STATIONworks
Technical Brief

November 1999

STATIONworks Version 2.1
A FASTech MES Product

Preliminary

Published by Brooks Automation, Inc.
15 Elizabeth Drive / Chelmsford, Massachusetts 01248 / USA
(978) 262-2400
FAX (978) 262-2500
http://www.brooks.com OR www.fastech.com

Table of Contents

How to Deploy STATIONworks

1

Topics in This Chapter

This chapter presents all the components of STATIONworks, describing how you put them together in a development environment, then deploy them on the manufacturing shop floor.

NOTE This document is subject to change without notice. It presents features planned for an upcoming release, but it is not a guarantee of availability of those features by any particular date.

Purpose of STATIONworks Systems

STATIONworks systems are designed to develop and deploy station controllers/equipment managers in manufacturing facilities.

With STATIONworks, you can startup up, monitor, and stop multiple Production Stations on multiple PCs, each operating various pieces of equipment.

The system is made up of Stations that:

- Automate multiple pieces of equipment.
- Provide communication between that equipment and the MES.
- Display an interface to the equipment for operators on the shop floor.

Who Uses STATIONworks?

Several players in the manufacturing scenario use STATIONworks:

- Station Controller Developers—Design and develop Station controllers/equipment managers in state machine form.
- System Administrators—Determine which Production Stations will run particular pieces of equipment from particular PCs. Ensure that STATIONworks systems can start, stop, and be managed.
- CIM Engineers—Roll out new development logic into the production environment. Actually start, stop, and manage the equipment at manufacturing time using a STATIONworks. Evaluate the equipment utilitzation and manufacturing results.
- Operators—Interact with STATIONworks applications created by the developers.

Let’s take a closer look at the components of STATIONworks.

STATIONworks System Components

What are the components that make up a STATIONworks system?

Development Components

STATIONworks development components are:

- SwIDE (this component uses the SwMonitor)
- TOM Builder
- TOM Explorer (as a debugger)
- Visual Basic (Microsoft product)

Configuration and Deployment Components

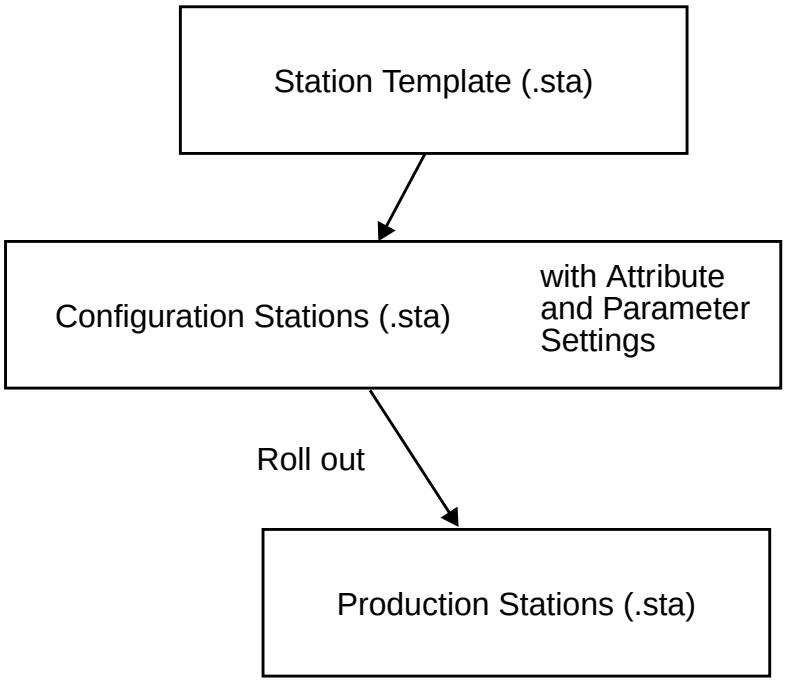
STATIONworks configuration and deployment components are:

- SwConfig
- SwRun
- TOM DB Browser

STATIONworks Development Components

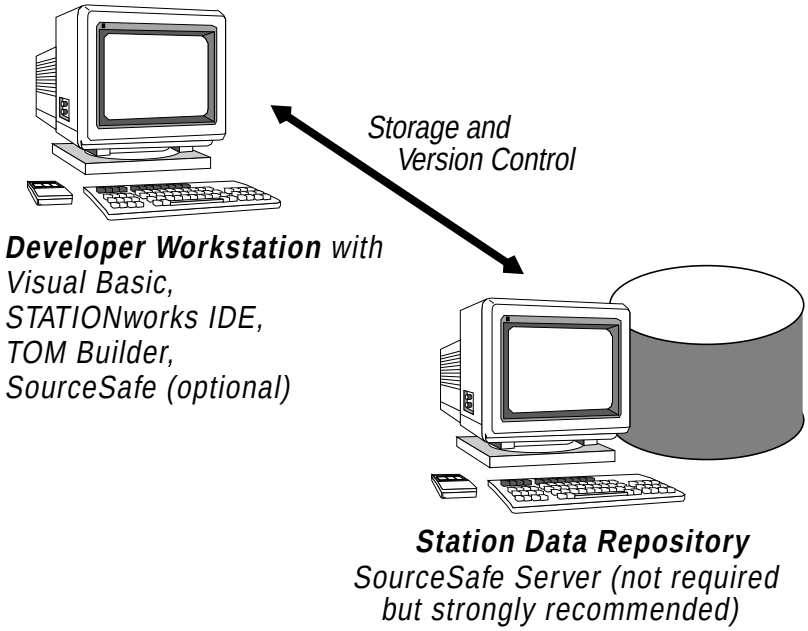
Let's first look at those intended for the STATIONworks developers:

- **STATIONworks IDE (SwIDE)**—The STATIONworks Integrated Development Environment. For developing state machines (*.fsm* files) on developer workstations. Through the IDE you can use the **STATIONworks Monitor** (**SwMonitor**) to run/test state machines while developing them and display *.fsm* diagnostic messages. When the developer creates a Station Template, the IDE creates a Station file (*.sta*) that contains that Station definition; this *.sta* file Station Template becomes the basis for one or more Configuration Stations, which you later roll out into Production Stations for actual manufacturing.



- **TOM Builder (TomBuilder)**—For developing tool databases or developing custom drivers on developer's workstations.
- **TOM Explorer (texplore)**—You can use the TOM Explorer to debug your Visual Basic applications.

- **Visual Basic** (Microsoft product)—For developing STATIONworks TOM Services or VBScript actions used in the state machine.

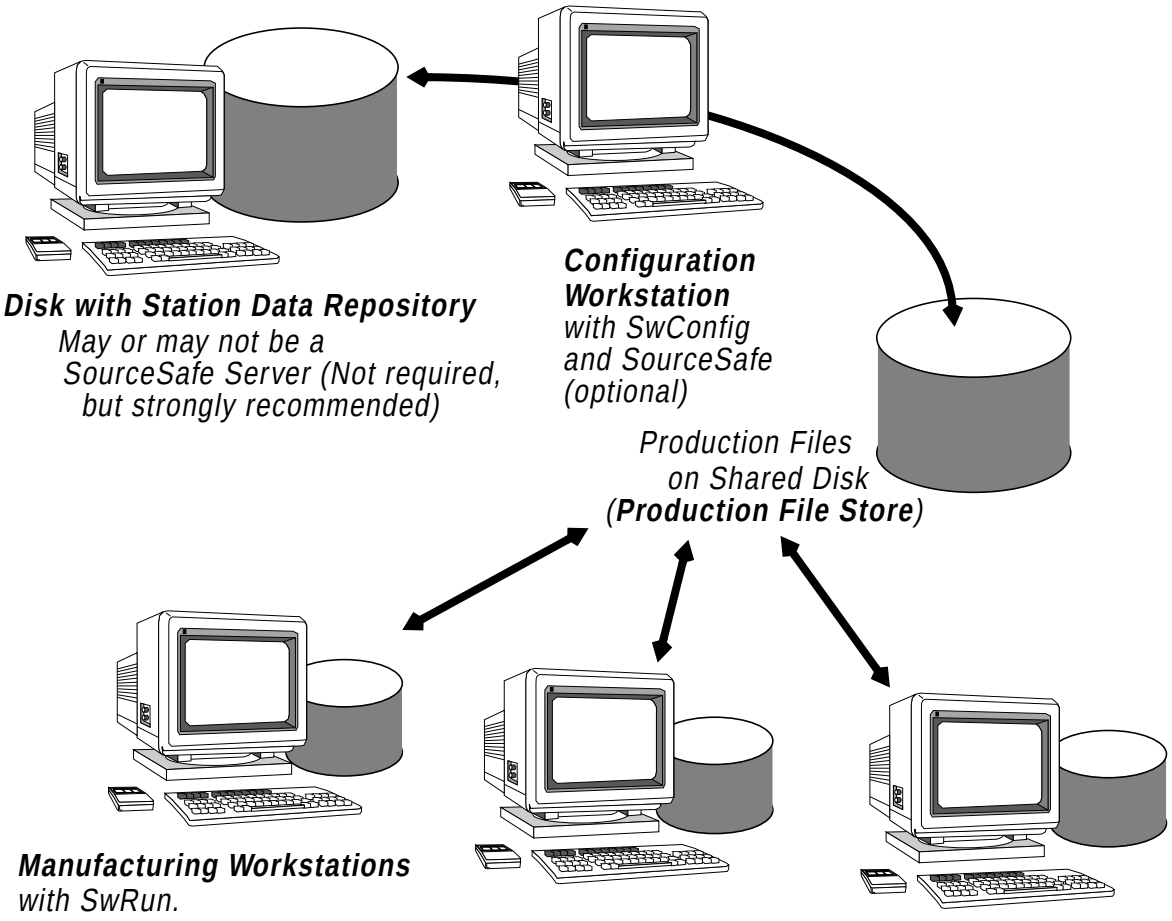


Structure of STATIONworks Developer Components

STATIONworks Configuration/Deployment Components

As part of preparing to deploy the applications, CIM engineers need to modify any TOM Attributes or state machine parameters as required before running the Production Station:

- **STATIONworks Configuration Manager** (**SwConfig**)—A central workstation that the CIM engineer uses to modify the configuration of a particular Configuration Station (*.sta* file) or group of Configuration Stations.
- **SwRun**—This element of the product runs the state machine and displays log messages in a very simple GUI. It lets you run, pause, and stop state machines.



Structure of STATIONworks Configuration and Deployment Components

- **TOM DB Browser**—This browser lets you view the Tools in the database at any time. Using this brower, you can quickly look at the Tool's Services, Attribute settings, and other characteristics.

How STATIONworks Components Work Together

How do these components work together?

Let's Look at the Development Stages...

Develop Tool Database

First, the developer develops a Tool database using TOM Builder. That database may include Tools provided with the product, custom Tool drivers, or drivers that are modifications of those provided—with custom attribute settings to apply to a particular piece of equipment.

Develop State Machines

Once the Tool database is established, the Station developer can create a state machine that uses a particular Tool in the database or a generic type of Tool, such as an ellipsometer or lot sorter. State machines that are designed to use a generic Tool can use any Tool of that type at run time.

Assign State Machines to Equipment

The Station developer or CIM engineer next designates the piece (or pieces) of equipment that a particular Tool driver and state machine will run. The developer assigns them by generating a Station Template in a configuration file—or Station (*.sta*) file—where that information is stored.

Check Tool Drivers, State Machines into SourceSafe

Once the developer has completed Tool drivers, state machines, and Station Templates, he or she checks them in to the source code control system, SourceSafe, where the CIM engineer using **SwConfig** can later retrieve them.

Next, Let's Look at the Configuration Stages...

Create Configuration Stations

The manufacturing engineer next matches up the Tools/state machines that are to run a given piece of equipment and works with the system administrator, who determines which PC will actually run a particular piece of equipment.

The CIM engineer uses **SwConfig** to create multiple copies of a developed Station Template. Those copies become Configuration Stations—more than one Configuration Station can use the same state machine and Station Template. For example, in a manufacturing area that has 10 lot sorters, the same Station Template can be used for each by generating a separate Configuration Station for each individual piece

of equipment. These 10 lot sorters all share the same state machine (*.fsm*) file or files and the same database (*.mdb*) file—the Station (*.sta*) file contains all this data as well as additional Station-specific data.

Configure Station to Run Equipment

Next, the engineer sets the parameters and TOM Attributes of the Configuration Stations for the particular pieces of equipment. The engineer may need to set parameters that are not known until this stage of the process, such as the IP address of the PC running the Tool and the settings for the serial port to the equipment.

Although the developer has already created the *.sta* file, the CIM engineer may modify the file as required before actually running the equipment. Modifying the Configuration Station's *.sta* file is how you configure the Station. At run time, **SwRun** uses the *.sta* file to determine the Tool driver, Tool database, and state machine that it needs to run a piece of equipment. A single PC might run several pieces of manufacturing equipment, so you might create several Configuration Stations for it.

The *.sta* file contains:

- Tool or Tools to run
- State machine(s) (*.fsm*) file to run and versions of the *.fsm* files
- Tool database (*.mdb* file) and version of that *.mdb* file
- Tool aliases (actually, Resource aliases, such as `Res1`, `Res2`, `Res3`. You can change the Resource associated with an alias names.)
- Log information
  - Path to the log file that receives messages the state machine logs
  - Maximum size log file is allowed to become
  - Maximum number of log files allowed to accumulate
- State machine parameter settings
- TOM Attribute settings

You can change any of this information after generating Configuration Stations based on the Station Templates.

Configure Production Stations —Generate Production File Store

Before running any of the state machines or working with any of the Tools in the database, the CIM engineer must first *configure* the Configuration Stations.

Now that you have generated Configuration Stations from the Station Templates, there are multiple Configuration Stations based on each Station Template. When those Configuration Stations have been completely configured (and checked into SourceSafe), the CIM

engineer can use **SwConfig** to *roll out* those Stations to production —putting them into a directory structure on a shared drive that mimics the Configuration Workspace area. This shared drive is the ***Production File Store*** area. The files in this area are the ones you can actually run over the network or copy to individual Production Station nodes as needed. The versions of these files are retaied in the Production File Store area (and ideally in SourceSafe as well) should there be a system crash. If you have put the files from the Production File Store area into SourceSafe, you can also use those files to roll back to an earlier version.

Now, the Deployment Actually Takes Place...

Start and Monitor the Station

Now, you are ready to run the Production Station from the Production File Store area's Station (*.sta)* file. You can start it from the Production File Store area and run it over the network or you can copy it to the individual PC that is running the particular equipment and run the *.sta* file locally there.

How do you run the *.sta* file? You use **SwRun**, which provides a GUI that displays messages about what the state machine is doing.

Once the Station is running, you can monitor it using the messages **SwRun** displays. Or if you are running the Station from the **IDE**, you can monitor its actions in the `SwMonitor` window.

At this point the Production File Store area contains all the files your Production Stations require. You should strictly control what happens to those files to ensure you can always restore them to their current revision levels.

What Files Does the STATIONworks System Use?

Below is a summary of the files the STATIONworks system uses:

State Machines and Stations:

- *.fsm*—Finite State Machine. Contains the state machine designed in the STATIONworks IDE using its browser or built-in Visio graphical user interface.
- *.sta*—Contains the Station configuration for a Station Template, Configuration Station, or Production Station. It includes the names of all related files as well as parameter settings for the *.fsm* and TOM Attribute settings for the Tool. When you first alter TOM Attributes on a PC, STATIONworks stores them in the Windows NT Registry, but it also copies them to the *.sta* file so that you can bring those registry settings to any machine.

  The Configuration Station is essentially a copy of the Station Template that has been configured with specific information for a particular Tool, such as the IP address of the node running that Tool.

  The Production Station is the copy of the Configuration Station rolled out to the Production File Store area.

Tools Drivers and Services:

- *.tbf*—TOM Builder File. Each contains information about a Tool driver to be built into a database with TOM Builder.
- *.mdb*—Access database. The type of file that results from building a TOM database with TOM Builder.
- *.vbp*—Visual Basic Project file. Contains STATIONworks TOM Service or TOM application code. You learn more about Services in later sections.
- *.dll*—Contains all compiled STATIONworks Services.

Next, let's take a closer look at the process of developing a Tool database.

Developing Tool Driver Database

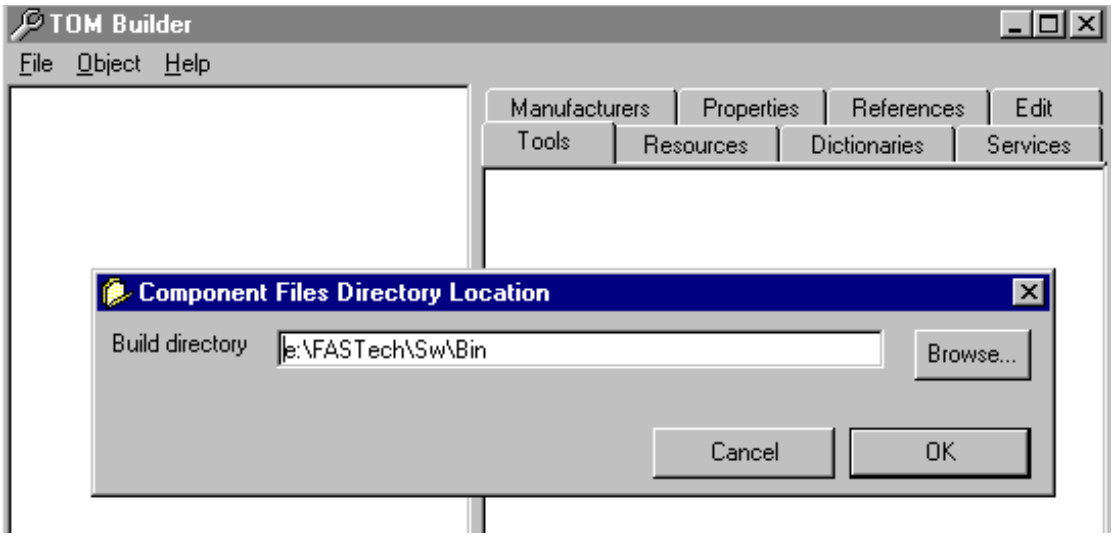
To develop a Tool database, you use **TOM Builder**, the database builder for the Tool Object Model (TOM). The Tools database that comes with STATIONworks actually contains *TOM Tool drivers*.

Each TOM Tool driver is made up to two major parts—a Tool definition and one or more TOM Services. While the *Tool definitions* describe the characteristics of a Tool/piece of equipment, the *TOM Services* define actions the Tool can take (capabilities of the equipment) and external events it can receive information about.

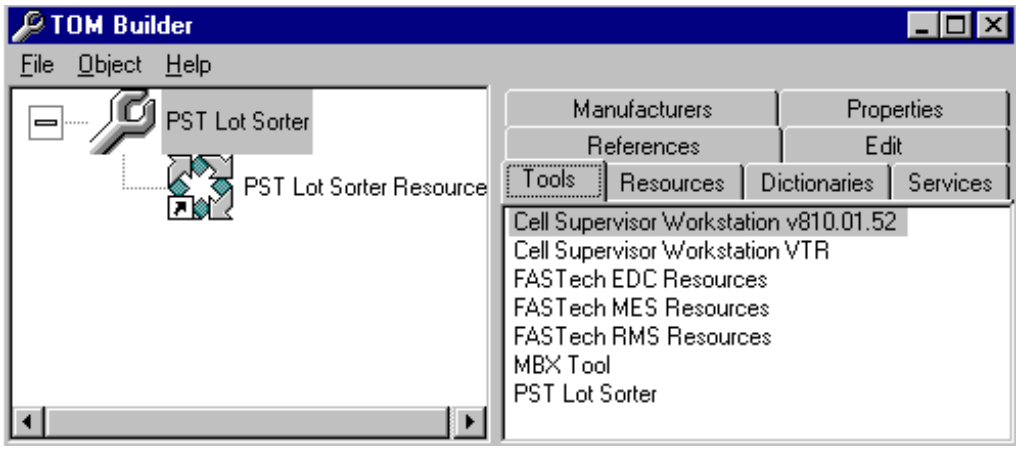
In the Tool database, Brooks provides an extensive set of Services that work with equipment that meets SECS, GEM, or VFEI standards. By joining the Tools with various Services, STATIONworks provides numerous preconfigured drivers. You can use **TOM Builder** to tweak these driver configurations so that they work with your facility's equipment.

You can also use **TOM Builder** to enhance the Tool driver database by adding new Tools. (You can develop custom Services for those Tools in Visual Basic using the Tool Object Model (TOM).)

You start developing the database by pointing **TOM Builder** to the directory location that contains the TOM Builder Files (*.tbf*s) you want to include in the database.



You then see the Tools whose drivers exist in that database displayed under the `Tools` tab.



From here, you can associate Resources with Tools (Tool drivers), assign Dictionaries to Tools, Assigns Services to Resources, and take other actions on the various Tools in the database. When you have finished making any changes you need, you build the database and create the *.mdb* file, a Microsoft Access database file that contains the Tools.

When you have finished building the database, you can use its Tools and their Services in a STATIONworks Station's state machine.

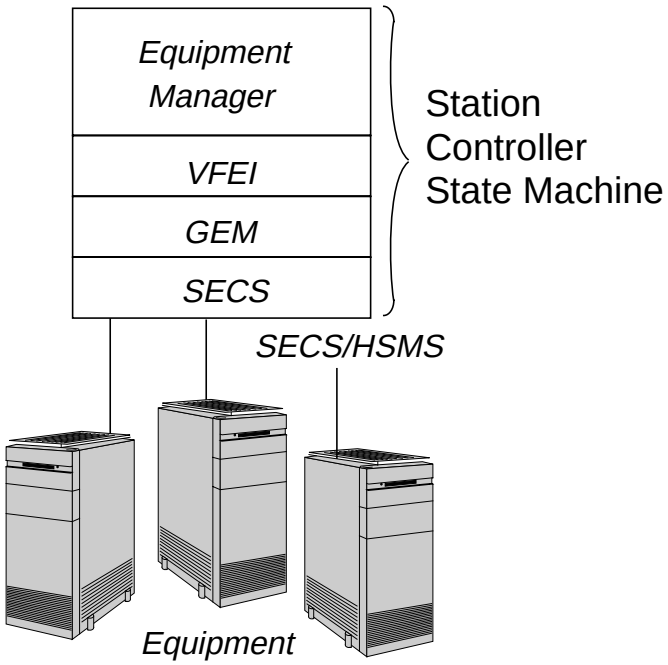
You can also view the Tools in the database without using **TOM Builder**, but by using the **TOM DB Browser** provided with STATIONworks. The **TOM DB Browser** lets you view existing Tools and Resources in the database. You can see their Services, Attribute settings, and other details.

The **TOM DB Browser** does have one somewhat surprising feature. In addition to using it as a view-only interface to the Tools database, you can also use it to mark a Tool driver as *released*, making it available for use in a state machine. Many Tool drivers ship not released because they have yet to be verified/certified on a real piece of equipment.

Next, let's take a closer look at what constitutes an equipment manager/state machine.

Developing State Machines

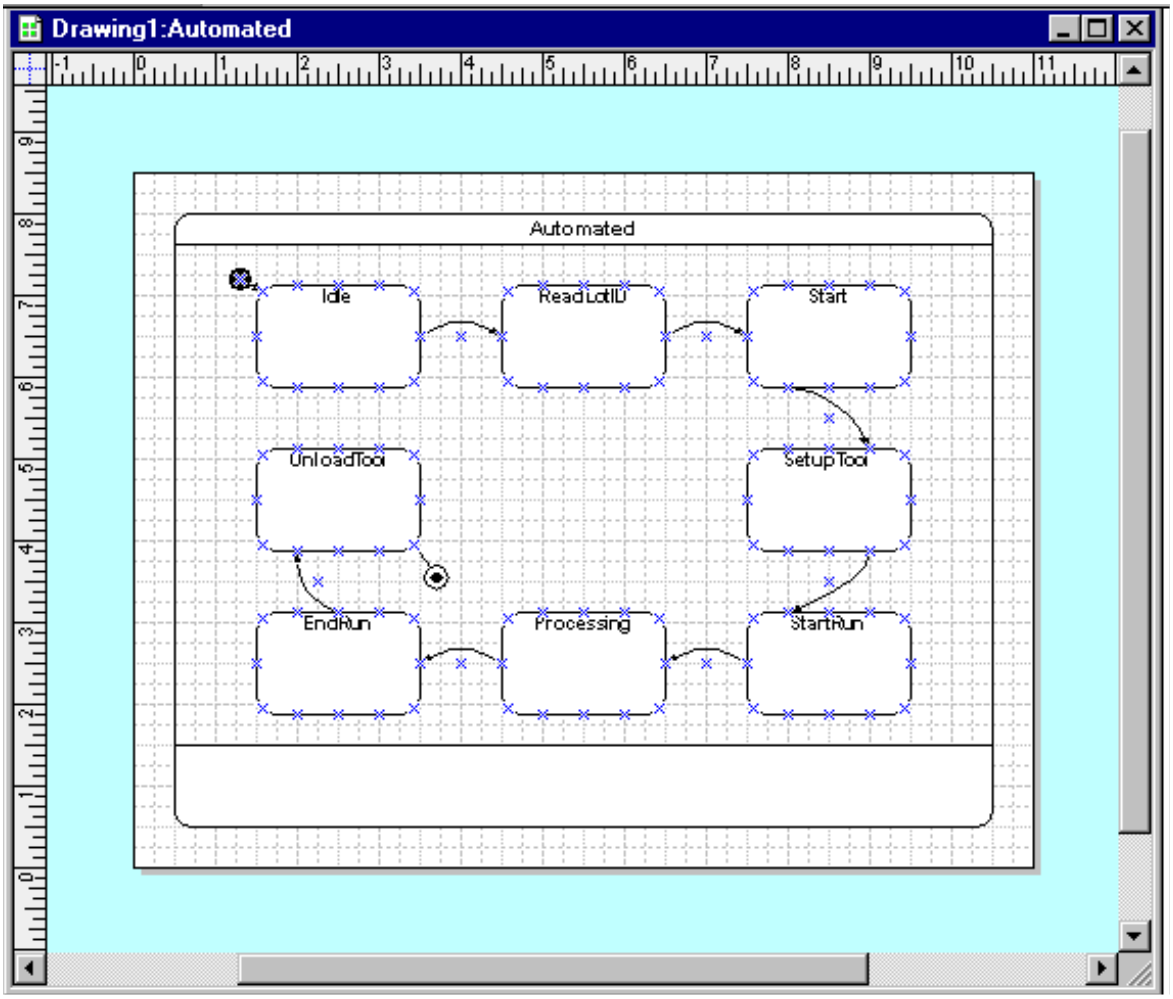
A STATIONworks equipment manager is always in the form of a state machine. This state machine is a file that controls a single piece of equipment.



The equipment manager is a piece of software you develop by creating a state machine for a piece of equipment using the STATIONworks Visual Station IDE (Integrated Development Environment).

You do not have to write more than a script's worth of code. You point and click, create state blocks in a flow chart form like the one shown

below, then select Methods to carry out actions, all through a Visio-based graphical interface.



You create transitions between the various states and choose TOM Events or other events to trigger those transitions. Alternatively, you can create an internal trigger using VBScript. You can also restrict those transitions so that they occur only when certain conditions have been met.

Creating a state machine using the IDE is straightforward enough that you can readily create one and have it up and running in hours because the IDE provides wizards:

- Action Wizard—Select a TOM Service and its Method to perform an action or write a short VBScript. You can have multiple sequential actions within a single state.

- Trigger Wizard—Select a TOM Event or another event to trigger a transition.
- Condition Wizard—Create a condition using a short VBScript that restricts when a transition can occur.

Actions you define for a state are also reusable in other states, as are triggers and conditions.

In addition, the entire state machine you develop is reusable, so you can assign the same code to multiple pieces of equipment.

Ways the Equipment Manager Works with Equipment

When controlling the equipment, the equipment manager can:

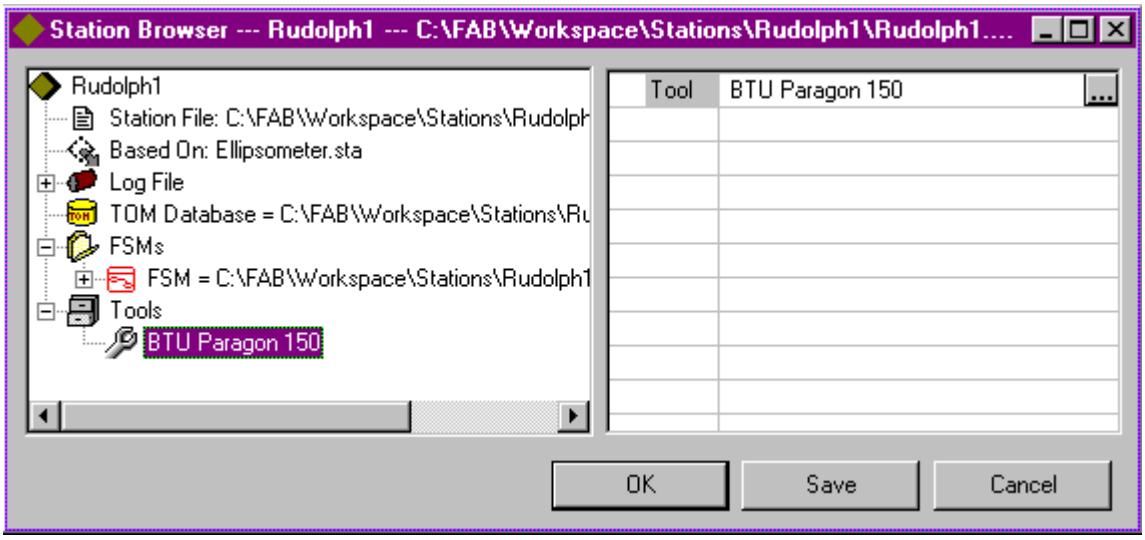
- Determine equipment has received material
- Set the equipment constants
- Send commands to the equipment, such as `ActivateResource`
- Send a recipe to the equipment
- Carry out engineering data collection
- Determine material has been removed from the equipment

To take most of these actions you employ TOM Services by selecting them from lists of Services associated with the piece of equipment the equipment manager is controlling.

Assigning State Machines to Equipment

To assign a state machine to a specific piece of equipment, you can have that equipment in the state machine code, but then how do you reuse that code?

You can assign another Tool to the Configuration Station using **SwConfig**.



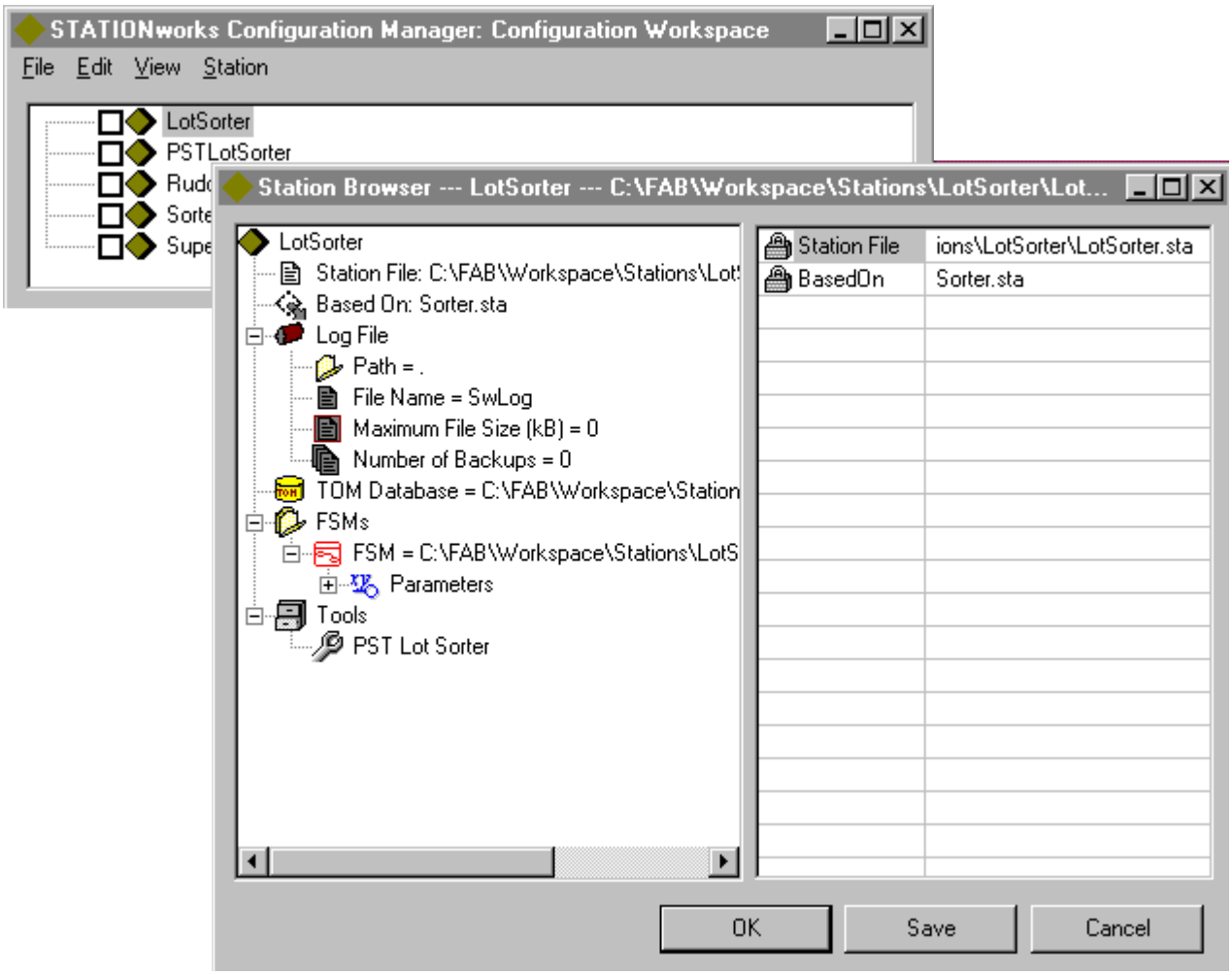
Checking Tools/State Machines into SourceSafe

While the developer creates a state machine, it is wise to insist that the state machine be checked in to a source code control system. Brooks recommends SourceSafe.

Configuring Station with SwConfig

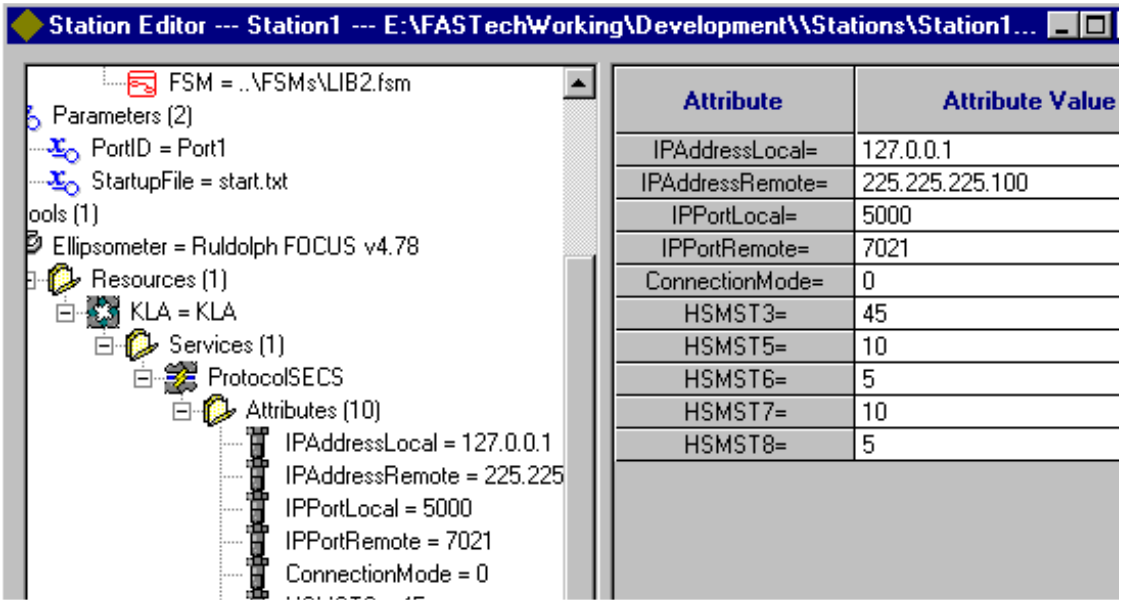
How exactly does **SwConfig** allow the engineer to prepare to run one or more Production Stations?

It lets the engineer display a list of all currently developed Station Templates, and then double click on a particular Station Template to see details on its database (*.mdb* file), finite state machine (*.fsm* file), parameter settings, specific Tools it uses, Services of the Tools, and even Attributes of the Services. .

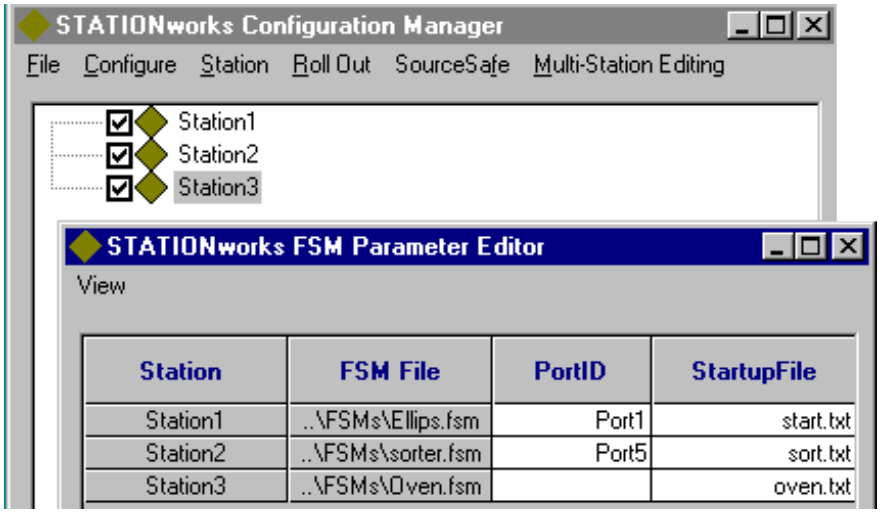


At this level, the engineer can change the particular Tool the application should work with (from PST Lot Sorter to another sorter) or change parameters and top level attributes of the application.

The next illustration shows the engineer changing settings of several Service attributes that tend to change depending on the particular Station that is running the application. For instance, the `IPAddressLocal` and `IPPortLocal` tend to change depending on the node that ends up running the Production Station, so the engineer may need to change these settings before rolling out to the Production Station.

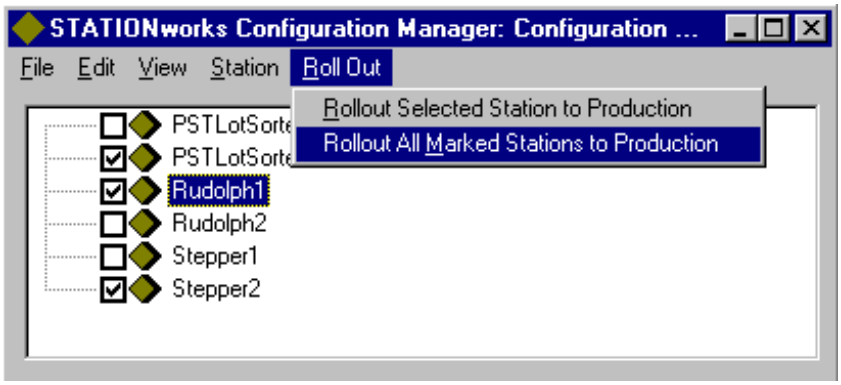


In addition, the engineer can alter configurations of multiple Stations at a time in the Configuration Workspace—altering Tool Attributes or top level parameters—before rolling those Configuration Stations out to Production Stations and running any actual equipment with them.



Rolling Out Stations to Production File Store

Once the configured Stations exist, the CIM engineer deploys them. The engineer can actually roll out the state machine for a particular Tool—to a Production Station. This CIM engineer uses **SwConfig** to roll out the configured STATIONworks Stations into the Production File Store area.

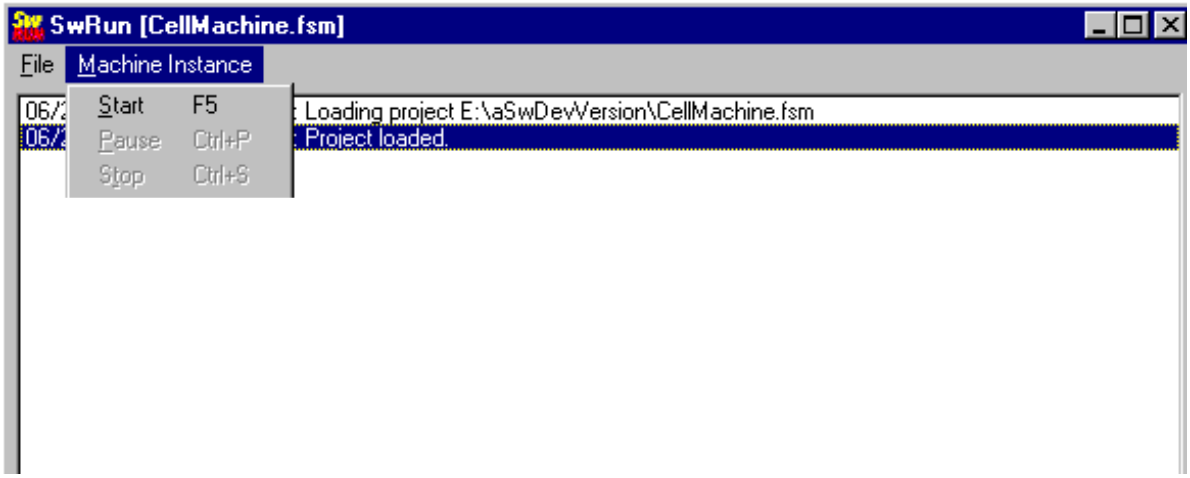


The CIM engineer also uses **SwConfig** to generate the Production File Store area where the rolled out Production Stations are stored. This separate copy of every station does not intefere with the work of the development engineer, who may need to make changes to a state machine or Tool that you are not yet ready to put into production. By keeping the Configuration Workspace and Production File Store databases distinct, you maintain strict version control over the Tools that you ultimately roll out.

Starting/Monitoring the Station

Once you have configured the Stations and rolled them out as Production Stations using **SwConfig**, you can run them.

You run Stations using **SwRun** or through the **IDE**.



Brooks recommends that you use **SwRun** at production time because it requires less memory than the IDE. However, if you are running a Production Station in the IDE, the `SwMonitor` window provides diagnostic messages that may help flush out any problems in the state machine.

Why a State Machine and Services? 2

Topics in This Chapter

This chapter presents all the components of STATIONworks, describing how you put them together in a development environment, then deploy them on the manufacturing shop floor.

Why a State Machine to Manage Equipment?

Suppose you want to open the port and establish communication with the Tool. You can take those actions with a TOM Service—in fact, there are instructions on how to do that in the *STATIONworks Service Developer's Guide*.

First you clone and execute the `Open` Method of the *ProtocolSECS* Service; then, you clone and execute the `Connect` Method of the *GemEstablishCommunications* Service.

You can take those actions in sequence by having the next action execute when the previous one completes. That sounds all well and good until the `Open` Method completes, but an error occurred while trying to open the port, so the port is not actually open. So, now you can't connect using the `Connect` Method, because the port is still closed. In addition, you need to subscribe to the event that `Open` generates and wait for that event to occur. You sometimes have to deal with completed methods, sometimes asynchronous events in the TOM Service code—without letting them collide. In addition, if an error did occur, how would you respond to that in the Service? Should you ignore it, trap it, or just raise it to the calling application? Can you then reinitiate the opening of the port?

There has to be a better way to write code that manages an event-driven system or piece of equipment. And there is—developing a finite state machine (*.fsm*) in the Visual Station IDE.

Product Evolution

Long before there were state machines, years ago, Brooks first started developing products to interface to standard equipment. At that time the focus was on raw SECS messages. You could create a driver by using raw SECS messages. You could create that driver using a UNIX-based product named Grapheq.

But customers harked for something more sophisiticated—perhaps it could be in Windows NT? So the company developed WinSECS, an NT-based system that managed the communication with SECS equipment using a library of SECS messages. You used the WinSECS control in your Visual Basic programs, then wrote those programs in a predefined structure to exchange messages with the equipment.

Brooks then developed the Tool Object Model (TOM), a revolutionary product that let you develop an object oriented equipment automation

system. You interfaced with the model by using Tools/Services supplied in a TOM Tools database. You could tweak the database to customize a Tool driver, then save and build the database. You also had the option of writing TOM Services or TOM applications in Visual Basic. But you were still functioning largely at the code level.

How could you get out of Visual Basic into a more intuitive interface? Brooks developed STATIONworks to put a graphical interface on top of TOM Tool drivers and let you add VBScripts as needed. STATIONworks works with equipment by modeling it in a state machine through an Harel-like state diagram.

What Are the Advantages of a State Machine?

Some of the advantages of a finite state machine in STATIONworks are based on the fact that it is essentially an Harel state diagram (or statechart, as explained in David Harel's article, "StateCharts: A Visual Formalism for Complex Systems" (*Science of Computer Programming 8* (1987) 231-274). Harel designed his statecharts to model event-driven systems. Harel's notation is widely used in the design of manufacturing solutions. The characteristics of his statecharts that help model those systems are included in the diagrams produced in the STATIONworks Visio-based Integrated Development Environment (IDE):

Visual Formalism:

The STATIONworks Visio-based IDE gives you a formal visual way of drawing a state machine, so you can see the relationships between states and draw the transitions that tie them together. Visio does the actual drawing for you, but you control the relationships on the screen and can readily see what you are developing. Some specific Harel-like features of the state machine in the IDE are listed below:

Modeling Equipment States:

- You can define the states of the equipment to model the piece of equipment. `Setting Up`, `Starting`, and `Loading` are some typical states.

Modeling Equipment Actions:

- In each state, the machine can have the equipment take one or more actions.

Clustering/Refinement of States:

- You can create the relationships between states—states can be clustered into a single state that contain substates. Or multiple states can exist on the same level as peers. These features allow the diagram to provide many levels of abstraction.

State Independence/Concurrency:

- Orthogonal states can work in parallel and are mutually exclusive. This feature lets you model a piece of equipment that has, for instance, two SMIF arms working simultaneously.

Transitioning Automatically on Events:

- The diagram shows how the piece of equipment (or one of its subparts) moves from state to state.
- Equipment can transition to another state on the same level in reaction to specific triggers (events) or conditions, or transition automatically when it completes the actions in a given state.
- If a state has multiple outgoing transitions, you can prioritize those transitions. As a result, when two conditions occur that would trigger different transitions, the highest priority transition actually occurs.

Multiple Transitions for Branching:

- By using multiple transitions out of a state, you can form major branches in the state machine's flow. Because of the Visio-based nature of STATIONworks, you can easily see where branching occurs and see multiple possible branches simultaneously.

Conditions/Selective Entrances to States:

- Guard conditions can *guard* entry into a given state via a particular transition. If guard conditions on a transition are not true, the state machine does not enter the state via that transition. (It can, however, enter the state through a transition whose conditions are true or through one that does not have guard conditions.)

Handling Errors:

- When an error occurs, the equipment can transition to a state that handles the error and can transition to different states for different types of errors.

Reusable States:

- Each state can have parameters associated with it. By changing the values of those parameters for a given application, you can reuse a state in a totally different application.

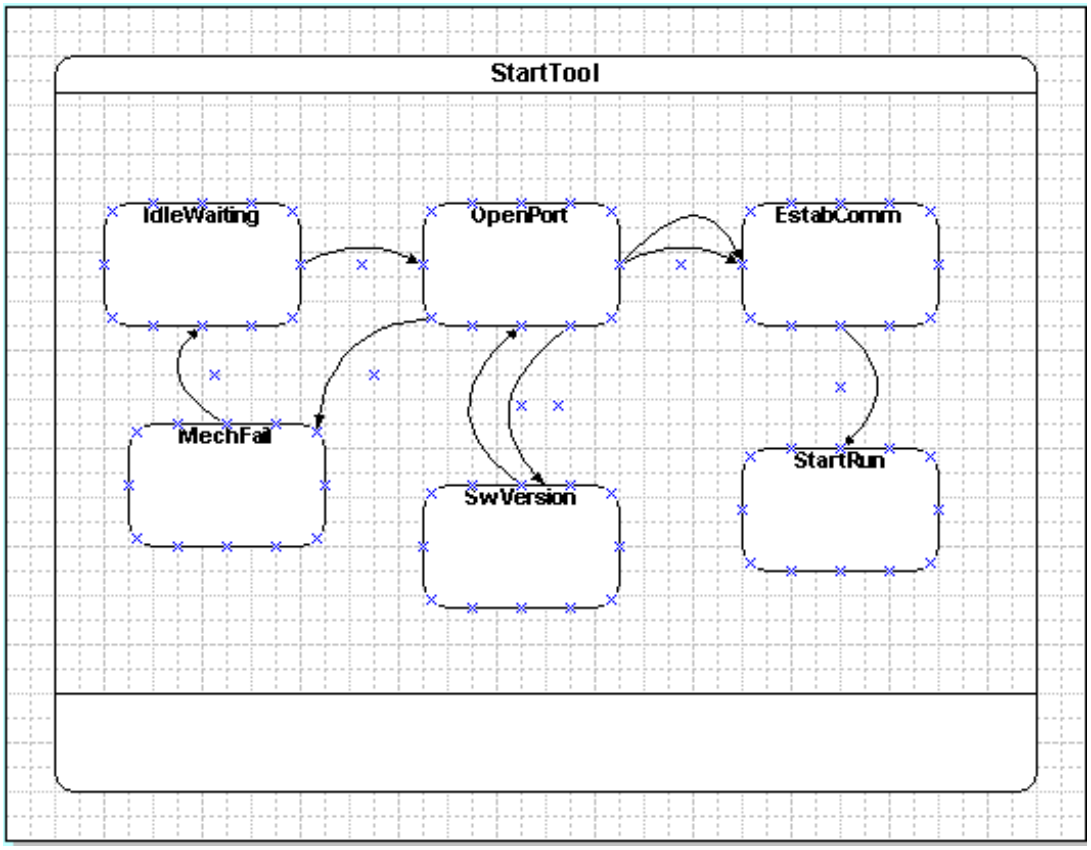
With its many levels of refinement, a statechart has become a vehicle for diagramming a complex system. STATIONworks has harnessed this vehicle in its Visio-based Visual Station IDE.

Resolving Flow Issues with State Machine

In the Visual Station IDE it is easy to resolve flow issues like those presented earlier when opening the port and establishing communication with the Tool.

For instance, in the IDE when the `Open` Method fails, the state machine can transition in multiple directions, depending on the reason for the failure. If the `Open` Method fails because the port is already open, the

state machine can transition to the next state based on the condition that the port is open.



If the `Open` Method fails because there is an equipment failure or a software error, the state machine can transition to different states to deal with those errors. Once the error is resolved, the machine can then transition back, once again, to the state where it tries to open the port.

In a visually enhanced product like STATIONworks, you can readily see each state and transition, and the relationships between them. In addition, you can easily modify the state machine because it is so readily clear what it is doing.

Let's take a quick look at how the STATIONworks state machine models equipment.

How Do State Machines Model Equipment?

State machines take action on equipment by using those same special programs that Tool drivers use—TOM Services.

What are TOM Services? They are special programs written using a Windows NT-based standard for equipment interface called the Tool Object Model—or TOM. TOM is an object oriented model that is the Brooks Automation implementation of the SEMI Object Based Equipment Modeling (OBEM) standard. TOM is the first complete implementation of that standard. In fact, TOM is the basis for the standard.

In designing STATIONworks, Brooks has leveraged industry and Microsoft standards wherever possible—building TOM on top of Microsoft's Component Object Model (COM) and taking it a step further—so it extends COM to create an object oriented automation architecture that operates in an asynchronous, real-time environment where distinct components subscribe to each others events to create an integral whole. (TOM allows some distinct components to be third party products or FACTORYworks ActiveX controls and servers.)

TOM Services do all the real work. These programs are written using specific rules that allow them to be installed with STATIONworks and work in harmony with other TOM Services. Services determine what a piece of equipment can do and the kinds of events it can respond to.

TOM was originally designed to interact with equipment, so it provides largely equipment connection Services. Brooks has expanded TOM within STATIONworks to provide a fully integrated automation system by adding Services that interact with the MES and carry out other manufacturing system functions, such as recipe management and data retrieval.

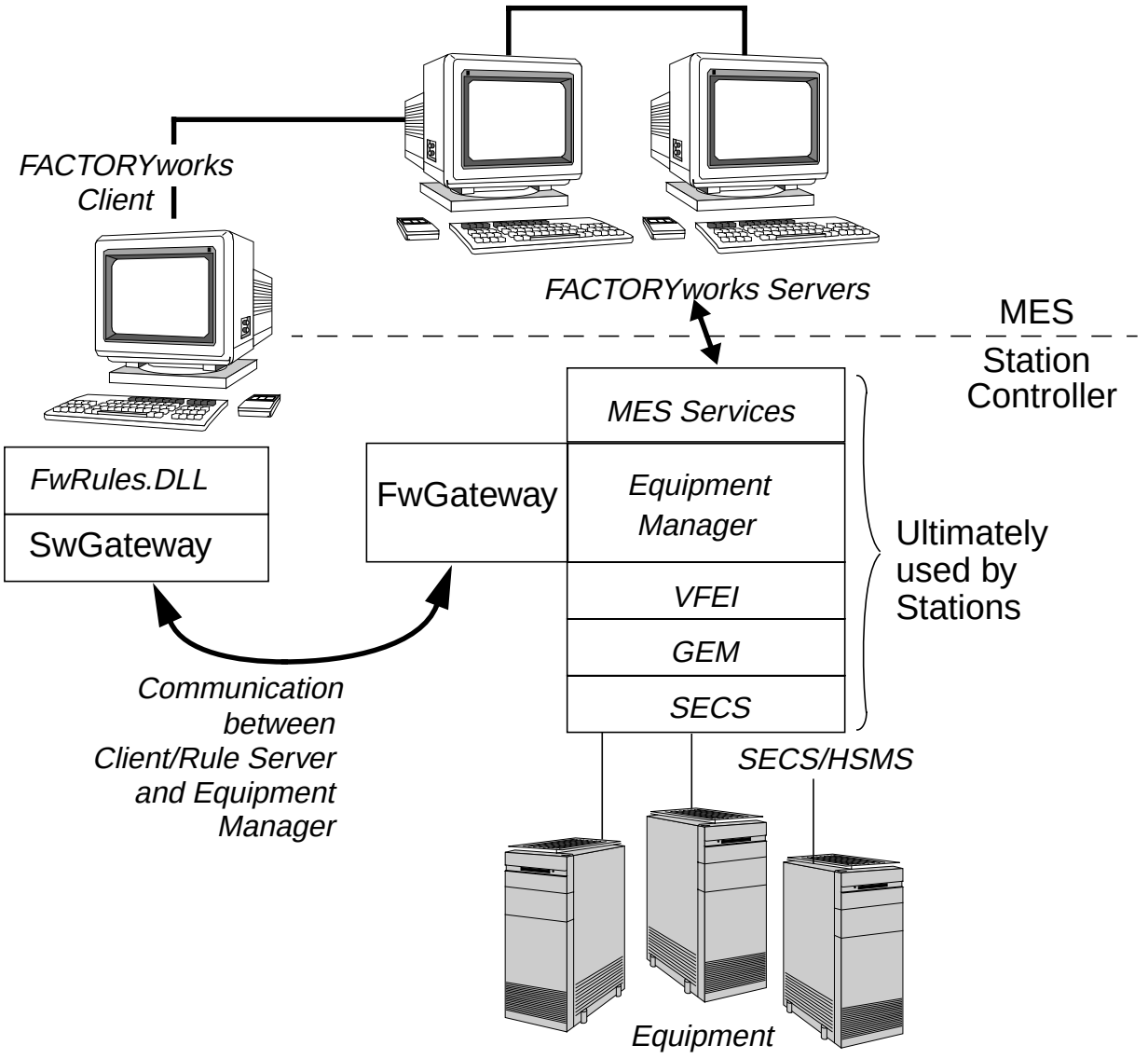
STATIONworks can also work with higher level TOM Services that carry out such functions as downloading a recipe, retrieving equipment constants, interacting with a Tool maintenance schedule, or running diagnostics on the equipment. STATIONworks provides many of these higher level TOM Services , such as *VFEIVariableManagement* (to download equipment constants) and *SecsLoopbackDiagnostic* (to test communications integrity); in addition, you can create your own higher level Services using TOM with Visual Basic.

The Services STATIONworks provides belong to several categories:

- MES Services
- Equipment Connection Services
- Host Services
- Utility Services
- Application and Custom Services

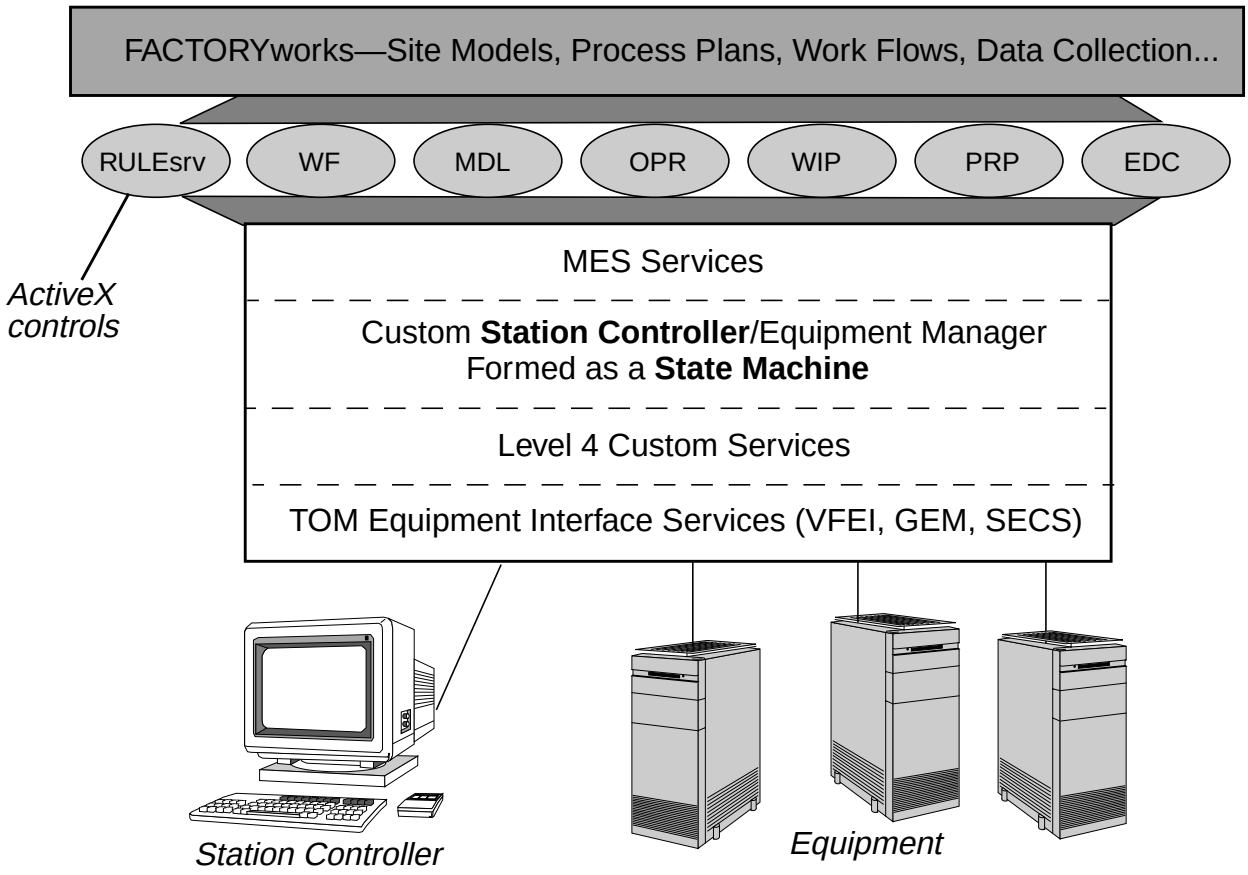
Using State Machine Equipment Managers with an MES

The state machine you develop can interact with FACTORYworks (or another MES) while controlling the equipment. While working with FACTORYworks, STATIONworks can fire off business rules run by the FACTORYworks Intelligent Client/Rule Server (RULEsrv).



In STATIONworks, you can have a custom equipment manager that talks to FACTORYworks, the FACTORYworks ActiveX controls, and

the Client/Rule Server above it and the equipment interface below it (see the illustration that follows).



Ways the Equipment Manager Works with MES

The equipment manager might interact with FACTORYworks in these ways:

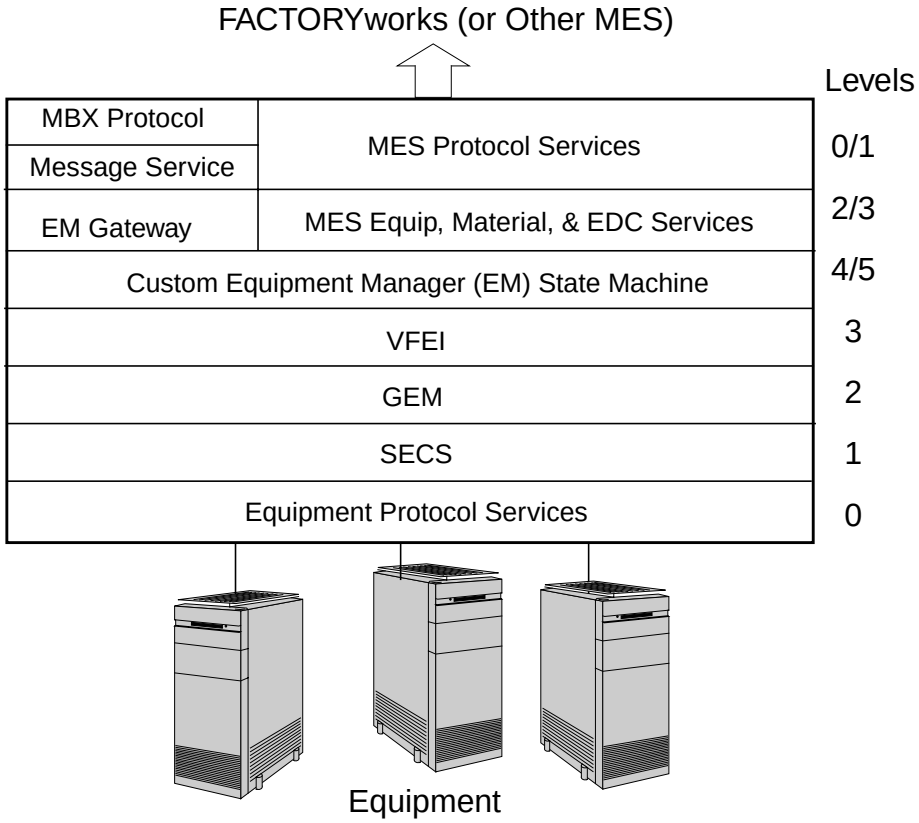
- Receive command from the Client and then send a response to the Client when the command has been executed.
- Retrieve a recipe from the Client.
- Retrieve data from the FACTORYworks database.
- Subscribe to the FACTORYworks lot or equipment updates.
- Dispatch a lot in FACTORYworks using the site model and process plan defined there.
- Execute various FACTORYworks business rules.
- Retrieve an engineering data collection plan.

MES Services Link the MES to Equipment

TOM Services work on multiple levels within the factory automation system.

At the highest level, TOM Services interact with other types of programs. For instance, the MES Services, the set of Services that handles equipment monitoring, material tracking, and engineering data collection, interact with the FACTORYworks ActiveX controls. In turn, those ActiveX controls communicate directly with the FACTORYworks Client, so you can use MES Services with FACTORYworks.

In addition, on the other end of the spectrum, MES Services integrate readily with equipment connection Services. Those Services are part of the Tool drivers on the lower levels of the hierarchy (see illustration below). They start with VFEI standard Services, and progress downward to GEM, then SECS Services, and finally to equipment protocol level Services.



The MES Services interact with several FACTORYworks ActiveX controls. They also use the same Tool Object Model (TOM) that the equipment connection TOM Services use, which means you can combine the station controller and the equipment interface in a single system that is fully integrated—from its MES (FACTORYworks) to its equipment interface—from top to bottom.

Custom Services Can Take MES Actions

When you develop your own Services, you may want to utilize the MES Services, since they provide a multitude of Methods and Events that help you customize your equipment manager.

Using Services to Connect to Equipment, Take Other Actions

Various other types of Services are available as part of STATIONworks. They are a key part of the product, but mostly operate in the background, so you don't notice them.

Equipment Connection Services Connect to Tools

Equipment connection Services connect to equipment that is GEM or SECSII compliant. Some use VFEI drivers to connect to equipment.

The VFEI, GEM, and SECS Services ultimately connect to the equipment through the *ProtocolSECS* protocol level Service. Some examples of these Services include:

- *VFEIEventManagement* to set up and manage the event reports received from a piece of equipment.
- *GemEstablishCommunications* to establish the actual connection with the equipment and set the `Communicating` attribute to `True`.
- *SecsMultiblockInquire* to requests permission to send primary SECS messages that contain more than one block.

You could even write Equipment connection Services that connect to equipment that operates on a standard other than SECS, such as PLCs.

Host Services Interact with Applications/ Message Buses

Host Services interact with CELLworks applications or message buses. For instance the *ProtocolMBX* connects a TOM Service to the FASTech products message bus, the MBX. This connection makes it possible for TOM to interact with a CELLworks application.

Similarly the *VFEItoMBX* provides a connection between a VFEI CELLworks host and a TOM Tool interface. This Service converts ASCII VFEI messages sent over the MBX to TOM Events and Methods and vice versa.

Utility Services Carry Out Tasks

Utility Services are general purpose Services that carry out such functions as Service verification and message logging. These Services do not fit into the other Service categories.

For instance, the *LOLogging* Service logs daily activity into a file that you can use to keep records and diagnose problems.

Another utility Service, the *Verification* Service, verifies all Services associated with a particular Resource.

Several other utility Services carry out similarly generic tasks.

Application Services—Expanding the Horizon

Brooks Automation provides some application Services that, for instance, collect engineering data or handle recipe management.

In addition, you can develop custom application Services to expand the capabilities of the equipment manager. For example, you might develop an alarm management or preventive maintenence Service. You can write your own application Service to take action on a higher or lower level in the STATIONworks hierarchy.

By writing your own application Services, you can expand the capabilities of STATIONworks to meet your custom needs.

Other Elements of STATIONworks

3

Topics in This Chapter

This chapter presents some other elements of STATIONworks that you may find useful:

- WinSECS
- TOM Applications
- TOM Explorer

What Is the Role of WinSECS?

WinSECS is an equipment connection program buried deep inside the Tool Object Model (TOM), so you receive this program with STATIONworks.

You do not need to work with WinSECS unless you are developing a Tool simulator. Such a simulator can be vital to debugging a complex state machine, Tool driver, or TOM Service.

In WinSECS you build a message library to indicate the messages your Tool simulator should send and receive. When you run the simulator in WinSECS, it then responds to messages by sending an appropriate response.

You can build the message library and run the simulator using the FASTsim application supplied (see the related appendix of the *STATIONworks Tool Deployment Guide*).

For communication with live equipment, you use TOM Tool drivers rather than using WinSECS. When you create custom Tool drivers, you add TOM Services to the Tool rather than having to add all the standard VFEI, GEM, or SECS messages. The Services add the standard messages for you. From there, you can customize any Attribute settings or other details that make the equipment less than standard.

What Are TOM Applications?

While the finite state machine is a STATIONworks application, a TOM application is a Visual Basic program with a graphical interface that uses the TOM object and the TOM object Methods, Properties, and Events (see the *STATIONworks Help* or *TOM Object Reference* for details). You can write such an application to interact with your equipment; however, if you develop a STATIONworks application, you generally do not need a TOM application—because STATIONworks state machines pull in the Tool Object Model and all its Services. The state machine even lets you directly access the object in scripts to take such actions as selecting another Tool or setting Tool or Service Attributes.

Using TOM Explorer as Debugger

The **TOM Explorer** and the STATIONworks **IDE Browser** are virtually identical. They both let you interface to any standard STATIONworks Tool so that it is no longer a black box—instead, you can see its Attributes, Methods, and Events displayed in a tree-like structure. You can use either one to debug STATIONworks TOM Services or Tool drivers.

The TOM Explorer, however, has some extra debugging features—you can see a separate list of the Methods executed, Errors that have occurred, and Events that have fired. You can then query those Method, Error, and Event objects for their property values while debugging your Service or Tool driver.