
```

- Input: Data operation details for this event - Includes log position, file ID, unique key, record image; the callback can change the key value and record image.

## 4.6 rxEXCOP

The exception operation structure (*rxEXCOP*) is passed to exception callbacks:

```
typedef struct _rxexcop {
    LONG    errcod;        /* [IN] Error code */
    pctCHGB pchgbuf;       /* [IN] Operation that failed */
    pctFILH pfilhnd;       /* [IN] File information */
} rxEXCOP, *prxEXCOP;
```

- Input: Error code for this exception.
- Input: Details for the operation that failed - Includes information relevant to the type of operation that failed (for example data operation includes log position, file ID, unique key and record image).
- Input: File information for this exception - Includes data file ID, data file name, and data file number if open.

# Copyright Notice



**Trademarks**



11/16/2018

# 5. Index