



## SQL Server Integration Services Standards

---

\*The ideas provided in this document are not “set in stone” and should be adapted to fit any specific business and technology needs.

### SQL Server Integration Services (SSIS) Standards and Best Practices

Integration Services is an ETL (Extraction, Transformation, and Loading) tool designed to aid developers with the task of moving data from specific sources and into specific destinations. SSIS facilitates this movement by making the data available to be manipulated while in transit so that it conforms to the destination’s specific data formats. SSIS is an “in-memory” pipeline which means once the data is pulled from the source it resides in memory until the user chooses to write it. This makes SSIS very fast and efficient for loading data. However, it requires careful watch on how much data is currently in memory, as it could easily lead to resource contention between processes.

#### Suggested Naming Conventions

Naming within SSIS is important for several reasons. It can act as a link between different Visual Studio projects that are all parts of the same IT project. It helps to describe the purpose of each package or each task within a package. It identifies where connection managers and source/destination items reside and/or are pointing too, and can aid in SSIS project deployments.

- **Visual Studio Project Name**
  - Using “name spaces” when naming projects works to create a logical link between different projects. For example, an SSIS package could be named *Client.ProjectABC.ETL* while an SSRS project might be named *Client.ProjectABC.Reports*. Using this approach, it’s easy to identify that these 2 separate Visual Studio projects are really different pieces of an overall IT project and are related.
    - Ex. *Client.ProjectABC.ETL*
  
- **Package Name**
  - The name of the package should reflect its function. The use of name spaces can be used here as well, although periods should not be used to separate spaces as those would cause any packages deployed to the file system to be incorrectly read. Underscores could be used in their place.
  - Reusing part of the project name in the package name can be useful for deployment as once a package is deployed, it is no longer part of the Visual Studio project.
    - Ex. *ProjectABC\_LoadTimeDimension.dtsx*
  
- **Task and Dataflow Item Names**
  - In general, the name of the task should describe its overall function.
    - Ex. A Dataflow task used to load only new records into a data warehouse fact table could be called *Load New factSales*.
    - Ex. A Lookup Dataflow item used to look up the ID for an employee might be called *Get Employee ID*.



- Generally, Dataflow source and destination item names should reflect the source or destination to which they are specifically connected. This could be a table or stored procedure within a database, or a flat file, etc
  - Database Table Ex. *dimEmployee*
  - Stored Procedure Ex. *uspGetAllEmployees*
- **Connection Manager Names**
  - Similar to Dataflow Sources and Destinations, the name of a Connection Manager should reflect the actual connection that they are making.
    - Database Ex. *ABCDataWarehouse*
    - Flat File Ex. *NewOrders.csv*
  - It is important to note that, by default, SQL Server will append the Host/Instance name to the beginning of the Connection Manager's name. Generally, it is better to remove this as this may not accurately reflect the true name of the Host/Instance once a package is migrated from one environment to another.
- **Package Variable Names**
  - Package variable names should describe their contents and use
- **Running Packages As a SQL Agent Job**
  - When setting up a package to run on a schedule via a SQL Server Agent job, it is important to name it in such a way that a system administrator can easily identify it and, to some degree, be able to tell its overall function and/or to which project, program, or process it belongs.

## Package Documentation

Thorough in-package documentation greatly aids in allowing more than one developer to understand the purpose of a given SSIS package and how it functions. It conveys meanings and ideas that simply interpreting the structure of a package or following along with the code cannot.

- **In Task Descriptions**
  - Each Control Flow Task and Data Flow Item allows a user defined description to be written as part of the Task or Item properties. These Description properties should be utilized to further describe the functionality taking place within the task or item, any special circumstances, or really any other bit of information the developer feels would facilitate another developer interpreting and taking over development of a given SSIS package.
- **Variable Descriptions**
  - As part of the properties associated with a package variable, a description field is made available. A developer should use this field to identify what the role of the variable will be and what data should be contained within it.



- **Annotations**
  - In SSIS, package annotations act as descriptive labels helping to illustrate how a package works. These annotations can be placed anywhere within the background of a Control Flow or Dataflow.
  - It is good practice to place very large annotations inside of a collapsible Sequence Container in order to minimize the impact on space taken up by the Annotation.
  
- **Package Flower Box**
  - An annotation should be used as an initial flowerbox type of documentation at the top of each package. This flower box should contain information that is general to the package including such things as author, package name, package creation date, and the general purpose and description of what the package does.
  
- **Script Tasks and SQL Tasks**
  - In-line documentation should be done per regular .NET/T-SQL coding standards. An overall flower box comment does not, necessarily, need to be used within the script code itself as this would normally be handled already by the Task or Item description field.
  
- **Running Packages As a SQL Agent Job**
  - When setting up a package to run on a schedule via a SQL Server Agent job, it is important to add a description that a system administrator can use to easily identify over all function, project, program, or process it belongs. This description may also be used to note the original developer or owner of the application and/or overall process.

## **Error Reporting**

Regardless of the skill of the developer, or the cleanliness of the data, errors will occur. For this reason, it is a good idea to have a well established, reusable strategy for error reporting. There are many methods for error reporting, and it usually makes sense to implement more than one.

- **SSIS Built In Logging**
  - SSIS's built in mechanism for logging is a simple, yet very effective way to keep track of the execution (either manually or from scheduled jobs) of SSIS packages. Any package can make use of it and the level of detail logged for a package can be as granular as the user wishes.
  - Following are the author's general recommendations for logging in SSIS.
    - Each package should have logging enabled
    - At the least, OnExecStatusChanged, OnError, and OnTaskFailed events should be logged. This will capture the start and end time of the package execution as well as any fatal errors. So, if a package executes without error, there will normally be only 2 records logged, package start and package end. However, if an error occurs within the package, detailed information about the errors will be captured as well.



- SSIS provides several mechanisms to store log records generated from SSIS including Text Files, SQL Profiler, Windows Event Logs, XML Files, and SQL Server Database tables and it is possible to log to more than one of the above listed destinations. It is the preference of the author to utilize the SQL Server log provider and write log records to a database table for the following reasons:
    - Avoids unnecessary extra security at the file system level
    - Log records are automatically backed up when the database is backed up making them easier to restore.
    - All logging is centralized for “one stop shopping”
    - Log records are easily reportable via other applications including Reporting Services
    - During a reimplementation and movement of a database or instance, all log files associated with the database or instance are moved simultaneously vs. tracking down and copying them in a separate task.
  - The SQL Server Log provider can be used in concert with other log providers at the same time. For example, a developer can choose to log to the Windows Event Log, as well.
  - It is generally considered as good practice to include the logging tables within the same database as the package configuration data.
- **Emailing Error Information**
    - It is often desired or necessary to make the failure of an executing SSIS package known immediately and, potentially, to several people or groups. There are several ways to accomplish this, but the author has found that utilizing e-mail is consistently the most effective and easiest to maintain. The following guidelines are helpful when making use of this approach:
      - Only send e-mails to Active Directory distribution groups or Exchange groups. Individual e-mail addresses should not be used inside of an SSIS package or package configuration.
        - Ex. Utilize an existing (or create a new) AD security enabled distribution group for “ITSupport”. Any error messages created by an SSIS package get sent to that distribution group. In this way, as people join or leave the team, they will automatically be included or excluded from e-mails with no manual interaction.
      - Only send e-mails when errors are encountered.
      - Utilize the OnError and OnTaskFailed Event handlers to capture error messages and generate e-mails
        - It is important to note that SSIS will fire the OnError event handler at each hierarchical level in a package. For example, if a master package calls a child package which executes a control flow calling a dataflow task and the OLEDB source item fails, the OnError event could fire 4 times duplicating the error 4 times in the e-mail. As this can be confusing, it is good practice to only write error messages once for the initial error source only. A technique for accomplishing this may be included as an addendum to this document.



## Implementation, Migration, and Package Configurations

During the lifecycle of an SSIS package, it often becomes necessary to migrate it from one environment to another as is the case when implementing into production from development or test environments.

Manually updating such things as connection managers, variables, etc is not only tedious but quite dangerous. It is all too easy to introduce a typo, syntax error, or even to completely forget to make the update at all. In order to combat this, SSIS offers a method to deal with automatically updating these things called a Package Configuration.

SSIS Package Configurations allows a running package to dynamically set certain pieces of information pertaining to package functionality at runtime. These pieces of information are housed in a strategic location that the running SSIS package is aware of and can read from at the time of its execution.

- **Package Configurations**
  - Package configuration data can be housed in multiple ways
    - XML files located within the file system
    - Windows Environment Variables
    - Registry Entries
    - A parent package variable
    - A SQL Server database table
  - A package relies on a key or “Filter” within each of the above listed containers in order to access the correct information it needs in the package configuration. Each package should only utilize a single filter.
  - Anything external to a package (file location, connection string, etc) should make use of a package configuration.
  - The preferred method of utilizing package configurations is the indirect method. This method actually relies on 2 package configurations working in tandem.
  - Configuration 1
    - First, an XML configuration file is used. This file must be placed in each environment into which packages may be migrated. Generally, that means there will be one instance of this file residing on each machine housing a development, test, and production environment. The file must reside in exactly the same drive/directory in each environment and be named exactly the same.
      - Ex. “C:\SSIS\SSISConfigurations”
      - The content of the XML file is simply a single record containing a connection string.
      - The connection string points to a SQL Server database/table which acts as the main housing for the package configuration data.
      - A Windows environment variable could, potentially, work better in some situations where it is acceptable to restart the server as this alleviates the possibility of locating a file in a different place in the file system



- **Configuration 2**
  - Second, a SQL Server database and table are created to contain the actual package configurations. This database and table must exist in all environments and be named exactly the same.
    - The Configuration should be housed in its own database separating it from other program functionality.
    - SSIS logging tables should be included in this same database.
    - All package configuration data should be kept here
- **The use of a SQL Server database and table is preferred for the following reasons**
  - Avoids unnecessary extra security at the file system level
  - Configuration records are automatically backed up when the database is backed up making it easier to restore.
  - All configuration data is centralized
  - Configuration records are easily reportable via other applications including Reporting Services
  - During a reimplement and movement of a database or instance, all configuration records are moved simultaneously vs. tracking down and copying them in a separate task.
  - Configuration records are easily updated
    - Via SQL Queries
    - Directly to the database row
    - A custom GUI/application written for this purpose
- Utilizing an indirect package configuration as described above works as follows. Upon initial execution, the package first looks to the XML configuration file located on the file system. The XML file sets the connection string of a Connection Manager located within the executing package. Once the connection manager is set, the other SQL Server package configuration makes use of this connection manager to open a connection to a SQL Server database table which houses the actual configuration data. The configuration data in this table can be different in each environment so that packages in a production environment use properties specific to it, and packages in development environment use properties specific to that environment. In this way, a package can be moved from one environment to the other without having to manually update these properties.

## SSIS Template

A relatively easy way to aid in the implementation of SSIS standards is to utilize a templating strategy. A template can accomplish many of the routine tasks associated with new package creation allowing a developer to be more efficient and better adhere to set standards. With a template in place, a developer can simply choose to use that template when creating a new SSIS package. Listed below are several things that a template can accomplish.

- Automatically set up the standardized logging mechanisms



- Automatically set up Indirect Package configurations
- Provide pre-set flower box style annotations within the Control Flow

Setting up a template must be done on each local machine that will be creating SSIS packages. The template package must be copied to the following location on the each local machine:

```
C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\PrivateAssemblies\ProjectItems\  
DataTransformationProject\DataTransformationItems
```

It is good practice to house the original template on a network share or within some type of source control giving SSIS developers access to copy it to their individual local machines. In this way, any updates made to the template will be utilized.

It is important to note that, when utilizing a template for a new SSIS package, a New GUID ID must be generated. If not, the new package will retain the same GUID ID as the template and SQL Server will not be able to tell the difference between the new package and any other packing using the same GUID.

## Master Packages

When executing multiple related packages it is often useful to implement a Master or Control package. The Master package, in its most basic form, simply amounts to a Control Flow which sets precedence between other executing child packages. Using this approach one can:

- Set initial reusable values (aka global package variables) once and pass them to the child packages
- Wrap all executing packages up into a centralized error handling mechanism
- Control Local and Distributed Transactions for all child packages in one location
- When scheduling packages to execute via SQL Server Agent, one need only schedule the single Master package. All child packages will execute in the correct order and any messages generated by the executing job will be centralized.

## Package Deployment

When it comes time to deploy SSIS packages there are several different mechanisms and locations for storage. Each one has its pros and cons.

- **Deployment Locations**
  - SSIS packages may be deployed to the file system or to SQL Server. It is the preference of the author to deploy to SQL Server for several reasons. Because of that fact, this document will focus primarily on that type of deployment.
    - Avoids unnecessary extra security at the file system level
    - Packages are automatically backed up when the database is backed becoming much easier to restore.
    - All packages are centralized aiding in administration



- Reportable via other applications including Reporting Services
    - Simple queries can be used to list all package names and ID's, their user defined descriptions, when they were created, and their current version number. This data can then be linked to the tables that monitor and track SQL Agent jobs giving the administrator a good way to track all SSIS packages and when they execute.
  - During a reimplementation or movement of an instance, all packages are moved simultaneously vs. tracking down and copying them in a separate task and risking the possibility of missing some.
  - Package deployments through multiple environments are simplified
- **Deployment Tactics**
    - SSIS allows for deployment through multiple environments to be done in several different ways both manually, and automatic unattended. Deployments into either the file system or SQL Server are both similarly done and are outlined below.
      - Manual Deployment Via Visual Studio
        - Deployments through Visual Studio are not true deployments in the normal sense of the word. They are simply builds placed in the normal Visual Studio Project location. These builds CAN, however, be copied to a folder in the file system for a true deployment. Packages cannot be directly deployed to SQL Server using Visual Studio.
      - Package Deployment Utility
        - When a developer creates a package build from Visual Studio, they can choose to create a package deployment utility automatically. This utility creates an install file or "manifest". This manifest, along with any SSIS package files and configuration files is then copied to the server to which they should be installed. The user then double clicks on the manifest and an installation wizard walks the user through the installation process. The wizard can install packages to both the file system and to SQL Server
      - DTUtil
        - DTUtil is a command line tool packaged with Integration Services. This tool can be utilized to install SSIS packages to either the file system or to SQL Server via command line functionality. This allows SSIS packages to be installed via other programs or from user written scripts.
          - Ex. A dtutil command can be written to silently (unattended) install an SSIS package into SQL Server. This command can then be wrapped into a Windows Batch file or Powershell script. The batch file or script can then either be manually executed by a DBA or admin or can be scheduled for a one time, off hours, execution.
      - Manual Import
        - Packages can be manually installed into either SQL Server or the file system by connecting to an Integration Services instance from within Management Studio. By right clicking on the correct sub folder within



either MSDB or File System and choosing “Import” the user may then point Management Studio to the correct location of the SSIS package to be installed.

- As an important note to remember, it is very good practice to deploy SSIS packages through multiple environments in a linear fashion. That is, when deploying a package into a Production environment, always be sure to use the EXACT same package that was installed in the Test environment. A simple way to do this, and a technique employed by the author, is to manually import packages into the next environment by pointing the import process to the same package in the previous environment.
  - Ex. An SSIS package which has been running in the TEST environment and which has been thoroughly tested is now ready to move to production. The DBA connects to the production SSIS instance, navigates to or creates the appropriate folder under MSDB, right clicks that folder and chooses “import package”. The DBA then points the import wizard to the corresponding location of the same package within SQL Server on the Test Environment. By utilizing this technique, users can ensure that only tried and tested versions of packages are deployed forward. Copying packages directly from a development environment into a production environment is never a good idea as it’s possible to introduce unexpected changes and inconsistencies between the same package located in different environments.

## Source Control

It is vital to any software development endeavor that some form of source/version control be employed and Integration Services is no different. SSIS seamlessly integrates into many popular Source Control packages including Visual Source Safe and Team Foundation Server. Actual methodologies employed within the Source Control package for organization, branching, etc are out of scope of this document.

## Data Sources

A Visual Studio Data Source is a mechanism built into Visual Studio which allows a developer to set a connection string and then share that connection string within a project. It is important to note that a data source is NOT a mechanism of Integration Services. Although useful in other types of development, VS Data Sources should be avoided when creating an SSIS project. The reason for this is that these data sources cannot easily be automatically updated when packages move through different environments. Each data source must be manually updated when deployed by opening up the SSIS Package itself and changing the connection or altering the “.ds” file. This opens up the developer to introducing unintentional errors and inconsistencies to the package. Any data source type functionality should be contained within a Connection Manager. Connection Managers are specific to Integration Services and can, therefore, be manipulated via Package Configurations making deployments seamless and removing the possibility of the developer introduction errors to the package.



## General Recommendations and Guidelines

- ***SSIS is an “in-memory” pipeline. This is where a lot of its speed is derived from. It, therefore, makes sense to utilize this speed by not writing to disk as often as possible. Ideally, SSIS packages and processes should not write to staging tables, work tables, etc. Most functionality can and should be done in memory. This is, obviously, not always possible, but should be considered a goal to “shoot for”.***
  - Please take note that although the ability for SSIS to operate fully in memory is a great feature, you are limited by the amount of memory available to the process. When working with very large data sets, it is important to keep track of memory usage and contention. When possible, set the commit batch size to an appropriate number. Also, utilizing a batching methodology can be very useful. For example, a mechanism can be built into packages limiting that package to processing only a predefined number of records at a time.
- ***Remove unused columns from data sets.***
  - Often, columns are pulled from a source, used for a specific purpose, and then no longer needed for the remaining package execution. It is important to remove these columns from the pipeline at the point in the process where they are no longer necessary and free this previously allocated memory back up.
- ***Utilize Transactions when working with distributed data.***
  - When making use of data from multiple distributed systems, it is important to make use of SQL Server transactions. In this way, should connectivity between systems be lost, any altered data is returned to its previous state.
- ***Avoid using “Table or View” when pulling source data.***
  - “Table or View” is the equivalent of a SQL “Select \*” and brings in every row and every column in the selected table or view whether they are necessary to the process or not. This incurs extra overhead and memory usage. It is a better practice to filter both rows and columns in a source by utilizing T-SQL. Try to filter as much out from the source as possible before loading them into memory.
- ***Use Stored Procedures instead of in-line SQL***
  - If changes need to be made to a given SQL statement and those changes are housed within the SSIS package itself, then the entire package will need to be redeployed. However, if the same logic is encapsulated within a stored procedure, a developer can make alterations to that procedure without the need of redeploying an entire SSIS package.
  - Logic contained within a stored procedure is reusable between other SSIS packages and processes
  - Stored procedures cache an execution plan within SQL server, where as a new execution plan must be generated each time in-line SQL code is executed incurring extra overhead and processing time.
- ***Utilize the database for sorting activities when possible.***



- The sorting mechanisms contained within the RDBMS are not the same as the mechanisms contained within SSIS and will almost always be faster. Therefore, when possible, sort the data during the initial data pull and data set creation using a stored procedure.
- **Use the “WITH (NOLOCK)” hint**
  - It is good practice to use the “WITH (NOLOCK)” database hint when pulling source data. This helps to ensure that the SSIS package’s processing does not block other processes attempting to use the same data.
- **Package Protection Level**
  - By default, SSIS uses the “EncryptSensitiveWithUserKey” protection level. This means that any information contained within the package that SSIS deems as “sensitive” (ftp task passwords, database passwords, etc) are automatically encrypted when the package is built. The encryption key that is generated is based on the user’s Windows Key. If another user attempts to open or execute the package (ex. another developer or system account), an error message will be displayed and the package will neither open nor execute.
  - Other protection levels allow you to password protect either the sensitive data or the entire package. Although better, it’s still difficult to know what to do with the passwords, not to mention making deployment of packages more complex.
  - The recommended approach is to utilize the “Don’tSaveSensitive” protection level and store sensitive information like passwords outside of the package, then rely on Package Configurations to get to the sensitive data.
    - Ex. The existing SQL Server package configuration can be utilized to store sensitive information, because the database is already secured. The package will then read the configuration at run time and any sensitive information will be provided to the package.

## Final Thoughts

The information provided in this document is not comprehensive. There are various other recommendations and options that may be found useful and each suggestion should always be compared to other possible solutions. What works for some may not work for others, but this document provides a foundation for understanding and implementing SQL Server Integration Services into your workflow.